Recursive Agents and Landmarks Strategic-Tactical Planning (RALSTP)

Buksz, Dorian

Awarding institution:
King's College London

Recursive Agents and Landmarks Strategic-Tactical Planning (RALSTP)

Dorian Buksz

Supervisor: Prof. Derek Long

The Department of Informatics
King's College London

This dissertation is submitted for the degree of
Doctor of Philosophy

April 2024

I would like to dedicate this thesis to Professor Liviu Abramovici for introducing me to the concept of 'why?' early on in my life. May he rest in peace.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 100,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Dorian Buksz
April 2024

Acknowledgements

I am extremely grateful to my supervisor, Professor Derek Long, for his flawless feedback on anything planning-related and beyond as well as for his continuous support. My gratitude extends to the KCL Centre for Doctoral Studies for awarding me the Professor Sir Richard Trainor Scholarship to support my studies at the Department of Informatics, King's College London. I would also like to express my gratitude to Dr Daniele Magazzeni not only for introducing me to the world of AI Planning but also for providing the opportunity to work on exciting planning and robotics applications with prominent industry partners such as NASA and Ericsson Research. I am also extremely grateful to Dr Michael Cashmore for his time and patience in teaching me how robots and planners can be friends. I would also like to acknowledge and thank Associate Professor Erez Karpas for providing the temporal landmarks extraction code used in the implementation of this thesis.

Abstract

The use of AI planning beyond demonstration examples has proven to be challenging for expressive problems with numerous components. This happens primarily because the state space of a problem is prone to exponential expansion the more features are added. In such problems, current techniques either find poor-quality solutions or none at all.

Our approach, named RALSTP, identifies the agents in a problem and uses them for decompositions and relaxations that can exponentially increase the scale of solvable problems. Perhaps surprisingly, our method also increases the quality of the solutions. Our technique is domain-independent, fully automatic and PDDL compatible.

Our thesis introduces new AI Planning technical concepts along with automated extraction procedures that output data used as 'advice' for the recursive decomposition and abstraction of a planning problem. These concepts consist of formal definitions for the agents, the agent dependency relationships and classification as well as for the necessary and unnecessary static environments. A new type of 'relaxed' landmark based on the agents it may contain is introduced and used to create a novel goal clustering method based on the common landmarks found between the individual backchaining of each top-level goal. We also present a new framework for evaluating the difficulty of a planning problem according to the quantity and entanglement among the agents and the expressed dynamic and static environments.

RALSTP is evaluated on International Planning Competition (IPC) benchmark problems against a broad range of state-of-the-art planners. The evaluation shows huge benefits in scale and solution quality over the other planners, particularly in the larger, more difficult, problems.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Chapter Overview In this chapter, we introduce temporal numeric AI Planning as well as the use of decompositions for solving large planning problems. We also present an overview of our domain-independent approach for constructing efficient decompositions using problem-specific information. Furthermore, we highlight our research contributions and describe the structure of the thesis.

1.1 AI Planning

AI Planning is a branch of Computer Science that focuses on solving planning and scheduling problems commonly through automated search methods generally referred to as planners. Planners identify action sequences (called plans) in the problem state space that represent paths from a specific initial state to a desired goal state (detailed in Chapter 2 Section 2.1.1). The structure of a planning problem can be described as a domain of actions that can be applied to objects, with some of the objects becoming the active agents that will be responsible for carrying out the tasks that satisfy the problem goal. Planning problems usually offer different resource allocations between the actions that have to be performed and have alternative ways to achieve the goal. True planning problems differ from scheduling problems, as scheduling problems usually involve a notion of time or an allocation of resources which are not mandatory in all planning problems (for example in classical planning problems [56]). Also, the number of actions needed to achieve the goal of a true planning problem is not known before finding a solution.

1.2 Temporal Planning

Not all real-world planning problems can be modelled correctly with classical planning. For example, there are problems that can only be solved in a temporal framework [31]. In such a framework, the actions take time and each agent is typically constrained to perform a small collection of actions (as small as one) concurrently (detailed in Chapter 2 Section 2.1.2). The necessity of such a framework can be clearly observed in the MatchCellar planning problem [63]. This problem tackles the challenge of fixing a broken fuse in a cellar where the only sources of light are matches that can be lit. A fuse can only be fixed if there is light illuminating it for the whole duration of the fixing operation. This situation can not be modelled without a temporal framework, as the fuse must be fixed during a time interval when a match is lit and a lit match provides light only for a limited amount of time (until it burns). Modelling such problems using only the classical planning formalism does not guarantee the required concurrency between having a lit match in the cellar for illuminating the fuse at the same time and for the whole duration of the fuse fixing operation.

1.3 Numeric Planning

Classical planning formalism is too inexpressive to facilitate the straightforward modelling of many interesting problems, such as planning problems that involve numeric state variables [36]. Nevertheless, numeric planning extends the classical planning formalism with numerical preconditions, numerical effects and numerical goal conditions (detailed in Chapter 2 Section 2.1.2). An example of a planning problem where numeric planning is particularly useful is the Pandora planning problem [15, 18]. In Pandora, Autonomous Underwater Vehicles (AUVs) are required to travel to multiple underwater structures situated at distinct geographical locations with varying distances between locations. One of the goals of Pandora is to operate a minimum number of valves from all valves present in the underwater structures. Such a goal is easily expressed using the *greater than or equal to* formalism (detailed in Chapter 2 Section 2.1.2) present in numeric planning [31]. However, modelling such a problem using only classical planning formalism, while not impossible, would require expressing each possible combination of valve operations that satisfies the goal in a distinct action. This results in a very cumbersome and time-consuming process that will output a far larger and more unintuitive encoding than the one obtained with numeric formalism.

1.4 Decompositions in AI Planning

The use of AI planning for solving problems with a large number of components and detailed features has proven to be challenging [14, 59, 19, 44, 33] for current planners. This happens primarily because the state space of such problems is prone to exponential expansion the more features and components are added.

For example, in the Pandora planning problem [15], the operations are sometimes expected to last for weeks. In such situations, the state space becomes too large for current planners to solve as a single problem [15]. Using manual decompositions (detailed in Chapter 2 Section 2.1.6) in the Pandora planning problem allowed us to create multiple subproblems that had a state space with a solvable size. This allowed us to use existing planners to find solutions to the subproblems and merge those solutions into a plan for the initial problem (which was previously unsolvable due to the large state space).

The Pandora project inspired us to further investigate solving temporal and numeric planning problems that have a large state space using decompositions – a natural strategy inspired by humans who decompose everyday life problems into more convenient tasks on a regular basis. The decomposition strategy usually follows a pattern of dividing the initial problem into smaller sub-problems while identifying some kind of division of responsibility for solving the goals between the agents in order to deal with the goals locally, within smaller, more manageable, sub-problems. The use of decomposition is a standard attack strategy for solving all sorts of problems throughout most areas of computer science where solving complex large-scale problems with some kind of divide-and-conquer [13] decomposition strategy has been explored in detail over the years and always presents the same challenge: the more you isolate the components of a problem and disregard global constraints the more you risk obtaining a very poor-quality solution. As we will show in Chapter 3, past literature clearly shows that it is much easier to find ways to decompose a problem than it is to find ways to create decompositions that generate good quality solutions, as the two things are typically hard to achieve together.

However, humans are particularly good at using decompositions to obtain quality solutions for a lot of everyday problem-solving when the problems are formed of components that interact with each other and are at least effectively separable (even though they might not necessarily be fully separable). In such cases, it is clear that an optimal solution requires reasoning about all the component interactions simultaneously as the constraints that affect a part of the problem can also affect other parts of the problem, so an analysis of the solution impact on the whole problem space is required for any guarantee of optimality. In practice, humans split problems into sub-problems which are more convenient and more manageable to solve while accepting the consequences of a potentially sub-optimal solution to the overall structure of the initial problem and are content to obtain good quality solutions even if not optimal.

1.4.1 Landmarks-based Decompositions

Landmarks [42] are facts or actions that must be true at least once during any possible valid plan of a given planning problem (detailed in Chapter 2 Sections 2.1.3 and 2.1.4). Landmarks have been used for decompositions in planning by Hoffmann et al. (2004). Their approach is to create a sub-problem for every landmark present in an initial planning problem and to sequentially solve the sub-problems and concatenate their plans into the plan for the initial planning problem.

In our work, we focus on using landmarks for achieving automatic decompositions that are powerful in the separation of temporal numeric planning problems into sub-problems and at the same time lead to an efficient solution after recombination using strategies inspired by human contextual reasoning.

1.5 Domain Independent vs Domain Specific Planning

AI Planning is generally divided into domain-independent planning and domain-specific [66, 27] planning. Domain-independent planning focuses on techniques that do not require prior information about the problems they are solving and that try to use the "physics" [40] of a problem as much as possible to find a solution while domain-specific planning techniques support the inclusion of additional solving 'advice' [40] in the form of problem-specific modelling aid for finding a solution.

Even though planning problems come in different shapes and sizes, every planning problem consists of an encoding of the environment where the problem manifests along with one or more agents that interact with the environment, each other or both. In domain-independent planning, these interactions are usually modelled assuming the agents and environment as variables in order to extract and exploit the "physics" of the problem in the form of variable dependency relationships with techniques such as causal graphs [7] that are entirely generic. However, these techniques ignore the insights that can be obtained from the problem-specific interactions between the instantiated agents and the instantiated environment.

1.6 Domain Independent Decompositions using Problem-specific Information

Humans constantly use the specific structure of problems for everyday problem-solving and are particularly good at identifying the agents and agent-environment interactions in order to split complex tasks into convenient subtasks via partial or total ordering [21] to achieve satisfactory solutions. For example, in the Driverlog [63] planning problem, where drivers must board and drive trucks to transport packages and trucks to specific locations (detailed in Chapter 2 Section 2.1.5), we can easily deduce that if we have only truck and driver goals and no package goals, we can first focus on the truck goals and disregard the driver goals until all truck goals have been achieved and only start searching for the driver goals from the state where the truck goals have been achieved (while disregarding the trucks). This intuitive decomposition using partial ordering can exponentially diminish the size of the state space of the problem by dividing the initial problem into two more manageable sub-problems. However, the above ’advice’ is not useful if we also have package goals present in the problem, as focusing first on the truck goals and pursuing the other goals only from the state where all truck goals have been achieved would yield a poor-quality solution. The effort of first positioning the trucks in the final state would be wasted due to trucks being forced to abandon the final state to transport the packages. In this case, it would make sense to instead first ignore the truck and driver goals and focus on solving the package goals, then ignore the driver goals and solve the truck goals from the state where all package goals have been achieved, and lastly, solve the driver goals from the state where all trucks have been achieved (while disregarding the trucks). However, such partial orders cannot be extracted from the causal graph of Driverlog, as the graph only provides variable dependencies without considering the actual instances present in the problem.

In our thesis, we focus on domain-independent techniques for extracting and exploiting problem-specific ’advice’ from the "physics" of a temporal problem, when evaluated in the context of instantiated agent dependency relationships obtained from the interaction between agents, goals and environment, in order to decompose problems in a way that yields efficient solutions upon recombination. This is possible because, while we do not know the instantiated dependencies before attempting to solve a problem, we do know that the instantiated agents, goals and environment necessary for extracting the dependencies will be provided at the start of the search operation. Therefore, as we will show throughout the work presented in this thesis, we can identify generic patterns that occur within the interactions of instantiated agents, goals and environment in order to create techniques that exploit these patterns to automatically obtain problem-solving ’advice’. The use of this ’advice’ to create decompositions prior to search not only increases the scale of solvable temporal planning problems but also, perhaps surprisingly, increases the quality of the solutions.

1.7 Research Contributions

The focus of this thesis is on the role of agents and landmarks in a planning problem and how they can be used to identify and extract key planning problem elements, properties, and metrics in order to create data-driven algorithmic abstractions and decompositions. The obtained procedure simulates human intuition and increases the scale and solution quality of solvable planning problems. The contributions of the work presented in this thesis can be divided into four parts that are summarised below and will be presented in detail throughout the thesis:

- The first contribution is in the form of new AI planning technical concepts along with automated extraction procedures that generate information to be used as 'advice' for decomposing and abstracting planning problems. These concepts consist of formal definitions for the agents, the agent dependency relationships and classifications as well as for the necessary and unnecessary environments. A new type of landmark 'relaxed' based on the agents it may contain along with supporting PDDL encodings is also introduced.

- The second contribution consists of a new framework for evaluating the difficulty of a planning problem according to object-based difficulty metrics such as the number of agents and inactive dynamic objects, the number of types of dynamic objects and the number of static objects entangled in a planning problem.

- The third contribution comprises a detailed description of a new fully automated data-driven recursive decomposition and abstraction procedure. The procedure is guided by our difficulty metrics framework and significantly increases the solution quality of solvable planning problems. The procedure uses a novel goal clustering method based on the regular and 'relaxed' landmarks found in common between the individual backchaining of each top-level goal. The method can exponentially increase the scale of solvable planning problems that have a large state space due to numerous interacting components.

- The fourth contribution consists of an evaluation of the agent and landmarks decomposition and abstraction procedure against benchmark problems from past international planning competitions. The evaluation shows the scale and solution quality benefits of our method in comparison to the solutions obtained by a broad range of state-of-the-art temporal planners.

1.8 Thesis Layout

In Chapter 2, we introduce the concepts required for understanding the work presented in this thesis as well as the motivation behind the thesis. Chapter 3 outlines some of the previous work related to our thesis and how our work differs from existing techniques. In Chapter 4, we define and show the extraction procedure for the primary elements that will be used for constructing the decomposition procedure later used in solving planning problems. Chapter 5 provides the framework for evaluating the difficulty of a planning problem using agent-based difficulty metrics. In Chapter 6, we present the full procedure for automatically decomposing and solving a planning problem using landmarks and agents both in a high-level format and a detailed format. We then continue with Chapter 7 which presents an evaluation of the procedure described in Chapter 6. In Chapter 8, we discuss of the applicability, limitations and potential future areas of research derived from our thesis and present the conclusions.

Chapters 4, 5 and 6 represent the core of our thesis and each section in these chapters is written with a similar structure. We begin with the formal definitions of the elements presented in the respective section. We then continue with a description of the elements and procedures in the section. Afterwards, we provide an algorithm that illustrates the exact order of applying and obtaining the elements and procedures in the section. Then, we conclude with an example in the form of an instantiated description of the elements and procedures described in the section.

Chapter 2

Background and Motivation

Chapter Overview In this chapter, we provide the technical information required for a clear comprehension of the thesis as well as the experimental observations that motivated us to pursue the work presented in this thesis.

2.1 Background

Section Overview In this section, we provide a formal description of the technical concepts needed to understand the work presented in this thesis.

2.1.1 PDDL

The Planning Domain Definition Language (PDDL) [40] is an action language that attempts to standardise the syntax of AI planning languages, in a format that uses preconditions and effects for describing the applicability and outcomes of the actions in a domain. A PDDL planning model is separated into two parts: the specification of the domain (which can be seen as analogous to a class in object-oriented programming) and the problem configuration (which can be seen as an instance of a class). The domain specification provides a description of the environment in the form of propositions and actions while the problem configuration describes a particular instance of the domain for which we want to obtain the action sequence, called a plan, that transitions the initial state of the instance into a goal state of the instance.

Definition 2.1. PDDL - Let P be a finite set of propositional variables. A state, S, is a valuation over P (that is, a function $P \rightarrow \{T,F\}$). An action, a, is a pair $\{\ pre_a,\ eff_a\}$ where a is applicable in state S if $S \vDash pre_a$ and the result of applying a to S, $a[S]$, is the state S updated with the effects of a. The effects make some propositions true (add effects) and some false (delete effects). A planning problem $\Pi := \{P, A, I, G\}$ contains a set of propositional variables, P, a set of actions, A, an initial state, I and a goal, G, which is a propositional formula over P [40].

PDDL is factored into subsets of features named *requirements* [40] in order to allow the creation of planners that handle only specific parts of PDDL. Each planning problem encoding must explicitly contain the requirements it uses. A popular requirement is *typing*, which allows the ability to express the objects in a planning problem via a *hierarchical type structure* (similar to representing instances via classes with inheritance in Object-Oriented Programming).

Definition 2.2. The *lifted format* of a proposition or action is a schema including (typed) variables as parameters.

Definition 2.3. The *grounded format* of a proposition or action is a structure of the corresponding type (proposition or action) in which all the parameters have been replaced with objects of the appropriate type from the appropriate problem.

Definition 2.4. A macro action is a list of actions together with an equivalence relation between parameters of those actions in which no two parameters in any equivalence class are of different types. A grounded macro action is one in which all the actions are grounded so that the equivalent parameters are grounded with the same object. A macro action is executed by executing the list of actions in the order they appear in the list [11].

Definition 2.5. A plan π represents a solution to a planning problem Π and is formed of a sequence of grounded actions created from the actions in the action set A of a planning problem Π. The result of applying plan π to the initial state I in Π is a state where the goal G in Π is satisfied while respecting all the imposed conditions.

Definition 2.6. A *planner* is an algorithm that attempts to compute one or multiple plans of a given planning problem Π.

Definition 2.7. A procedure is regarded as *sound* if all outputs can be proved as valid with respect to considered semantics. For example, a planner is deemed sound if all found plans for compatible planning problems are valid.

Definition 2.8. A procedure is regarded as *complete* in reference to a specific property if all formulas encompassing the property can be obtained using the procedure. For example, a planner is deemed complete if, given enough time, it can construct all possible plans that satisfy the goal of compatible planning problems.

Definition 2.9. A *Relaxed Planning Problem* is a problem derived from a planning problem Π by replacing all delete lists with empty lists [8, 37].

Definition 2.10. The *Relaxed Planning Graph (RPG)* of a planning problem Π is a directed graph *(N,E)* in which the nodes in N map facts and actions from Π. The edges in E either connect nodes that map facts (tail of the edge) to nodes that map actions (head of the edge) if the mapped facts represent the preconditions of the mapped actions. The edges in E can also connect nodes that map actions (tail of the edge) to nodes that map facts (head of the edge) if the mapped facts represent add effects of the mapped actions. The nodes in an RPG are arranged into alternative fact and action layers, with the first fact layer consisting of the facts in the initial state of Π. The facts in a fact layer $F(n)$ determine which actions can appear in an action layer $A(n+1)$ (Algorithm 1 which computes in polynomial time). If all preconditions of an action a in Π are found in $F(n)$, then a is added to $A(n+1)$ (lines 6 to 8). Fact layer $F(n+1)$ will contain all facts in $F(n)$ plus all the add effects of the actions in $A(n+1)$ that were not present as facts in any of the previous fact layers (line 12). If, after adding a new fact layer to the RPG, the goal state of Π is found in the union of all fact layers in the RPG, we stop expanding the RPG (line 19). The procedure also stops when no new facts are found by expanding the graph (lines 21 and 22).

Definition 2.11. A *Fully Expanded Relaxed Planning Graph* is computed identically to a relaxed planning graph except we no longer stop the expansion when the goal state of Π is found in the union of all fact layers. Instead, we stop the expansion when all possible facts of Π are present within a fact layer.

Definition 2.12. The *Relaxed Plan* [8, 37] of a planning problem Π consists of all the actions in the RPG responsible for achieving the goal state of Π in the union of all fact layers in the RPG. The procedure for extracting the relaxed plan ((Algorithm 2 which computes in polynomial time) starts by checking if the goal of Π is reached in the RPG. If the goal was not reached in the RPG, then Π is not solvable and no relaxed plan can be extracted (lines 2 to 4). However, if the goal is reached in the RPG, the relaxed plan is computed by working backwards through the RPG. We start by searching if all the facts from the goal state G in Π, are present in $F(n-1)$ (the next to last fact layer in the RPG, lines 9 and 10). If a fact f is found in $F(n-1)$, we add f to $G(n-1)$ (line 11). $G(n-1)$ represents the next target set to be used for checking facts against its corresponding fact layer, $F(n-2)$. However, if a fact f is not found in $F(n-1)$, we add the action that made f true in $F(n)$ to the relaxed plan (always at the front of the sequence, line 14) and add the preconditions of the added action to $G(n-1)$ (line 13). We repeat the process until n = 0 (line 8).

Definition 2.13. Searching for the relaxed plan of a planning problem Π is referred to as performing a goal *Reachability Analysis*. Such an analysis is successful if we are able to find the relaxed plan of Π and unsuccessful if we are not able to find the relaxed plan Π.

Algorithm 1 Obtaining the Relaxed Planning Graph of a Planning Problem Π.

```
   Input: $\Pi$
   Output: The RPG of $\Pi$
 1: if $G$ in $\Pi$ is not in the initial state I in $\Pi$ then
 2:     n = 0
 3:     $F(n)$ = initial state $I$ in $\Pi$
 4:     add $F(n)$ to RPG
 5:     while true do
 6:         for all $a \in A$ when $A$ in $\Pi$ do
 7:             if $pre_a \in F(n)$ then
 8:                 add $a$ to $A(n+1)$
 9:             end if
10:         end for
11:         for all $a \in A(n+1)$ do
12:             add all add effects of $a$ to $F(n+1)$ if not present in any of the previous fact
                layers
13:         end for
14:         if  $F(n+1)$ not empty then
15:             add $A(n+1)$ and $F(n+1)$ to RPG
16:             if $G$ in $\Pi$ not in the union of all fact layers in RPG then
17:                 n += 1
18:             else
19:                 exit and return RPG
20:             end if
21:         else
22:             exit and return RPG
23:         end if
24:     end while
25: end if
```

Algorithm 2 Reachability Analysis - Extracting the Relaxed Plan of a Planning Problem Π.

```
Input: $\Pi$
Output: The Relaxed Plan of $\Pi$ or failure
1:  RPG = run Algorithm 1 on $\Pi$
2:  if $G$ in $\Pi$ not achieved in RPG then
3:      return failure
4:  end if
5:  relaxed-plan = empty sequence
6:  n = (number of fact layers in RPG ) - 1 // counting stars from 0
7:  G(n) = $G$ in $\Pi$
8:  while n $\neq$ 0 do
9:      for all facts $f \in G(n)$ do
10:         if $f \in$ F(n-1) then
11:             add f to G(n-1)
12:         else
13:             add the precondition $pre_a$ of action $a \in A(n)$ that has $f \in eff_a$ to G(n-1)
14:             add action $a \in A(n)$ that has $f \in eff_a$ to the front of the relaxed-plan
15:         end if
16:     end for
17:     n -= 1
18: end while
19: return relaxed-plan
```

2.1.2 PDDL 2.1

The capacity of planners for solving complex real-world problems that deal with resources and time is limited in PDDL 1.2. This led to the creation of PDDL 2.1 which extends the PDDL language with the capacity to model the temporal properties of domains [31] by using durative and instantaneous actions and allows planners to consider continuous numeric change [23].

Definition 2.14. PDDL 2.1 - In temporal domains, actions are extended to be represented by a pair of simple actions (a start and an end action), a duration constraint on the numeric variable representing the duration of the action, measuring the interval between start and end in an application of the action, and a propositional formula that must be maintained in all states between the start and end action applications. In this context, a plan is a mapping from a finite collection of action instances to a pair of real numbers, representing the start time and the duration for that action instance. A plan is valid if the duration constraints are satisfied, no actions applied at time points within epsilon of each other interfere, the states in which actions are applied satisfy the preconditions, and the resulting states are the consequences of the application of each of the actions at its appropriate start or end time.

Additional formulations of PDDL 2.1 can be found in the work of Fox and Long (2003).

Definition 2.15. A PDDL 2.1 planning problem is a tuple $\Pi := \{P, V, A, I, G\}$ that inherits all the elements and properties of a classical PDDL planning problem while also having a finite set of real variables (named fluents), V. The actions in A can also operate fluents in V. I also contains an initial assignment of values to V. G also contains numeric constraints over V that describe the goal.

Definition 2.16. A PDDL 2.1 durative action $a \in A$ is described as a tuple $a := \{\ pre_a, eff_a, dur_a\}$.

dur_a defines the duration constraint of a durative action a in the form of a conjunction of numeric constraints representative of the duration of a.

pre_a is the condition of action a that must hold for a to be applicable, with pre_a in the form of a conjunction of zero or more single conditions each formed either by a single proposition $p \in P$ or the negation of p or by a numeric constraint over V.

pre_a can be described by three disjoint subsets:

$$pre_{\vdash a}, pre_{\dashv a}, pre_{\leftrightarrow a} \subseteq pre_a.$$

$pre_{\vdash a}$ represents the conditions that must hold at the start of action a, $pre_{\leftrightarrow a}$ represents the conditions that must hold during the execution of action a and $pre_{\dashv a}$ represents the conditions that must hold the end of action a.

The numeric constraints in G, dur_a, $pre_{\vdash a}$, $pre_{\dashv a}$ and $pre_{\leftrightarrow a}$ can be formulated as $\langle f(V), op, c \rangle$ when $op \in \{\leq, <, =, >, \geq\}$, $f(V)$ is a function applied to V and c is an arbitrary constant [23].

eff_a is the effect of applying action a, with eff_a in the form of a conjunction of zero or more single effects each formed either by a single proposition $p \in P$ or the negation of p or by an operation defined as $\langle v, op, f(V) \rangle$ when $op \in$ {×=, +=, =, -=, ÷=}, $f(V)$ is a function applied to V and v is a numeric variable in V.

A numeric term is formed by an expression in Linear Normal Form (LNF) [23]. Such expressions have $f(V)$ as the weighted sum of variables plus a constant, in the form $W * V + c$ where W is a vector of constants.

eff_a can be described by seven subsets:

$$\begin{array}{l} \mathit{eff}^{+}_{\vdash a}, \mathit{eff}^{-}_{\vdash a}, \mathit{eff}^{num}_{\vdash a}, \\ \mathit{eff}^{+}_{\dashv a}, \mathit{eff}^{-}_{\dashv a}, \mathit{eff}^{num}_{\dashv a}, \\ \mathit{eff}_{\leftrightarrow a} \end{array}$$

The first six sets represent the instantaneous effects that add or remove propositions or instantaneous numeric effects at the start or at the end of action a. For example, $\mathit{eff}^{+}_{\dashv a}$ contains the propositions that have their truth value set to true at the end of action a. In PDDL2.1, the instantaneous effects enable the execution of other actions after a very small amount of time called an *epsilon separation* [31]. The last set, $\mathit{eff}_{\leftrightarrow a}$, represents a conjunction of continuous numeric effects which are applied in a continuous form during the execution of action a.

Definition 2.17. A PDDL 2.1 instantaneous action a is a special case of durative action that has its duration dur_a equal to 0, has pre_a as the only set of preconditions and has only three effects sets: $\mathit{eff}^{+}_{\vdash a}$, $\mathit{eff}^{-}_{\vdash a}$ and $\mathit{eff}^{num}_{\vdash a}$.

PDDL 2.1 actions can be concurrently applied only if they are not mutually exclusive. For example, durative actions a and a' can only have overlapping execution times only if:

$$\begin{array}{c} pre_a \cap (\mathit{eff}^{+}_{a'} \cup \mathit{eff}^{-}_{a'} \cup \mathit{eff}^{num}_{a'}) = \emptyset \\ \mathit{eff}^{+}_{a} \cap \mathit{eff}^{-}_{a'} = \mathit{eff}^{+}_{a'} \cap \mathit{eff}^{-}_{a} = \emptyset \\ \{v \in \mathit{eff}^{num}_{a}\} \cap \{v' \in \mathit{eff}^{num}_{a'}\} = \emptyset \end{array}$$

2.1.3 Landmarks

Definition 2.18. A landmark [38] L in Π is a fact f achievable in Π with the property that, $\forall$ plan π that achieves the goal G in Π when applied to the initial state I in Π, fact f appears true in at least one of the states between and including the initial state I in Π and the goal state G in Π.

Definition 2.19. A *backchaining operation* [50, 42] is a polynomial procedure which extracts new landmarks from a known landmark L in Π. The procedure marks as landmarks the intersection of the conditions of all the operators that achieve L in Π. The procedure usually starts from the facts in the goal of Π and stops either when a fact from the initial state of Π is identified or no more new landmarks are found. The procedure is sound, but not complete (some landmarks might not get identified).

2.1.4 Temporal Landmarks

Definition 2.20. An *event* e [42] is described by a tuple $e := \{\, pre_e,\, eff_e\}$ where pre_e represents the condition that must hold right before the execution of event e and eff_e represents the effect of executing event e. Durative actions with a duration larger than zero triggers two events when executed, one at the start of the action and one at the end of the action, while instantaneous actions trigger a single event.

The above definition of *event* should not be confused with other AI Planning concepts that use the same word for their identification - such as in the work of Coles and Coles (2014).

Definition 2.21. A temporal action landmark [42] *occurs(E)* is a set of events E and a time point t with the property that at least one of the events $e \in E$ is executed at time point t $\forall$ plans $\pi \in A$ with A in Π when applying plan π to I in Π.

Definition 2.22. A temporal fact landmark [42] $holds_{t_s:t_e}(bool)$ is a boolean formula $bool$ over P in Π with the property that $bool$ becomes true at time point t_s and is no longer needed to hold true at time point t_e $\forall$ plans $\pi \in A$ with A in Π when applying plan π to I in Π.

2.1.5 PDDL Running Examples

Section Overview In this section, we provide the running examples used throughout the thesis and the motivation behind selecting the examples.

2.1.5.1 Driverlog

Domain Description

The temporal version of the Driverlog planning problem [63] was used as a benchmark in the 2014 International Planning Competition (IPC). The problem consists of packages that must be delivered to various locations via trucks and involves the sub-problem of allocating a driver to a truck before the truck can transport a package between locations. The trucks can perform a package loading or unloading operation irrespective if they are boarded by a driver. The drivers have a set of paths that they can use to walk between locations, distinct from the routes used for driving. The PDDL encoding of the domain is provided in Appendix A.1.1.1.

The Driverlog temporal planning problem was selected as a running example not only because of its familiarity within the planning community but also because it has a structure similar to many real-world problems where the efficiency of concurrent task executions is required. Its main problem (deliver packages) / sub-problem (assign drivers to trucks) format yields problems very difficult to solve due to the resulting large size of their state space. Driverlog had the least solved problem instances in the last International Planning Competition where it was used as a benchmark (Temporal Track 2014) with only 8 of the 20 problem instances solved cumulatively by all temporal planners in the competition.

Thesis Driverlog Running Example 1 The DLOG-5-5-10 problem has five drivers, five trucks, ten packages and ten locations, with drivers 2,4,5, trucks 2,3,4,5 and packages 2,4,5,6,7,8,9,10 having a corresponding *(at locatable location)* goal in the goal state. This problem was chosen as a running example because it is the problem with the least number of elements, so it is easier to refer to when illustrating the work in this thesis. The PDDL encoding of this problem is provided in Appendix A.1.1.2.

Thesis Driverlog Running Example 2
The DLOG-7-7-16 problem has seven drivers, seven trucks, sixteen packages and eighteen locations, with drivers 1,2,3,4,5,6, trucks 1,2,5,7 and all 16 packages having a corresponding *(at locatable location)* goal in the goal state. This problem was chosen as a running example because it was not solved by any of the planners in IPC 2014. The PDDL encoding of this problem is provided in Appendix A.1.1.3.

Thesis Driverlog Running Example 3
The DLOG-8-8-19 problem has eight drivers, eight trucks, nineteen packages and twenty-two locations, with drivers 2,3,4,6,7, trucks 2,3,4,5,6 and all 19 packages having a corresponding *(at locatable location)* goal in the goal state. This problem was chosen as a running example because it was solved in IPC 2014 and it has sufficient elements to apply and clearly illustrate intended procedures. The PDDL encoding of this problem is provided in Appendix A.1.1.4.

2.1.5.2 Road Traffic Accident Management (RTAM)

Domain Description

Road Traffic Accident Management (RTAM) is a temporal numeric planning problem [55] that was used as a benchmark in IPC 2014 and IPC 2018. RTAM simulates a scenario where multiple simultaneous car accidents happen at distinct locations and need to be efficiently addressed. The accidents must be managed by multiple ambulances, police cars, fire brigades and tow trucks that are located at multiple hospitals, police stations, fire stations and garages. Accidents must be confirmed by the police cars. Fire brigades must extinguish any existing car fires and untrap any trapped accident victims. Ambulances must provide first aid and hospital transportation for the accident victims. Tow trucks must transport the cars involved in accidents to a garage. The numeric part of the problem consists of the multiple routes with distinct lengths between all existing locations as well as the various speeds of the rescue vehicles. Time is of the essence in such a situation, so assigning the right rescue vehicles to the right accidents via the right routes is critical. The PDDL encoding of the domain is provided in Appendix A.1.2.1.

The RTAM temporal numeric planning problem was selected as a running example not only due to its numeric component but also due to its complex real-world scenario where the efficiency of concurrent task executions is required.

Thesis RTAM Running Example 1
The RTAM_5_1_35 problem has one ambulance, one police car, one fire brigade, and three tow trucks. The problem also has thirty-three cars and thirty-five accident victims spread across five accidents at distinct locations. This problem was chosen as a running example because it is the problem with the least number of elements, so it is easier to refer to when illustrating the work in this thesis. The PDDL encoding of this problem is provided in Appendix A.1.2.2.

Thesis RTAM Running Example 2
The RTAM_5_2_35 problem has four ambulances, five police cars, three fire brigades, and seven tow trucks. The problem also has thirty-three cars and thirty-five accident victims spread across five accidents at distinct locations. This problem was chosen as a running example because it has multiple rescue vehicles of the same type, so it provides the required complexity of some of the examples in the thesis. The PDDL encoding of this problem is provided in Appendix A.1.2.3.

2.1.6 Manual Strategic Tactical Planning

Strategic-tactical planning (STP) [15] is a manual bottom-up decomposition technique that potentially increases the scale of solvable temporally expressive planning problems by dividing the difficulty of a planning problem among multiple sub-problems. STP uses a hierarchical decomposition with two levels in which the information of the sub-problems solved at the lower level, called *tactical*, is used to construct sub-problems at the higher level, called *strategic*.

The tactical level consists of a decomposition of the initial problem into sub-problems. The sub-problems are formed of clusters of top-level goals which are determined by a domain engineer from analysing the elements of the initial problem. The tactical decomposition yields sub-problems that allow temporal expressiveness and have a lower state space than the initial problem.

The information from solving the tactical problems is ported at the strategic level by encapsulating information from each tactical sub-problem's initial state, final state and plan duration as a durative macro action in an abstracted version of the initial problem at the strategic level. The abstracted version of the initial problem has a lower state space than the initial problem and is responsible for mitigating any potential constraint violations among the tactical plans.

The final solution to the initial problem is obtained by solving the abstracted strategic problem and replacing the macro actions with their corresponding tactical sub-plans obtained from the previously solved tactical sub-problems at the tactical level.

The manual strategic tactical planning technique is capable of finding solutions for problems with a state space too large for planners to solve without decompositions due to the decreased state space of the sub-problems at the tactical level and of the abstracted strategic problem. A detailed description of STP is provided in Chapter 6 Section 6.4.

2.2 Motivation: Empirical Observations of Large Scale Planning in PDDL

Section Description In this Section, we present the tactical decompositions guided by the manual strategic tactical planning technique that motivated us to pursue the work presented in this thesis. The decompositions yielded solutions better than the results of past International Planning Competitions on the tested problems.

2.2.1 Difficulty of Solving Large Scale Planning Problems

Planners have historically struggled to efficiently manage the resources of a planning problem [59], particularly when attempting to solve planning problems with a large number of elements. Current state-of-the-art PDDL temporal planners have the same issues when attempting to solve such problems, particularly if a problem contains one or more sub-problems. For example, in the Radio Base Stations (RBS) inspection planning problem [16], drones are required to perform inspection tasks on the antennas present at each station. Testing showed a correlation between the number of inspection goals and the capacity of executing the Optic [6] planner without decompositions in finding a better solution (if a solution was even found) than using Optic with the manual strategic tactical planning decomposition (Table 2.1). Using the manual strategic tactical planning decompositions, we were able to find solutions to problems that were unsolvable without decomposition.

			Purley Tactical			**Strategic Tactical Planning**		
Radio Stations	**Top-Level Goals**	**Strategic Goals**	Plan Time (seconds)	Makespan	States Evaluated	Plan Time (seconds)	Makespan	States Evaluated
1	4	1	110.08	**23.056**	1780	**8.03**	48.363	**155**
2	8	2	572.15	78.731	5895	**16.19**	**74.889**	**312**
3	12	3	949.78	102.289	8538	**24.8**	**77.785**	**469**
4	16	4	1206.67	352.145	9728	**32.96**	**325.072**	**639**
5	20	5	Failed	Failed	Failed	**41.56**	**365.684**	**796**
6	24	6	Failed	Failed	Failed	**50.52**	**392.21**	**953**
7	28	7	Failed	Failed	Failed	**77.11**	**440.574**	**7848**
8	32	8	Failed	Failed	Failed	**82.37**	**440.574**	**7449**
9	36	9	Failed	Failed	Failed	**91.56**	**444.651**	**7189**
10	40	10	Failed	Failed	Failed	**161.2**	**710.162**	**27124**

Table 2.1 Optic Purely Tactical Vs Optic STP on a set of problems that have from 1 to 10 radio stations, 4 inventory-mapping goals per station and 6 drones with identical configurations [16]

.

Another example is the Driverlog planning problem, where we can observe that the planers in IPC 2014 generally stop finding solutions to problems that have an increased number of initial state facts and goals relative to the number of initial state facts and goals in the solved problems.

2.2.2 Reducing the Difficulty and Solving Large Scale Problems Using Decompositions

The following examples showcase the effects of applying a tactical decomposition guided by the manual strategic tactical planning approach to the Driverlog planning problem. We first used the DLOG-7-7-16 benchmark problem, as it was not solved by any of the planners that participated in IPC 2014. In effect, we decomposed the DLOG-7-7-16 into 7 sub-problems that each have one distinct driver along with its initial state facts and goals, one distinct truck along with its initial state facts and goals and among which we evenly split the 16 packages along with their initial state facts and goals. The resulting subproblems have smaller state spaces in comparison to the initial problem due to having far fewer elements than the initial problem. Considering that no trucks, drivers and packages are present in more than one sub-problem, solving all sub-problems and merging their plans will provide a valid plan for the DLOG-7-7-16 planning problem.

We attempted to solve the DLOG-7-7-16 sub-problems using four robust and well-known (battle-tested) temporal numeric planners that employ different solving techniques: Optic [6], Itsat [49], Temporal Fast Downward (TFD) [26] and Yahsp3 [64] on a Dell XPS 15 9560 laptop with 32GB total and unrestricted RAM and a threshold of 120 seconds per problem. The results in Table 2.2 show that Optic, Itsat, and Yahsp3 were able to find solutions to all sub-problems while TFD was able to solve only one of the sub-problems.

Problem Name	Decomposition	Makespan				Plan Time (seconds)			
		Optic	Itsat	TFD	Yahsp3	Optic	Itsat	TFD	Yahsp3
Sub-problem 1	driver1 truck1 package1 package2 package3	292.016	280.24	-	416	0.66	2.23	-	0.002
Sub-problem 2	driver2 truck2 package4 package5 package6	334.017	354.31	-	672	1.27	3.93	-	0.002
Sub-problem 3	driver3 truck3 package7 package8	200.011	160.17	-	242	0.11	0.82	-	0.002
Sub-problem 4	driver4 truck4 package9 package10	150.009	190.21	-	150	12.84	2.5	-	0.002
Sub-problem 5	driver5 truck5 package11 package12	300.017	300.26	-	444	0.44	1.83	-	0.002
Sub-problem 6	driver6 truck6 package13 package14	175.01	185.14	165.013	245	0.36	1.63	0.008	0.002
Sub-problem 7	driver7 truck7 package15 package16	167.01	147.13	-	137	17.3	0.48	-	0.003
DLOG-7-7-16	**All drivers trucks and packages**	**334.017**	**354.31**	-	**672**	**32.98**	**13.42**	-	**0.015**

Table 2.2 Results of the DLOG-7-7-16 decomposition.

The results show that using tactical decompositions in order to reduce the initial state facts and goals parsed by the planners allowed us to solve a Driverlog problem that was previously not solved by any planners during the international planning competition where it was last used as a benchmark.

If we apply the same decomposition approach to the DLOG-8-8-19 benchmark problem under the same conditions as in the DLOG-7-7-16 example, we again obtain solutions for all sub-problems when executed with the Optic, Itsat, and Yahsp3 planners while TFD is able to solve only one of the sub-problems (Table 2.3).

The DLOG-8-8-19 planning problem is one of the benchmark problems solved in IPC 2014, with 1021.54 being the best makespan achieved in the competition. The results in Table 2.3 show that using decompositions in order to reduce the initial state facts and goals parsed by the planners allowed us to obtain a solution with less than half of the cost as the one obtained in IPC 2014 with three out of the four planners tested on our decomposition.

The individual results in both the DLOG-7-7-16 and DLOG-8-8-19 test problems (Tables 2.2 and 2.3) show that the Optic and Itsat planners obtained similar makespans while Yahsp3 obtained considerably inferior makespans in comparison to Optic and Itsat. TFD was the worst performer by far, as it was able to solve only one sup-problem from each example, with makespans similar to Optic and Itsat.

Problem Name	Decomposition	Makespan				Plan Time (seconds)			
		Optic	Itsat	TFD	Yahsp3	Optic	Itsat	TFD	Yahsp3
Sub-problem 1	driver1 truck1 package1 package2 package3	253.013	263.25	-	277	0.25	3.5	-	0.006
Sub-problem 2	driver2 truck2 package4 package5 package6	344.016	324.3	-	398	0.89	1.03	-	0.006
Sub-problem 3	driver3 truck3 package7 package8 package9	324.018	332.3	-	456	1.12	1.096	-	0.006
Sub-problem 4	driver4 truck4 package10 package11	280.014	276.2	-	432	103.82	1.31	-	0.006
Sub-problem 5	driver5 truck5 package12 package13	97.007	89.12	-	127	0.37	1.416	-	0.004
Sub-problem 6	driver6 truck6 package14 package15	380.02	410.31	-	492	1.12	3.09	-	0.004
Sub-problem 7	driver7 truck7 package16 package17	170.009	170.17	-	194	0.04	1.43	-	0.002
Sub-problem 8	driver8 truck8 package18 package19	145.008	155.11	145.01	165	0.17	0.55	0.008	0.005
DLOG-8-8-19	**All drivers trucks and packages**	**380.02**	**410.31**	-	**492**	**107.78**	**167.982**	-	**0.039**

Table 2.3 Results of the DLOG-8-8-19 decomposition.

Even though the overall results in the examples above are encouraging both in terms of scale and solution quality, the tactical decompositions were obtained from human intuition and would require a domain engineer to replicate in other problems. This motivated us to seek a procedural approach for creating such decompositions which resulted in a novel method based on reachability analysis for the algorithmic decompositions of the Driverlog planning problem and other similarly structured problems.

In the next chapters, we will formally define and show the procedure for the automatic identification and extraction of the elements needed to construct efficient decompositions. We will also present a procedure that uses the extracted elements to algorithmically decompose and solve all IPC 2014 Driverlog benchmark planning problems (as well as other similarly structured problems) without human intervention and with solutions better than the cumulative results of all planners that participated in the International Planning Competition.

2.2.3 Solution Quality Impact of Decompositions Based on Proposition Similarity

In the Pandora planning problem [15], Autonomous Underwater Vehicles (AUVs) are required to perform inspection tasks and valve-turning tasks on large sub-sea structures called manifolds that can be located at large distances from each other. The structure of the Pandora environment presented an opportunity to manually decompose the planning problem based on an intuitive observation of the geographical locality of manifolds in order to efficiently manage the AUVs in achieving their tasks.

However, planners struggle to efficiently identify such strategies on their own, particularly in large problems. This happens partially because of the previously outlined difficulty caused by a large number of initial state facts and goals and partially because of the diminished capacity of planners to extract contextual meaning from the PDDL-encoded data representation of real-world environments necessary for data clustering decomposition strategies.

An example of such a scenario can also be observed in the RTAM planning problem. To highlight the issue, we designed a tactical decomposition strategy similar to the one in the DLOG-7-7-16 example which we compared with regular solving. The decomposition strategy consists of creating multiple sub-problems with one ambulance, one tow truck, one police car and one fire brigade and spreading the cars and accident victims across the sub-problems according to their locations in the initial state. The test problems have been created based on the RTAM_5_1_35 IPC 2014 benchmark problem which has simultaneous accidents happening at five distinct locations. The test consists of how a problem that contains the car and accident victim facts and goals of any two of the five accident locations in RTAM_5_1_35 is better solved by two rescue groups each formed of the facts and goals of a single ambulance, a single tow truck, a single police car and a single fire brigade with no overlapping rescue vehicles among the two groups. We have limited the test to the facts and goals of two locations per problem as more would have prevented the majority of the sample planners from finding a solution.

We first solved the ten possible double-location problems representing all possible combinations of two out of five locations in the RTAM_5_1_35 planning problem without any decomposition and with both rescue groups added to them. Each double-location problem was given a threshold of 900 seconds. We then decomposed each double-location problem into two sets of single-location sub-problems (Figure 2.1).

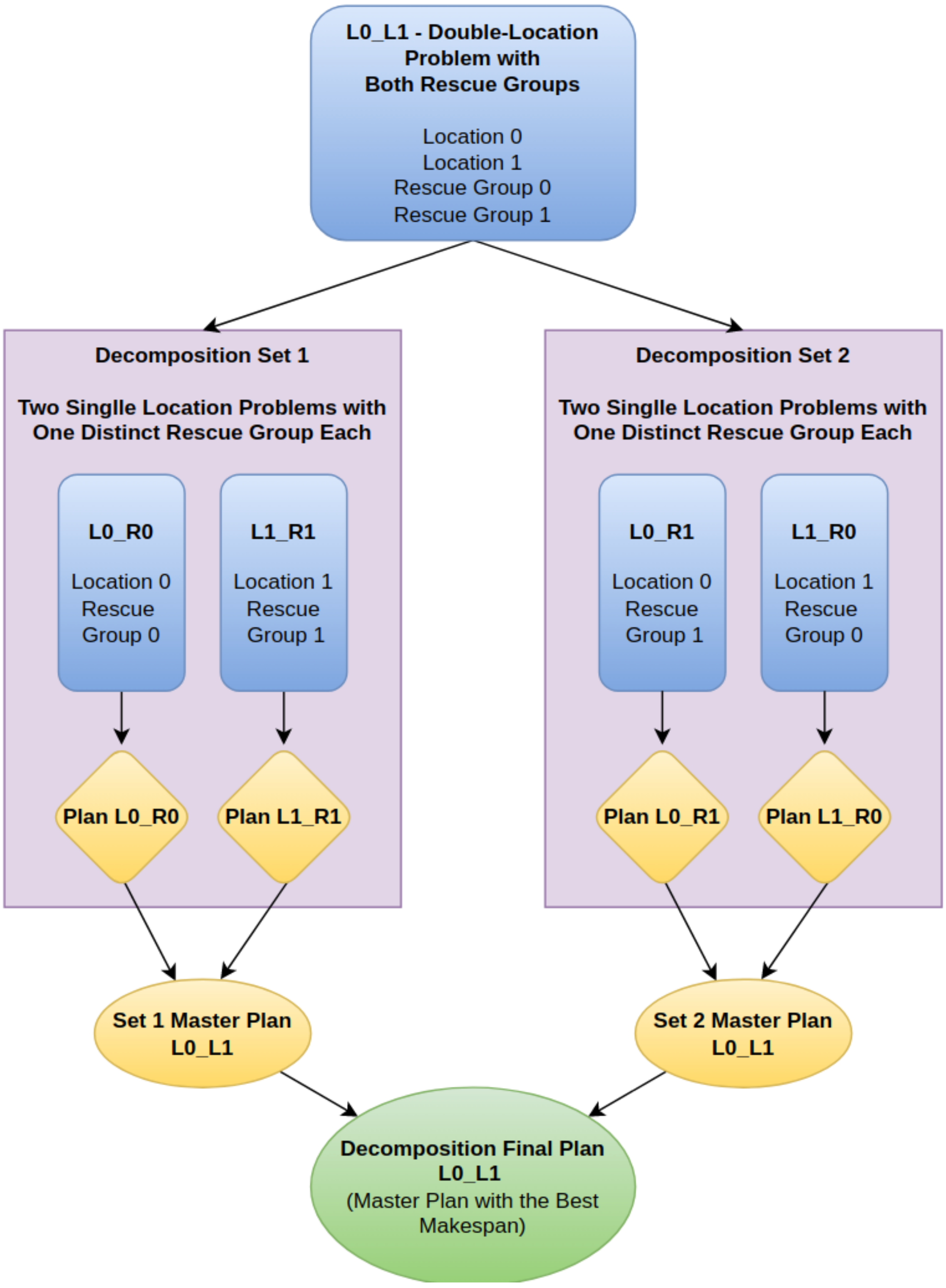

L0_L1 - Double-Location Problem with Both Rescue Groups
Location 0
Location 1
Rescue Group 0
Rescue Group 1
Decomposition Set 1
Two Singlle Location Problems with One Distinct Rescue Group Each
L0_R0
Location 0 Rescue Group 0
L1_R1
Location 1 Rescue Group 1
Plan L0_R0
Plan L1_R1
Decomposition Set 2
Two Singlle Location Problems with One Distinct Rescue Group Each
L0_R1
Location 0 Rescue Group 1
L1_R0
Location 1 Rescue Group 0
Plan L0_R1
Plan L1_R0
Set 1 Master Plan L0_L1
Set 2 Master Plan L0_L1
Decomposition Final Plan L0_L1
(Master Plan with the Best Makespan)

Fig. 2.1 Decomposing a double-location problem into two sets of single-location problems. The final plan of the decomposition is the master plan with the best makespan.

Each set contains two single-location sub-problems (one for each location in the double-location initial problem) and each sub-problem in a set was added only one of the two rescue groups, with no sub-problem having the same rescue group and location combination as any other sub-problem in any of the two sets. Each single location sub-problem was given a threshold of 120 seconds. The plans of the sub-problems in a set were merged to form a master plan representative of the cars and accident victims at both locations and both rescue groups in the initial double-location problem. The master plans of each of the two sets were compared with each other and the one with the best makespan was marked as the final plan for the decomposition approach.

The overall results show that the decomposition approach yielded better solutions in 73% of cases where the planners solved the double location problem as well as several orders of magnitude faster planning times in almost all cases. The results when using the Optic planner (Table 2.4) show that 50% of the problems have better results with decompositions and 50% of the problems have better results using regular solving.

Optic - Single Location				
Problem ID	Location	Rescue Group	Makespan	Plan time (seconds)
L0_R0	0	0	282.532	1.07
L0_R1	0	1	287.025	1.79
L1_R0	1	0	308.366	1.04
L1_R1	1	1	257.356	1.15
L2_R0	2	0	348.706	0.99
L2_R1	2	1	290.532	1.86
L3_R0	3	0	193.693	0.47
L3_R1	3	1	189.693	0.77
L4_R0	4	0	340.066	1.31
L4_R1	4	1	265.699	3.68

Optic - Double Location						
Problem ID	Location	Rescue Group	Makespan	Plan time (seconds)	Best Option	Best Option Makespan
L0_L1	0 & 1	0 & 1	276.697	114.63	L0_L1	276.697
L0_L2	0 & 2	0 & 1	321.705	70.29	L0_R0 & L2_R1	290.532
L0_L3	0 & 3	0 & 1	224.2	35.82	L0_L3	224.2
L0_L4	0 & 4	0 & 1	260.7	107.01	L0_L4	260.7
L1_L2	1 & 2	0 & 1	344.378	56.23	L1_R0 & L2_R1	308.366
L1_L3	1 & 3	0 & 1	269.369	23.17	L1_R1 & L3_R0	257.356
L1_L4	1 & 4	0 & 1	238.362	71.69	L1_L4	238.362
L2_L3	2 & 3	0 & 1	304.549	691.2	L2_R1 & L3_R0	290.532
L2_L4	2 & 4	0 & 1	352.71	318.95	L2_R1 & L4_R0	340.066
L3_L4	3 & 4	0 & 1	226.377	30.93	L3_L4	226.377
Best With Decomposition						**50%**

Table 2.4 Results of the RTAM_5_1_35 decompositions solved with the Optic planner.

The results using the Itsat planner (Table 2.5) are considerably better when using decompositions, with 70% of the problems having better solutions when decomposed into two sub-problems.

Itsat - Single Location				
Problem ID	Location	Rescue Group	Makespan	Plan time (seconds)
L0_R0	0	0	250.25	7.2425
L0_R1	0	1	230.28	5.47627
L1_R0	1	0	210.14	9.40927
L1_R1	1	1	210.18	7.19469
L2_R0	2	0	210.21	5.95224
L2_R1	2	1	200.14	2.15037
L3_R0	3	0	105.15	2.9216
L3_R1	3	1	95.11	2.57688
L4_R0	4	0	185.45	73.6311
L4_R1	4	1	185.41	31.8131

Itsat - Double Location						
Problem ID	Location	Rescue Group	Makespan	Plan time (seconds)	Best Option	Best Option Makespan
L0_L1	0 & 1	0 & 1	275.44	363.719	L0_R1 & L1_R0	230.28
L0_L2	0 & 2	0 & 1	245.15	592.741	L0_R1 & L2_R0	230.28
L0_L3	0 & 3	0 & 1	180.18	48.2217	L0_L3	180.18
L0_L4	0 & 4	0 & 1	245.34	411.889	L0_R1 & L4_R0	230.28
L1_L2	1 & 2	0 & 1	225.13	426.214	L1_R0 & L2_R1	210.14
L1_L3	1 & 3	0 & 1	165.12	23.7897	L1_L3	165.12
L1_L4	1 & 4	0 & 1	250.38	240.49	L1_R0 & L4_R1	210.14
L2_L3	2 & 3	0 & 1	190.18	64.2576	L2_L3	190.18
L2_L4	2 & 4	0 & 1	210.13	843.714	L2_R1 & L4_R0	200.14
L3_L4	3 & 4	0 & 1	195.41	438.796	L3_R0 & L4_R1	185.41
Best With Decomposition						**70%**

Table 2.5 Results of the RTAM_5_1_35 decompositions solved with the Itsat planner.

The results when using the Yahsp3 planner (Table 2.6) show overwhelmingly better results when using decompositions, with 100% of the problems having better results when decomposed into two sub-problems.

Yahsp3 - Single Location				
Problem ID	Location	Rescue Group	Makespan	Plan time (seconds)
L0_R0	0	0	378.5	0.005
L0_R1	0	1	375.01	51.4
L1_R0	1	0	324.59	11.787
L1_R1	1	1	267.09	5.986
L2_R0	2	0	488.68	0.004
L2_R1	2	1	595.5	0.004
L3_R0	3	0	277.68	36.243
L3_R1	3	1	344.5	42.09
L4_R0	4	0	438	17.611
L4_R1	4	1	325.68	0.003

Yahsp3 - Double Location						
Problem ID	Location	Rescue Group	Makespan	Plan time (seconds)	Best Option	Best Option Makespan
L0_L1	0 & 1	0 & 1	436.67	0.011	L0_R1 & L1_R0	375.01
L0_L2	0 & 2	0 & 1	673.68	0.434	L0_R1 & L2_R0	488.68
L0_L3	0 & 3	0 & 1	450.68	0.386	L0_R1 & L3_R0	375.01
L0_L4	0 & 4	0 & 1	1005.84	0.014	L0_R0 & L4_R1	378.5
L1_L2	1 & 2	0 & 1	630.43	0.011	L1_R1 & L2_R0	488.68
L1_L3	1 & 3	0 & 1	371.68	0.008	L1_R1 & L3_R0	277.68
L1_L4	1 & 4	0 & 1	620.34	131.092	L1_R0 & L4_R1	325.68
L2_L3	2 & 3	0 & 1	772.5	0.262	L2_R0 & L3_R1	488.68
L2_L4	2 & 4	0 & 1	567.68	0.011	L2_R0 & L4_R1	488.68
L3_L4	3 & 4	0 & 1	344.68	0.009	L3_R0 & L4_R1	325.68
Best With Decomposition						**100%**

Table 2.6 Results of the RTAM_5_1_35 decompositions solved with the Yahsp3 planner.

TFD was able to solve only one single location sub-problem and none of the double location problems so the results are not providing any insight into the difference in solution quality between the two solving techniques.

Temporal Fast Downward - Single Location				
Problem ID	Location	Rescue Group	Makespan	Plan time (seconds)
L0_R0	0	0	-	-
L0_R1	0	1	-	-
L1_R0	1	0	-	-
L1_R1	1	1	-	-
L2_R0	2	0	-	-
L2_R1	2	1	-	-
L3_R0	3	0	262.21667	2.883669
L3_R1	3	1	-	-
L4_R0	4	0	-	-
L4_R1	4	1	-	-

Temporal Fast Downward - Double Location						
Problem ID	Location	Rescue Group	Makespan	Plan time (seconds)	Best Option	Best Option Makespan
L0_L1	0 & 1	0 & 1	-	-	-	-
L0_L2	0 & 2	0 & 1	-	-	-	-
L0_L3	0 & 3	0 & 1	-	-	-	-
L0_L4	0 & 4	0 & 1	-	-	-	-
L1_L2	1 & 2	0 & 1	-	-	-	-
L1_L3	1 & 3	0 & 1	-	-	-	-
L1_L4	1 & 4	0 & 1	-	-	-	-
L2_L3	2 & 3	0 & 1	-	-	-	-
L2_L4	2 & 4	0 & 1	-	-	-	-
L3_L4	3 & 4	0 & 1	-	-	-	-
Best With Decomposition						**N/A**

Table 2.7 Results of the RTAM_5_1_35 decompositions solved with the TFD planner.

The individual results of the RTAM test problems (Tables 2.4, 2.5, 2.6 and 2.7) show that the Itsat planner obtained the best makespans, Optic the second-best makespans and Yahsp3 the third-best makespans. TFD was again the worst performer, as it was able to solve just one single location sub-problem with a makespan similar to the one obtained by Yahsp3 in the same sub-problem.

The overall results show that a tactical decomposition guided by the similarities of locations of the cars and accident victims clearly outperforms the solutions of the sample state-of-the-art temporal planners when used without decomposition. However, as in the case of the Pandora [15] and RBS [16] planning problems, the decomposition in the above example was only possible due to our intuitive contextual understanding of the environment. This motivated us to pursue a domain-independent context-extracting procedure from problem-specific PDDL data which culminated in a novel method for algorithmically extracting contextual meaning from PDDL-encoded data with no human input. The method is based on the similarities of landmarks between top-level goals in order to decompose the goals based on their location or based on other types of meaningful context in the initial state and beyond the initial state of a planning problem. The method obtains better results than the cumulative results of the RTAM benchmark problems solved by all planners that participated in the International Planning Competition and will be detailed in the following chapters.

Chapter 3

Related Work

Chapter Overview In this chapter, we present some of the previous decomposition approaches to AI planning and consider how they relate and compare to the techniques presented in our thesis.

3.1 Domain Independent Decompositions and Abstractions

Divide-and-conquer approaches have been successfully applied in various areas of computer science [13]. Divide-and-conquer decomposition algorithms vary from simple binary-search, merge sort and quicksort algorithms to integer and matrix multiplication algorithms as well as cryptography algorithms. The success and long heritage of decompositions have led to an interest in creating AI Planning decomposition techniques since the earliest foundations of the topic.

3.1.1 Abstractions and Macro Actions

Abstractions in planning have been explored by Sacerdoti (1974), who represented a STRIPS [28] problem domain as an abstraction space hierarchy from higher to lower level details [52] that provided increased problem-solving capabilities. Tenenberg (1988) defines the abstraction of a concrete representation *r* as a construction that keeps only the properties of the highest importance from *r* and ignores the lower importance properties of *r* [61]. Tenenberg (1988) performed a detailed analysis of abstractions in planning in order to simplify the action reasoning of agents in a complex environment [61]. His analysis has two main directions. The first direction looks at ignoring some of the conditions required for actions when applied at the abstract level (the conditions that are considered details). Such abstractions imply an upward-solution property [61]: if a concrete (base) level solution *s* exists, then an abstraction of *s* must also exist at a higher (abstracted) level.

The second direction looks at abstractions from the perspective of analogies. If two or more objects can be generalised (implied existence of a common abstraction), then the objects are considered analogies and belong to the same class. Classes are defined by the common properties of their members, while the distinct properties are what differentiate the members of the same class from each other. Classes are used to create an inheritance structure of the relationships between actions and objects. These types of abstractions imply a downward-solution property [61]: if a higher level (abstract) solution s' exists, then a specialisation of s' must also exist at the lower levels.

Nilsson (1990) uses the structure of triangle arrays [48] to encapsulate pre-computed plans into triangle tables [48] and tree plans into triangle-table trees. Triangle tables can be seen as a collection of offline macro-actions where the precondition of an action is equivalent to a table kernel [48] and the effect of an action is equivalent to the table effects. If a table kernel is found as the current state during a triangle table scan, then the effects of the table are instantly triggered without executing any other primitive actions. Our method does not require pre-computed plans and is capable of handling domains with a high expressiveness where pre-computed plans are not an option.

Early decomposition strategies relied on human domain engineers who manually captured decompositions explicitly via hand coding. For example, the Hierarchical Task Network (HTN) [25] decomposition strategy uses a top-down approach where an initial planning problem is broken down offline by a domain engineer into compound tasks in order to use pre-constructed plans for solving the non-primitive tasks. However, manual decomposition techniques rely on the ability of the domain engineer to understand how to decompose the problem into components that are at least effectively separable. Our technique eliminates the need for a domain engineer to identify the decompositions or to construct the compound tasks.

As domain-independent planning grew in sophistication, there has been an increasing emphasis on automated approaches for abstracting and decomposing planning problems with domain-independent techniques. Knockblock (1994) presents a fully automated domain-independent procedure for generating problem-specific planning abstractions. Knockblock (1994) also introduces the ordered monotonicity property (OMP) [43] which states that an efficient hierarchical decomposition solves the problem at the lower levels (the unabstracted level being the lowest) without affecting the literals obtained at the higher levels. Our technique respects the OMP as it maintains the structure of the solution obtained at the abstract level (called strategic in our work) also at the unabstracted level (called tactical in our work).

Bacchus & Yang (1994) introduce the downward refinement property (DPR) [4, 5] which states that every abstract solution can be refined to the unabstracted solution without backtracking across other abstract levels if an unabstracted solution exists. Our technique respects the DPR, as once we find a solution π at the abstract level (strategic) we immediately refine π with information obtained from the non-abstract level (tactical) and are never required to backtrack to the abstract level.

Abstractions have also been used to construct powerful heuristics that had a major impact on AI Planning [45, 3]. Bonet et al. use the *ignore delete lists* [8] abstraction (relaxation) as the heuristic in the Heuristic Search Planner (HSP). HSP obtained second place in the competition held during the 1998 International Conference on Artificial Intelligence Planning and Scheduling Systems (AIPS-98). A similar abstraction inspired by HSP was used by Hoffmann (2001) to create the heuristic in the fast-forward planning system (FF) [37], the planner which obtained first place in the AIPS-00 [3] competition.

Botea (2006) extended the FF [37] solver with macro actions to create the Macro FF [11, 10] solver. Macro FF obtained very good results in IPC-4 and made the use of macro-actions gain popularity. Marvin [22] is another solver that uses macro-actions. Marvin employs action-sequence-memoisation techniques to generate the macro actions. Gerevini et al. (2009) use macro actions generated from training problem sets in the portfolio planner PbP [34]. Chrpa (2010) also introduces a procedure that looks for actions that can be assembled into a macro-operator [20]. Hoffman et al. (2017) also introduce a macroplanning database-driven [39] procedure for learning action sequences that frequently appear in plans. These approaches use sample problems to generate macros with the hope that the obtained macros would be applicable to future problems. However, Macro-FF, Marvin and PbP as well as the procedures of Hoffman et al. and Chrpa are dependent on the samples given and operate only in classical planning.

Botea (2006) also describes a top-down abstraction technique for planning that decomposes a Sokoban maze problem into sub-problems based on the rooms and tunnels in the maze. The maze is solved as a high-level global problem that deals only with the transfer of rocks between rooms while each room is considered as a distinct local problem where the local constraints of the abstract actions are mitigated. Our technique uses a bottom-up approach that encapsulates plans of algorithmically generated sub-problems online into macro-actions represented at an abstract level where the global constraints are mitigated.

The Component Abstraction Planner [2] uses an algorithm to identify abstract components and defines them as sub-problems that have their plans integrated as macro-actions in the initial problem. The augmented problem is then passed to a regular solver to find a solution in which the macro-actions are replaced with the corresponding sub-plans. This technique, however, is limited to classical planning.

3.1.2 Multi-agent Planning Decompositions

While the focus of our work is not in the area of planners with a distributed architecture (I.e. multiple planning agents) [60], our technique does identify and exploit the multiple executive agents present in a planning problem for constructing decompositions. Such exploitations have been previously utilised in REALPlan [59], a planner that decomposes an initial problem based on the number of available resources identified within the initial problem and creates an individual sub-problem for each available executive agent identified in the initial problem. REALPlan achieves this by separating the causal reasoning and resource allocations of a problem in order to convert them into constraint satisfaction problems (CSP) [12] solvable with CSP solver. REALPlan, however, is limited to classical planning.

Crosby et al. (2013) present the Agent Decomposition Planner (ADP), a planner which participated in the centralised track of the Competition of Distributed and Multi-Agent Planners (CoDMAP-15) [60]. ADP performs the decomposition of classical planning problems that have a 'multi-agent nature' [24] (I.e. multiple executive agents) by identifying problem variables that behave as executive agents. The technique is based on eliminating all two-way cycles from the causal graph of a given problem and marking the remaining variables that have at least one successor as agent variables. The initial problem is then decomposed based on the identified agents into multiple agent variable decompositions. The decompositions are exploited via a heuristic search algorithm based on the popular 'ignore delete effects' heuristic [37] in order to obtain a total order for achieving the propositions required to satisfy the goal state of the initial problem. ADP obtains competitive results in compatible planning problems, but it is limited to classical planning. Also, the agents identified via ADP do not directly affect each other and can only act on the environment. On the other hand, the agent extraction technique in our thesis identifies agents that directly interact not only with the environment but also with each other and the agent-to-agent interactions are exploited for creating new efficient decompositions.

3.1.3 Goal Decompositions

The decomposition of a planning problem Π based on the decomposition of its goal into multiple sub-goals is another previously explored approach. In such techniques, Π is decomposed into multiple sub-problems, with each sub-problem being responsible for achieving one of the sub-goals obtained from decomposing the goal of Π. The goal of Π is then found by merging the solutions of the sub-problems of Π while making sure all global constraints of Π are satisfied.

An example of such a technique is presented by Yang (2012), who provides a broad and detailed description of decomposed-based planning [67]. The method described by Yang (2012) decomposes a planning problem using goal-directed decompositions. The conflicts between the solutions of individual decompositions are resolved as constraint satisfaction problems. When multiple solutions exist, Yang either performs a cost analysis or uses constraint satisfaction to select the best solution. Yang also describes optimal and sub-optimal plan merger techniques in the form of optimisation problems. In contrast, our technique employs relaxations and abstractions into the decompositions and solves the global constraints at the abstract level.

In more recent work, The Automated Cyclic Planner [1] is designed for problems where multiple instances of the same type of subgoal can be identified in the goal of a problem in order to separate the solving of the goal into multiple cyclic sub-problems. However, this approach only tackles problems where a single repetitive subgoal can be identified.

3.1.3.1 Temporal Goal Decompositions

Manual STP [15] (described in Section 2.1.6 of the previous chapter) decomposes a temporal problem based on the geographical locality of the goals. However, manual STP requires a domain engineer for the decomposition of the top-level goals as well as for the creation of the macro action template and strategic abstraction for each individual planning domain. On the other hand, our thesis extends STP with fully automatic procedures for goal decomposition, macro action creation and strategic abstractions which, as we will show in the Evaluation chapter, yield quality solutions to benchmark planning problems.

Carreno et al. (2020) present the Decentralised Heterogeneous Robot TaskAllocator (DHRTA) algorithm which enhances goal distribution in temporal planning problems by considering task spatial distribution, execution time, and the capabilities of the available robot agents [17]. DHRTA was designed as a response to the poor task allocation strategies existing planners employ. The DHRTA objective function is formed of two cost functions. The first function evaluates the total number of goals each robot agent can complete based on matching the capabilities of each robot to the requirements of each goal. The second function evaluates the distance between points of interest and the makespan of completing each goal. Using DHRTA, an initial problem is effectively split into multiple independent sub-problems, with one sub-problem per robot agent that has goals based on the output of the objective function. However, the capabilities of the robot agents in DHRTA are identified and inputted by a human operator, while in our work, such capabilities are algorithmically identified using a novel analysis procedure of PDDL-encoded data based on landmarks. Also, DHRTA is not applicable to problems where the decompositions violate any of the global constraints of the initial problem (for example problems where robots must share resources to complete the goal). On the other hand, in our work, any violated global constraints are mitigated at the abstract (strategic) level which allows temporal problems with shared resources to also be successfully solved.

3.1.4 Landmarks Decompositions

In classical planning, a landmark [38] is a constraint in the form of a fact that must hold true at least once during the execution of any possible valid plan of a planning problem. Landmarks are discovered by first performing a backchaining operation from the top-level goals in the problem in order to obtain potential landmark candidates. Then, each candidate L is evaluated by removing all actions that make L true in order to assess if a relaxed plan can be found without the removed actions. If a relaxed plan is not found, then L is a landmark. The landmarks of a problem can be used to form a landmarks graph [38] which represents a skeleton of the final solution of a given problem. The landmarks graph has been used in classical planning decompositions by Hoffmann et al. (2004) who developed a decomposition technique that creates a sub-problem for every landmark present in the graph and solves the sub-problems one by one in a longitudinal "landmark to landmark" approach that outputs as the final solution the concatenated sub-plans of each sub-problem.

Another decomposition technique that uses landmarks is STELLA [54]. STELLA decomposes a problem into sub-problems that have as goals clusters of landmarks that are consistent with each other amounting to all the landmarks in the landmarks graph of the problem. Our technique, however, uses a novel approach for exploiting landmarks as it focuses on analysing the similarity of the landmarks as well as an agent-based relaxation of the landmarks between each individual top-level goal in order to group the top-level goals into subgoals based on contextual analyses of all similarities between the top level goals.

3.1.4.1 Temporal Landmarks Decompositions

Landmarks have also been used for decompositions in temporal problems. The TEMPLM [46] planner is able to extract classical landmarks from temporal problems that have deadline constraints and creates a landmarks graph that incorporates the temporal interactions between the extracted landmarks. TEMPLM uses the landmarks graph to identify unsolvable problem instances and to solve problems in which deadline constraints are present. However, TEMPLM is limited only to temporal problems that contain deadline constraints.

Another temporal landmarks extraction technique is described by Karpas et al. (2015) who propose a method capable of extracting temporal landmarks without the need for deadlines to be present in a planning problem. This is achieved by performing a backchaining operation starting from the top-level goals of a problem and deriving new temporal fact and action landmarks along with the temporal constraints among them from each newly identified landmark that is not present in the initial state. Karpas et al. (2015) propose the usage of temporal landmarks for developing new heuristics in order to help existing planners find better solutions. However, our technique takes a distinct approach to the use of landmarks for solving a planning problem by evaluating the similarity of the landmarks extracted from individual top-level goals in order to decompose the problem based on the contextual relationships between the top-level goals.

3.1.5 Recursive Decompositions

Decomposing a classical planning problem using recursions has also been previously explored. Yu Han et al. (2004) decompose the goal of a problem into a set of serializable subgoals solvable in a total order [47]. This strategy considerably reduces the state space search effort of a compatible planning problem. However, the technique comes with no formal definitions or a clear informal guide about compatible problems and is limited to classical planning.

3.1.6 Temporal Causal Graph Decompositions

Using causal graphs in the decompositions of temporal problems has also been previously investigated. The TBurton [65] planner creates decompositions by mapping a planning problem to timed controlled automata in order to obtain a representation of the problem in the form of an acyclic causal graph with each factor of the graph being used to create a sub-problem. The sub-problems are individually solved by a heuristic temporal solver and the resulting sub-plans are combined into a master plan that solves the initial problem. The necessity of TBurton to convert PDDL to timed controlled automata, however, hinders its current ability to handle temporal problems with numeric fluents. On the other hand, our technique uses a full representation in PDDL 2.1 [31] at every step of the solving procedure and is compatible with temporal problems with numeric fluents as a consequence.

Chapter 4

Fundamental Decomposition Elements

Chapter Overview In this chapter, we extract the fundamental elements required for our decomposition technique. The extracted elements are the agents, the agent dependency relationships, the agent classifications, the landmarks and the relaxed version of propositions, events and landmarks.

4.1 Agents Identification and Classification

Section Overview We catalogue as agents all objects that have dynamic types [58] and are active during a goal state search. The dynamic types are extracted from the first argument of all dynamic predicates present in a specific domain via the approach described in Simspon et al. 2000 [58] which, even though it has obvious limitations, they can be overcome by implementing other previous techniques such as the type inference module (TIM). The agents are extracted from the set of dynamic objects by a procedure that determines which dynamic object is a potential part of the solution. We then use a novel method based on reachability analysis [9] to identify the dependency relationships between the identified agents of a planning problem. We further process the obtained relationships to obtain an acyclic dependency graph and classify the agents according to their dependency status and hierarchical position in the obtained graph.

4.1.1 Dynamic And Static Objects Identification

Section Overview In this section, we present and analyse the technique used for identifying the dynamic and static objects of a planning problem.

Definition 4.1. A predicate is *dynamic* if it is part of one or more effects belonging to one or more domain operators. All predicates that are not dynamic are classified as *static*.

Definition 4.2. A type t is *dynamic* if t or any of the supertypes of t are found as the first argument of any dynamic predicate in a given domain. All types that are not dynamic are classified as *static*.

Definition 4.3. An object is *dynamic* if it has a dynamic type or *static* if it has a static type.

The identification of *dynamic* and *static* objects constitutes the building block for the work presented in this thesis. Our classification method follows the description in Simspon et al. 2000 [58]. The procedure (Algorithm 3 which computes in polynomial time) starts with the extraction of *dynamic predicates* by identifying the domain predicates that are present in at least one effect of at least one domain operator (lines 1 to 5). We then mark the type of the first argument of each dynamic predicate as a *dynamic type* (line 6). All objects that have a dynamic type will be classified as *dynamic objects* (lines 11 to 15). All types not identified as dynamic and their respective objects will be marked as *static* (lines 17 to 20). Even though the construction of this procedure seems arbitrary, empirical evidence shows it is successful [58], as it is similar to human intuition in the way predicates are generally formulated by domain engineers - usually the state of the first argument of a predicate is described by the following arguments. The procedure is successful particularly when the PDDL problem/domain configuration is described in English, as in English an action verb is generally used to portray the subject of a sentence rather than the other objects [58].

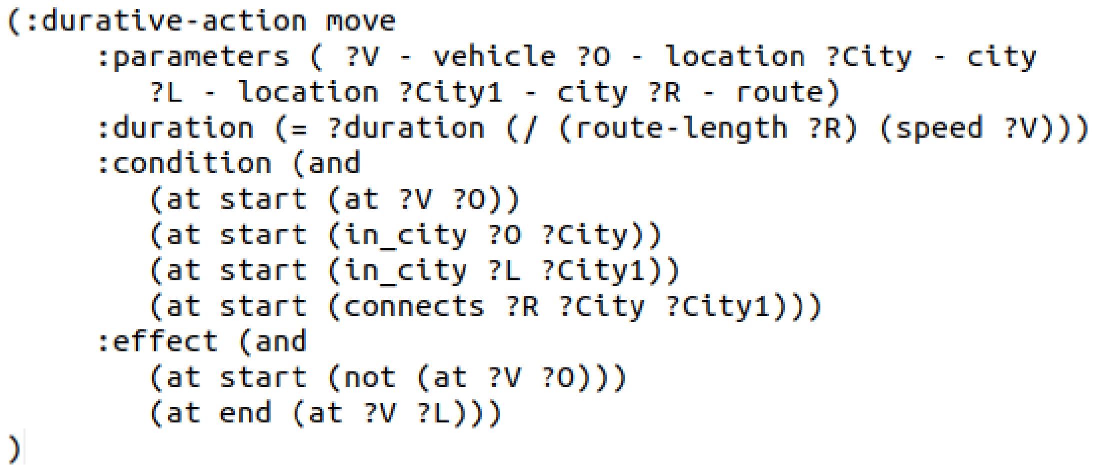
Algorithm 3 Algorithm for extracting the dynamic and static types and objects

```
Input: $\Pi$
Output: The dynamic and static types and objects of $\Pi$
for all predicates $p$ in $\Pi$ do
    for all actions $a \in A$ when $A$ in $\Pi$ do
        for all $eff_a \in a$ do
            if $p \in eff_a$ then
                mark $p$ as a dynamic predicate
                mark type of the first parameter of $p$ as a dynamic type
            end if
        end for
    end for
end for
for all objects $o$ in $\Pi$ do
    for all types $t$ in $\Pi$ do
        if $t$ is a dynamic type then
            if $o$ has type $t$ then
                mark $o$ as a dynamic object
            end if
        else
            mark $t$ as a static type
            if $o$ has type $t$ then
                mark $o$ as a static object
            end if
        end if
    end for
end for
```

Example 4.4. In the RTAM domain, the predicate *(in_city ?L - location ?City - city)* is not found in any of the effects of any domain operator, therefore it will get classified as a *static predicate*. The predicate *(at ?V - vehicle ?L - location)*, however, is part of the effects of the *move* operator (Fig. 4.1), therefore it will be marked as a *dynamic predicate*.

In the predicate *(at ?V - vehicle ?L - location)*, the vehicle location can be changed by the effects *(at start (not (at ?V ?O)))* and *(at end (at ?V ?L)). ?V* is the first argument of the *at* predicate, therefore its type, *vehicle*, and all sub-types will be marked as *dynamic*. The *location* type has not been identified as dynamic, therefore it will get marked as *static*.

Following the action-subject logic of the English language, the predicate *(at ?V - vehicle ?L - location)* describes the location of the vehicle, not the state of the location. If a vehicle were to move from one location to another, we would intuitively perceive that the vehicle has changed its location, not that the state of locations was changed due to the arrival or departure of the vehicle. Therefore, marking the *vehicle* type and all its sub-types as dynamic corresponds with the structure of the domain as well as with the real environment modelled by the domain.



```
(:durative-action move
     :parameters ( ?V - vehicle ?O - location ?City - city
        ?L - location ?City1 - city ?R - route)
     :duration (= ?duration (/ (route-length ?R) (speed ?V)))
     :condition (and
        (at start (at ?V ?O))
        (at start (in_city ?O ?City))
        (at start (in_city ?L ?City1))
        (at start (connects ?R ?City ?City1)))
     :effect (and
        (at start (not (at ?V ?O)))
        (at end (at ?V ?L)))
)
```

Fig. 4.1 RTAM domain *move* action

It is important to note that a bad domain design could interfere with the procedure used for determining dynamic types. Therefore, Algorithm 3 is not sound nor complete. However, there are more robust methods which would identify the correct arguments even in a domain with a bad design for a large number of cases. For example, Simpson et al. (2002) use the type inference module to extract the "mobile" and "location" generic types of a given domain. The same procedure could be used in our work to correctly identify the argument of interest regardless of its position within a predicate. We, however, have not used the mentioned procedure due to the lack of a working implementation for temporal domains and building one ourselves would not have constituted novel work. Also, we have yet to encounter a situation in which a found plan was invalid when using this algorithm as a component in the overall solving procedure (described in Chapter 6). Additionally, our solving technique uses VAL [41] to check for validity, so soundness is computationally cheap and guaranteed.

4.1.2 Agents and Necessary Static Objects Identification

Section Overview In this section, we present and analyse the technique used for a more detailed classification of dynamic objects and static objects.

Definition 4.5. The set $T_d \subseteq T$ is the subset of the set of types T in Π that has all dynamic types which exist in T.

Definition 4.6. do_t is a dynamic object that has type $t \in T_d$.

Definition 4.7. A *dynamic object fact* f_{do_t} is a fact represented by a proposition or function that has dynamic object do_t in Π among one or more of its parameters.

Definition 4.8. A *dynamic object goal* $g_{do_t} \in G$ is a top-level goal represented by a proposition or function that has dynamic object do_t in Π among one or more of its parameters.

Definition 4.9. A *single dynamic object per dynamic type planning problem* $\Pi_{do_t} := \{P,V,A,I,G\}$ is a planning problem focused on a dynamic object do_t in Π which contains dynamic object do_t along with all its initial state dynamic object facts $f_{do_t} \in I$ when I in Π and all its dynamic object goals $g_{do_t} \in G$ when G in Π. Π_{do_t} is identical to a planning problem Π except it does not contain any other dynamic objects, dynamic object facts or goals with the same type as dynamic object do_t. If a dynamic object do'_t in Π_{do_t} has the same type as do_t, then $do'_t = do_t$.

Definition 4.10. Dynamic object do_t is an *active dynamic object* α_t if it is a parameter in at least one condition in a_{pre} or in at least one effect in a_{eff} of at least one action a that is found in the relaxed plan of the single dynamic object per dynamic type planning problem Π_{do_t} focused on dynamic object do_t. The active dynamic objects of a planning problem Π are the *agents* of Π.

Definition 4.11. Dynamic object do_t is an *inactive dynamic object* $\neg\alpha_t$ if it is not a parameter in any condition in a_{pre} or in any effect in a_{eff} of any of the actions in the relaxed plan of the single dynamic object per dynamic type planning problem Π_{do_t} focused on dynamic object do_t or if no relaxed plan is found for the single dynamic object per dynamic type planning problem Π_{do_t} focused on dynamic object do_t.

Definition 4.12. The set $T_s \subseteq T$ is the subset of the set of types T in Π that has all static types which exist in T.

Definition 4.13. so_t is a static object that has type $t \in T_s$.

Definition 4.14. Static object so_t is a *necessary static object* Φ_t if it is a parameter in at least one condition in a_{pre} or in at least one effect in a_{eff} of at least one action a that is found in the relaxed plan of at least one single dynamic object per dynamic type planning problem derived from Π. The necessary static objects of a planning problem Π are the effective environment of Π.

Definition 4.15. Static object so_t is an *unnecessary static object* $\neg\Phi_t$ if it is not a parameter in any of the conditions in a_{pre} or in any effect in a_{eff} of any action a that is found in the relaxed plan of any single dynamic object per dynamic type planning problem derived from Π.

Even though the dynamic-static classification of elements is sufficient to apply the solving technique presented in Chapter 6, the classification can be extended to a lower granularity. We can further differentiate between active and inactive dynamic objects. The active dynamic objects of a problem are the dynamic objects that have a potential role in achieving the goal state of a planning problem Π. We will refer to active dynamic objects as the agents of a problem as they are the potential active participants in finding a solution. We use the term *agents* as it will provide a complementary intuitive understanding of the decomposition procedures described in future sections. The procedure to determine if a dynamic object do_t is an agent (Algorithm 4 which computes in polynomial time) starts by creating a single dynamic object per dynamic type planning problem Π_{do_t} for every dynamic object in a planning problem Π and attempts to find the relaxed plan of each such problem (lines 4 to 7). If a relaxed plan is not found for Π_{do_t} or if do_t is not found in any of the conditions or effects of any of the actions of the relaxed plan of Π_{do_t} we mark dynamic object do_t as an inactive dynamic object $\neg\Phi(do_t)$ (line 5). If do_t, however, is found in at least one condition or in at least one effect in any of the actions of the relaxed plan of Π_{do_t} then we mark agent do_t as an agent of Π (lines 8 to 11).

Similarly to dynamic objects, the static objects of a problem can also be catalogued into necessary and unnecessary static objects. The necessary static objects have a potential role in achieving the goal state of a problem while the unnecessary static objects will never be required for solving a problem. The necessary static objects are identified by their presence in at least one of the conditions or effects of at least one of the actions of the relaxed plan of at least one of the single dynamic object per dynamic type planning problem Π_{do_t} created for each dynamic object in a planning problem Π (lines 13 to 15). If a static object is not found in any of the conditions or effects of any of the actions in the relaxed plan of any of the Π_{do_t} problems, then it is marked as an unnecessary static object (lines 1 and 2).

Algorithm 4 is a heuristic approach designed to potentially reduce the difficulty of a planning problem (detailed in Chapter 5) and to potentially increase the accuracy of the extracted 'advice' (detailed in Sections 4.1.3 and 4.1.4). Algorithm 4 is not complete, as it can be the case that a necessary static object might not be found in the relaxed plan due to the relaxation heuristic. However, our aim is to find quality solutions to large-scale (hard) problems, so guaranteeing completeness is not a practical concern, as using other existing techniques either yields bad solutions or no solutions at all (as showcased in Chapter 7). Also, a domain could be engineered in such a way that the procedure might incorrectly classify a necessary static object as an unnecessary one (due to the relaxation heuristic). However, we have yet to encounter a situation in which a found plan was invalid while experimenting with benchmark planning problems modified according to the criteria in this procedure. Additionally, our solving technique uses VAL [41] to check for validity, so soundness is computationally cheap and guaranteed.

Algorithm 4 Algorithm for Identifying Agents, Inactive Dynamic Objects as well as Necessary static objects and Unnecessary Static Objects

```
Input: $\Pi$ and its dynamic/static object classification
Output: All $\alpha_t$, $\neg\alpha_t$, $\Phi_t$ and $\neg\Phi_t$ in $\Pi$
1:  for all static objects $so_t$ in $\Pi$ do
2:      mark $so_t$ as an unnecessary static object $\neg\Phi_t$
3:  end for
4:  for all dynamic objects $do_t$ in $\Pi$ do
5:      mark $do_t$ as an inactive dynamic object $\neg\alpha_t$
6:      construct single dynamic object per dynamic type planning problem $\Pi_{do_t}$ (as described in Definition 4.9)
7:      attempt to obtain $relaxed\_plan(\Pi_{do_t})$
8:      if $relaxed\_plan(\Pi_{do_t})$ is found then
9:          for all actions $a \in relaxed\_plan(\Pi_{do_t})$ do
10:             if $do_t$ is a parameter in $pre_a$ or in $eff_a$ then
11:                 mark $do_t$ as an active dynamic object $\alpha_t$
12:             end if
13:             for all static objects $so_t$ in $\Pi$ do
14:                 if $so_t$ is a parameter in $pre_a$ or in $eff_a$ then
15:                     mark $so_t$ as an necessary static object $\Phi_t$
16:                 end if
17:             end for
18:         end for
19:     end if
20: end for
```

Example 4.16. In the DLOG-5-5-10 Driverlog planning problem, we have five dynamic objects with the *truck* dynamic type and five dynamic objects with the *driver* dynamic type. In the DLOG-5-5-10 Driverlog planning problem, we also have ten dynamic objects with the *package* dynamic type: *package1, package2, package3, package4, package5, package6, package7, package8, package9, package10* . DLOG-5-5-10 has an *(at package location)* dynamic object goal for each of the *package* dynamic objects except *package1, package3* and the goal for *package7* is achieved in the initial state.

If we apply Algorithm 4 to DLOG-5-5-10, the *package2, package4, package5, package6, package8, package9, package10* dynamic objects will get marked as agents as they will be a parameter in at least a condition of an action in the relaxed plan of their respective single dynamic object per dynamic type planning problem due to the corresponding *(at package location)* goals unachieved in the initial state. However, *package1, package3* will get marked as inactive dynamic objects as they have no corresponding *(at package location)* goal so they will not be a parameter in any of the actions of the relaxed plan of their respective single dynamic object per dynamic type planning problem or in any other action of any other relaxed plan of any other single dynamic object per dynamic type planning problem. *package7* will also get marked as an inactive dynamic object even though it has a corresponding goal in the goal state. This happens because the goal is already achieved in the initial state so *package7* will not be a parameter in any of the actions of the relaxed plan of its single dynamic object per dynamic type planning problem or of any other single dynamic object per dynamic type planning problem.

Applying Algorithm 4 to DLOG-5-5-10 will also mark each truck and driver dynamic objects as agents as each of the trucks and drivers can be potentially selected by a planner to achieve a package goal so each truck and driver will be a parameter in at least a condition of an action in the relaxed plan of their respective single dynamic object per dynamic type planning problem.

Some of the necessary static objects in DLOG-5-5-10 are the *location* static objects that are parameters in the *at driver location* and *at truck location* initial state facts, as each *location* will be a parameter in at least one condition in the relaxed plan of the corresponding single dynamic object per dynamic type planning problem Π_{do_t} created for the *driver* or *truck* dynamic object that is a parameter in the same *at* initial state fact as the *location*. If, however, we modify DLOG-5-5-10 and add a new location l and connect it to the existing locations map as a dead-end, l would be an unnecessary static object as it would not be a parameter in any of the conditions in a_{pre} of any effects in a_{eff} of any action a that is found in the relaxed plan of any single dynamic object per dynamic type planning problem derived from DLOG-5-5-10 (a more detailed example is provided in Section 5.5 of Chapter 5).

4.1.3 Cleaning Planning Problem

Section Overview In this section, we present a technique for eliminating potentially redundant elements from a planning problem.

Definition 4.17. A *cleaned planning problem* is a planning problem Π that has no facts in I that have inactive dynamic objects or unnecessary static objects among any of their parameters, no goals in G that have inactive dynamic objects or unnecessary static objects among any of their parameters and no inactive dynamic objects or unnecessary static objects.

Definition 4.18. An *static object fact* f_{so_t} is a fact represented by a proposition or function that has static object so_t in Π among one or more of its parameters.

Definition 4.19. An *static object goal* $g_{so_t} \in G$ is a top-level goal represented by a proposition or function that has static object so_t in Π among one or more of its parameters.

Our thesis uses the instantiated agents and environment in order to extract accurate decomposition 'advice' (procedure described in Section 4.1.4). The presence of inactive dynamic objects or unnecessary static object facts might interfere with the accuracy of the extracted 'advice'. Therefore, we "clean" the original planning problem Π before proceeding with extracting the elements required for obtaining the 'advice'. The procedure for cleaning a planning problem (Algorithm 5 which has a supportive role and computes in polynomial time) identifies and removes all inactive dynamic objects $\neg\Phi(do_t)$, all inactive dynamic object facts $f_{\neg\Phi(do_t)}$ and all inactive dynamic object goals $g_{\neg\Phi(do_t)}$ (lines 2 to 5) as well as identifies and removes all unnecessary static objects $\neg\Phi(so_t)$, all unnecessary static object facts $f_{\neg\Phi(so_t)}$ and all unnecessary static goals $g_{\neg\Phi(o_t)}$ (lines 7 to 10) from a planning problem Π. Since the cleaning procedure is reliant on the output of Algorithm 4, it is also not complete and not sound but practically useful for large-scale (hard) problems. Also, the reliance on Algorithm 4 means that a domain can be engineered in such a way that the cleaning procedure might remove incorrectly classified objects. However, we have yet to encounter a situation in which a found plan was invalid while experimenting with the removal of the elements this algorithm would identify from the sampled benchmark planning problems. Additionally, VAL is used [41] to check for validity, so soundness is computationally cheap and guaranteed.

Algorithm 5 Algorithm for Cleaning a Planning Problem Π

```
Input: $\Pi$, $\neg\alpha_t$ in $\Pi$ and $\neg\Phi_t$ in $\Pi$
Output: the cleaned version of $\Pi$
 1: for all inactive dynamic objects $\neg\alpha_t$ in $\Pi$ do
 2:     remove inactive dynamic object facts $f_{\neg\alpha_t}$ from $\Pi$
 3:     remove inactive dynamic object goals $g_{\neg\alpha_t}$ from $\Pi$
 4:     remove $\neg\alpha_t$ from $\Pi$
 5: end for
 6: for all unnecessary static objects $\neg\Phi_t$ in $\Pi$ do
 7:     remove unnecessary static object facts $f_{\neg\Phi_t}$ from $\Pi$
 8:     remove unnecessary static object goals $g_{\neg\Phi_t}$ from $\Pi$
 9:     remove $\neg\Phi_t$ from $\Pi$
10: end for
11: return cleaned version of $\Pi$
```

Example 4.20. In the DLOG-5-5-10 Driverlog planning problem, we have ten dynamic objects with the *package* dynamic type: *package1, package2, package3, package4, package5, package6, package7, package8, package9, package10* . DLOG-5-5-10 has an *(at package location)* dynamic object goal for each of the *package* dynamic objects except *package1, package3* and the goal for *package7* is achieved in the initial state.

The cleaned version of DLOG-5-5-10 Driverlog planning problem no longer has the *package1, package3, package7* dynamic objects, dynamic object facts and dynamic object goals, as *package1, package3, package7* are inactive dynamic objects (as described in Example 4.16).

4.1.4 Agents Dependency Relationships

Section Overview In this section, we present and analyse the method for extracting the dependency relationships between agents (preferred) or between dynamic objects.

Definition 4.21. An *agent type* t_α is a dynamic type $t \in T_d$ that has at least one agent instance α_t with type t in a planning problem Π.

Definition 4.22. An *agent fact* f_{α_t} is a fact represented by a proposition or function that has agent $\alpha_t \in \Pi$ among one or more of its parameters.

Definition 4.23. An *agent goal* $g_{\alpha_t} \in G$ is a top-level goal represented by a proposition or function that has agent $\alpha_t \in \Pi$ among one or more of its parameters.

Definition 4.24. The *reachability impact* of t onto $t\prime$ when $t, t' \in T_d$ and $t \neq t'$ represents the effect of removing all f_{α_t} facts from the initial state $I \in \Pi$ in achieving all $f_{\alpha_{t\prime}}$ facts in all fact layers when conducting the reachability analysis [9] of planning problem Π.

Definition 4.25. Set $F(\Pi)$ represents the union of all facts from all fact layers obtained when performing the reachability analysis of planning problem Π.

Definition 4.26. Set $F_{t_\alpha}(\Pi) \subseteq F(\Pi)$ is the subset of $F(\Pi)$ that has all f_{α_t} facts present in $F(\Pi)$.

Definition 4.27. Set $I_{\neg t_\alpha} \subseteq I$ is the subset of I that contains all facts $f \in I$ that are not f_{α_t} facts.

Definition 4.28. Set $G_{\neg t_\alpha} \subseteq G$ is the subset of G that contains all goals $g \in G$ which are not represented by propositions or functions that have any α_t.

Definition 4.29. Planning problem $\Pi\neg_{t_\alpha}$ is defined as the tuple $\{P, V, A, I_{\neg t_\alpha}, G_{\neg t_\alpha}\}$.

Definition 4.30. Set $\neg F_{t'_\alpha}(\Pi\neg_{t_\alpha}) \subseteq F(\Pi)$ is the set of all facts $f \in F_{t'_\alpha}(\Pi)$ when $f \notin F_{t'_\alpha}(\Pi\neg_{t_\alpha})$ and $t_\alpha \neq t'_\alpha$ (Algorithm 6). $\neg F_{t'_\alpha}(\Pi\neg_{t_\alpha})$ represents the set of facts affected by the reachability impact of t_α onto t'_α in the reachability analysis of planning problem Π.

Definition 4.31. We mark $t_\alpha \longmapsto t'_\alpha$ as a dependency relationship if the size of the facts affected by the reachability impact of t_α onto t'_α (i.e. size of $\neg F_{t\prime_\alpha}(\Pi\neg_{t_\alpha})$) is greater then zero when $t_\alpha \neq t'_\alpha$ and $t'_\alpha, t_\alpha \in T_d$.

A $t_\alpha \longmapsto t'_\alpha$ dependency relationship signifies that agents with type t'_α are dependent and hierarchically inferior to agents with type t_α.

Definition 4.32. $\Delta(\Pi)$ is the set of all dependency relationships of planning problem Π.

The agents dependency relationships are a core component of the work presented in this thesis. They are required for classifying the agents of a planning problem Π in order to use the classifications as decomposition 'advice' (described in Chapter 6). Our method for determining the agent dependencies of a planning problem (Algorithm 6 which computes in polynomial time) is based on the relationship between the agents present in a problem, so it disregards each dynamic type $t \in T_d$ that does not have any agent instances regardless if t is the supertype of an existing agent. The method consists of removing all agents with a particular type from the planning problem in order to observe the reachability impact of the removal upon the remaining agents when conducting a reachability analysis [9]. As a first step, we compute the reachability analysis facts, preferably for a planning problem that has been cleaned (line 1). Then, we iteratively compute the reachability analysis facts for each planning problem $\Pi\neg_{t\alpha}$ (lines 1 and 2). Every fact that is found in $F_{t'_\alpha}(\Pi)$ and is not found in $F_{t'_\alpha}(\Pi\neg_{t_\alpha})$, when $t \neq t'$, is added to its respective reachability impact set $\neg F_{t'_\alpha}(\Pi\neg_{t_\alpha})$ (lines 4 to 7). If $\neg F_{t'}(\Pi\neg_t) > 0$, then the removal of f_{α_t} facts from the initial state of Π has impacted the reachability of $f_{\alpha_{t'}}$ facts in the reachability analysis of planning problem Π, therefore we mark $t \longmapsto t'$ as a dependency relationship and add it to the dependency relationship set $\Delta(\Pi)$ (lines 10 to 13).

Algorithm 6 Algorithm for extracting the dependency relationships

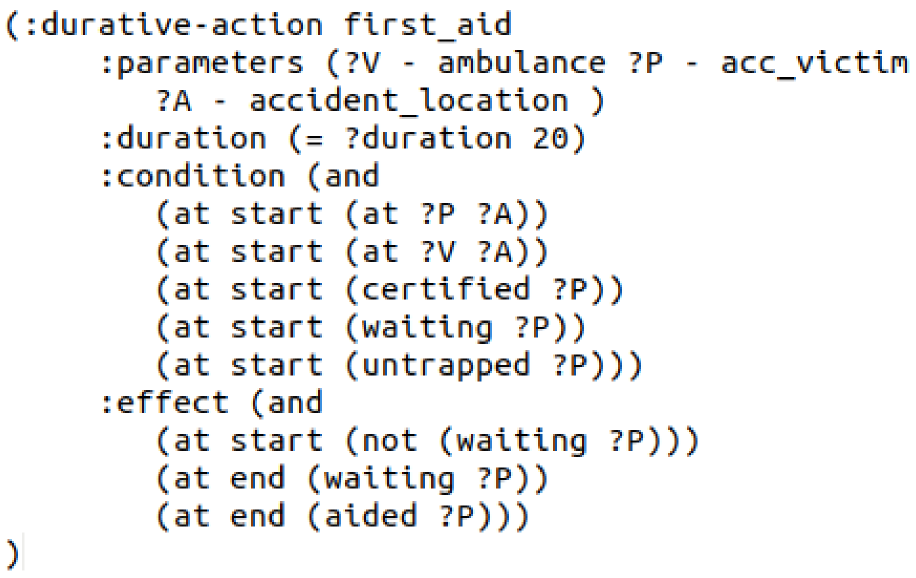
```
Input: Π as well as all object and type classifications in Π
Output: $\Delta(t_\alpha)$ in Π
1:  compute $F(\Pi)$
2:  for all agent types $t_\alpha \in T_d$ do
3:      compute $F(\Pi\neg_{t_\alpha})$
4:      for all agent types $t'_\alpha \in T_d$ when $t'_\alpha \neq t_\alpha$ do
5:          for all facts $f \in F_{t'_\alpha}(\Pi)$ do
6:              if $f \notin F_{t'_\alpha}(\Pi\neg_{t_\alpha})$ then
7:                  add fact $f$ to $\neg F_{t'_\alpha}(\Pi\neg_{t_\alpha})$
8:              end if
9:          end for
10:         if size of $\neg F_{t'_\alpha}(\Pi\neg_{t_\alpha}) > 0$ then
11:             mark $t_\alpha \longmapsto t'_\alpha$ as a dependency relationship;
12:             add $t_\alpha \longmapsto t'_\alpha$ to $\Delta(\Pi)$
13:         end if
14:     end for
15: end for
16: return $\Delta(t_\alpha)$
```

It is important to note that the dynamic types that are not agent types but are supertypes of existing agents, even though they are not part of the agent dependencies identification procedure, do have an impact on the final solution and will be addressed in Chapter 5 along with the creation of the sub-problems. Also, identifying inactive dynamic objects and eliminating them (cleaning the problem) can potentially provide more accurate dependency relationships, particularly for problems that have inactive dynamic objects which cause distinct dependency relationships to the ones caused by the active dynamic objects (agents). Algorithm 6 is not complete, but rather a heuristic approach to enable an efficient decomposition of large-scale (hard) problems where completeness is not a practical concern (due to the poor or nonexistent solutions obtained if using other existing techniques). The correctness of the dependency relationships identification procedure relies on the correctness of the agent identification procedure, so it might be possible to obtain incorrect relationships. However, we have yet to encounter a situation in which a found plan was invalid when using this algorithm as a component in the overall solving procedure (described in Chapter 6). Additionally, VAL is used [41] to check for validity, so soundness is computationally cheap and guaranteed.

Example 4.33. The RTAM_5_1_35 planning problem does not contain any inactive dynamic objects, so the cleaned version of RTAM_5_1_35 is the same as the original RTAM_5_1_35 problem. In RTAM_5_1_35, set T_d contains the following dynamic types: { *acc_victim, car, ambulance, fire_brigade, police_car, tow_truck, subject, vehicle* }. RTAM_5_1_35 does not have any agent instances with the *subject* and *vehicle* dynamic types so *subject* and *vehicle* do not constitute agent types. RTAM_5_1_35 does have at least one agent with the *acc_victim, car, ambulance, fire_brigade, police_car, tow_truck* } dynamic types so they constitute agent types. When applying Algorithm 6 to planning problem *RTAM_5_1_35*, we iterate through all facts with agent types in T_d to determine the reachability impact of t_α onto t'_α when $t_\alpha, t'_\alpha \in T_d$ and $t_\alpha \neq t'_\alpha$. When we are at the iteration where $t_\alpha = ambulance$, $t'_\alpha = accident\_victim$ and $\Pi \neg_{t_\alpha}$ = $RTAM\_5\_1\_35_{\neg ambulance}$, fact *(aided acc_victim2)* is found in $F_{acc\_victim}(RTAM\_5\_1\_35)$. However, *(aided acc_victim2)* is not found in $F_{acc\_victim}(RTAM\_5\_1\_35_{\neg ambulance})$.

The reason fact *(aided acc_victim2)* is not found in the above example is that the state space of the reachability analysis of planning problem $RTAM\_5\_1\_35_{\neg ambulance}$ is never expanded via the operator *first_aid* (Figure 4.2), and *first_aid* is the sole operator that has the *(at end (aided ?P))* effect in the *RTAM* domain. The state space expansion from *first_aid* is not possible as the *(at start (at ?V ?A))* condition of the operator requires a fact that can not exist in the fact layers of the reachability analysis of a planning problem that has no $\alpha_{ambulance}$ agents.

Therefore we add fact *(aided acc_victim2)* to its respective reachability impact set $\neg F_{accident\_victim}(\Pi_{\neg ambulance})$, and consequently mark $ambulance \longmapsto accident\_victim$ as a dependency relationship and add it to the dependency set $\Delta(RTAM\_5\_1\_35)$.



```
(:durative-action first_aid
     :parameters (?V - ambulance ?P - acc_victim
        ?A - accident_location )
     :duration (= ?duration 20)
     :condition (and
        (at start (at ?P ?A))
        (at start (at ?V ?A))
        (at start (certified ?P))
        (at start (waiting ?P))
        (at start (untrapped ?P)))
     :effect (and
        (at start (not (waiting ?P)))
        (at end (waiting ?P))
        (at end (aided ?P)))
)
```

Fig. 4.2 RTAM domain *first_aid* action

4.1.5 Conflicting Dependency Relationships and Their Mitigation

Section Overview In this section, we present and analyse the mitigation procedure in case of conflicting dependency relationships.

Definition 4.34. The directed graph $DG(\Pi) = (N, E)$ represents the *dependency graph* of a planning problem Π. The nodes in N are a mapping of the agent types in T_d of a problem Π. Each edge in set E is a directed path from the node mapping the independent type to the node mapping the dependent type within each dependency relationship in $\Delta(\Pi)$. A dependency cycle is a cycle in the dependency graph.

Definition 4.35. The dependency relationships equivalent to the edges that form one or more cycles in graph $DG(\Pi)$ are called *conflicting* dependency relationships. If $DG(\Pi)$ is acyclic, then planning problem Π has no conflicting dependency relationships.

Definition 4.36. The *weakest link* of a dependency cycle is the dependency relationship $t_\alpha \longmapsto t'_\alpha$ with no propositions from its respective reachability impact set $\neg F_{t'_\alpha}(\Pi_{\neg t_\alpha})$ present in the goal state G in Π if and only if it is the single such dependency in an identified dependency cycle.

There is the possibility of extracting conflicting dependency relationships when using the procedure described in Algorithm 6. In such a case, both $t_\alpha \longmapsto t'_\alpha$ and $t'_\alpha \longmapsto t_\alpha$ could be identified as dependency relationships when $t_\alpha \neq t'_\alpha$. However, determining a hierarchy between agents is not possible if conflicting dependency relationships exist.

To mitigate the conflicts, we exploit the fact that conflicting dependency relationships form cycles when mapped into a directed graph (similarly to how variable dependencies can cause cycles in the causal graph[7] of a planning problem) and that the removal of a single directed edge from a cycle is sufficient to break the cycle and therefore eliminate the conflicts between the remaining relationship. Our method for determining what $t_\alpha \longmapsto t'_\alpha$ dependency relationship should be removed relies on comparing the propositions in the reachability impact set to the top-level goals from G in Π. We mark a dependency as the weakest link and remove it from $\Delta(\Pi)$ if and only if it is the single dependency within a cycle that has a reachability set that does not contain any facts in common with G.

Algorithm 7 Algorithm for mitigating conflicting dependency relationships

```
Input: $\Delta(t_\alpha)$ in $\Pi$, $T_d$ in $\Pi$ and G in $\Pi$
Output: Acyclic $DG(\Pi)$ or failure
1: if $\exists\ DG(\Pi)$ then
2:     clear $DG(\Pi)$
3: end if
4: generate $DG(\Pi)$ from all $t_\alpha \in T_d$ in $\Pi$ and $\Delta(\Pi)$
5: if $DG(\Pi) \neq acyclic$ then
6:     C = first identified cycle via depth-first search (DFS)
7:     depedencies_with_no_facts_in_G = 0
8:     for all $t_\alpha \longmapsto t'_\alpha \in$ C do
9:         if $\neg F_{t'_\alpha}(\Pi_{\neg t_\alpha}) \cap G == 0$ then
10:            weakest-link = $t_\alpha \longmapsto t'_\alpha$
11:            depedencies_with_no_facts_in_G++
12:        end if
13:    end for
14:    if depedencies_with_no_facts_in_G == 1 then
15:        remove weakest-link from $\Delta(\Pi)$
16:    else
17:        return procedure failed
18:    end if
19: end if
20: repeat
21:     Algorithm 7
22: until $DG(\Pi)$ is acyclic
```

The procedure for mitigating conflicting dependency relationships is presented in Algorithm 7, an algorithm which acts as an extension to the procedure for identifying the agent dependency relationships (described in Algorithm 6). We first generate a new graph $DG(\Pi)$ from the agent types in T_d of Π and from $\Delta(\Pi)$ (lines 1 to 4). Then, we attempt to find a cycle within $DG(\Pi)$ via a depth-first search (lines 5 and 6). If a cycle is detected, we identify its weakest link by determining if there is a single cycle dependency relationship that has no propositions from its reachability impact set present in G (lines 7 to 11).

If such a dependency is found we remove it from the dependency relationships set $\Delta(\Pi)$ (lines 14 and 15). The mitigation algorithm fails and stops the whole procedure if more than one such dependency is found in the same cycle (lines 16 and 17). We repeat Algorithm 7 until no more cycles are found in $DG(\Pi)$ (lines 21 and 22).

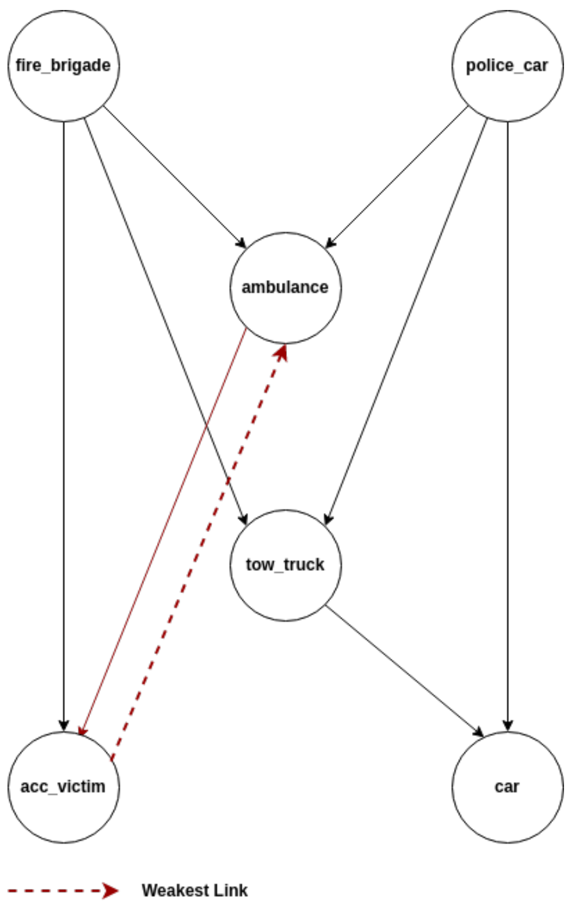


Fig. 4.3 Unmitigated dependency relationships graph DG(RTAM_5_1_35)

Example 4.37. Our method for mitigating conflicting dependency relationships leverages the structure formed by the top-level goals of each specific planning problem. For example, when we apply Algorithm 7 to the Δ*(RTAM_5_1_35)* dependency relationships set, we obtain the following cycle of conflicting dependency relationships in graph *DG(RTAM_5_1_35)* (Figure 4.3):

$$accident\_victim \longmapsto ambulance$$
$$ambulance \longmapsto accident\_victim$$

Reachability impact set $\neg F_{ambulance}(\Pi^{\neg}_{accident\_victim})$ has a single fact:

- *(busy ambulance0)*

Reachability impact set $\neg F_{accident\_victim}(\Pi^{\neg}_{ambulance})$ has a total of 175 facts with the following format:

- *(aided ?accident_victim)* - 35 facts
- *(delivered ?accident_victim)* - 35 facts

- *(at ?accident_victim ?location)* - 105 facts

None of the facts from $\neg F_{ambulance}(\Pi\neg_{accident\_victim})$ are found in G, but the *(delivered ?accident_victim)* facts from $\neg F_{accident\_victim}(\Pi\neg_{ambulance})$ are found in G. Therefore *accident_victim* $\longmapsto$ *ambulance*, the respective dependency relationship of reachability impact set $\neg F_{ambulance}(\Pi\neg_{accident\_victim})$, is identified as the weakest link (Figure 4.3) and removed from the dependency relationships set $\Delta(RTAM\_5\_1\_35)$.

The conflicting dependency relationships mitigation procedure, however, is not guaranteed to remove the minimum number of edges from dependency graph $DG(\Pi)$. The procedure is a polynomial problem intended to eliminate all dependency cycles in a timely manner by removing a sufficient number of edges rather than optimised for keeping the maximum number of dependencies. A more robust procedure that removes the minimum number of edges, however, would be NP-Hard, as such a procedure constitutes a search problem that must parse all the subsets of the graph which would increase the run time exponentially in the worst-case scenario.

4.1.6 Agent Classification

Section Overview In this section, we use the dependency relationships in the acyclic graph $DG(\Pi)$ obtained after eliminating the conflicting dependencies, as well as the agent occurrence in the actions and the relaxation of a planning problem, to classify the agents in Π. The classifications will be used in analysing the difficulty of planning problems (described in Chapter 5) and in the decomposition procedures (Chapter 6).

4.1.6.1 Agent Dependency Status

Section Overview In this section, we present an agent classification based on the dependency status in the agent dependency relationships.

Definition 4.38. A *dead-end type* is an agent type $t_\alpha \in T_d$ mapped to a node that does not have any outgoing edges in the acyclic dependency relationships graph $DG(\Pi)$ and has at least one incoming edge in the acyclic dependency relationships graph $DG(\Pi)$.

Definition 4.39. A *parent type* is an agent type $t_\alpha \in T_d$ mapped to a node that at least one outgoing edge in the acyclic dependency relationships graph $DG(\Pi)$.

Definition 4.40. A *dead-end agent* $d\alpha_t$ is an agent that has a dead-end type $t \in T_d$ when T_d in Π.

Definition 4.41. A *parent agent* $p\alpha_t$ is an agent that has a parent type $t \in T_d$ when T_d in Π.

Definition 4.42. A *dead-end agent fact* $f_{d\alpha_t}$ is an agent fact represented by a proposition or function that has dead-end agent $d\alpha_t$ among one or more of its parameters and no parent agents among its parameters.

Definition 4.43. A *parent agent fact* $f_{p\alpha_t}$ is a fact represented by a proposition or function that has parent agent $p\alpha_t$ among one or more of its parameters.

Definition 4.44. A *dead-end agent goal* $g_{d\alpha_t}$ is an agent goal represented by a proposition or function that has dead-end agent $d\alpha_t$ among one or more of its parameters and no parent agents among its parameters.

Definition 4.45. A *parent agent goal* $g_{p\alpha_t}$ is an agent goal represented by a proposition or function that has parent agent $p\alpha_t$ among one or more of its parameters and no dead-end agents among its parameters.

The classification used for constructing the decompositions in our thesis (Chapter 6) is based on the agent dependency status (Algorithm 8 which has a supportive role and computes in polynomial time). The classification is used to decompose the problem based on a two-level hierarchy (detailed in Chapter 6 Sections 6.1, 6.3 and 6.4). We separate the agents that have no influence on other agents (lines 3 and 4) from the agents that do (lines 6 and 7). We refer to the former as *dead-end agents* and to the latter as *parent agents*. We then classify all objects, facts and goals according to their agent dependency status (line 10).

Algorithm 8 Algorithm for extracting Agent Dependency Status from the Dependency Relationships

```
Input: $DG(\Pi)$
Output: All $d\alpha_t$, $p\alpha_t$, $f_{d\alpha_t}$, $f_{p\alpha_t}$, $g_{d\alpha_t}$, $g_{p\alpha_t}$ and all dead-end/parent types in $\Pi$
 1: for all nodes N $\in DG(\Pi)$ do
 2:     $t_\alpha$ = mapped to N
 3:     if N outgoing edges == 0 and N incoming edges >= 1 then
 4:         mark $t_\alpha$ as a dead-end type
 5:     end if
 6:     if N outgoing edges >= 1 then
 7:         mark $t_\alpha$ as a parent type
 8:     end if
 9: end for
10: mark all agents, agent facts and agent goals as described in Definitions 4.40 to 4.45
```

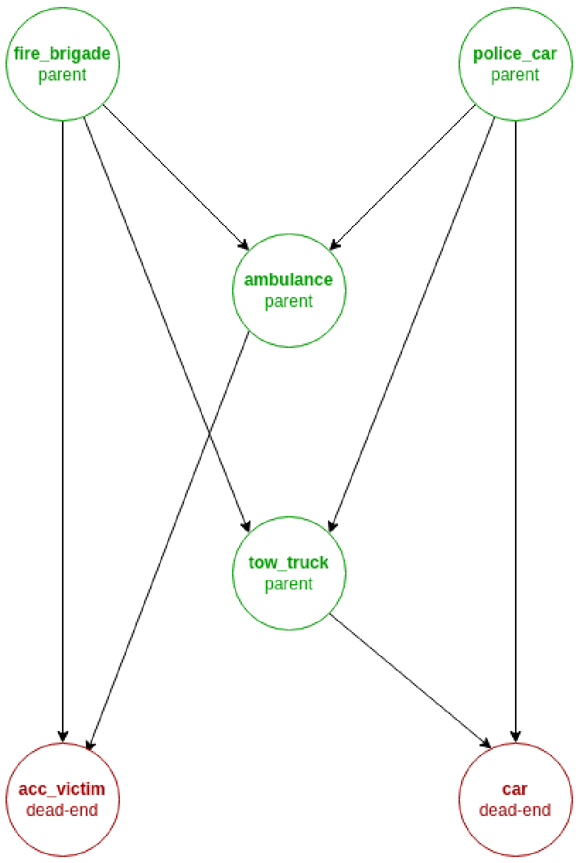


Fig. 4.4 Agent Dependency Status in acyclic graph DG(RTAM_5_1_35)

Example 4.46. In acyclic graph *DG(RTAM_5_1_35)*, the nodes mapped to the *acc_victim* and *car* agent types each have no outgoing edges and two incoming edges. Therefore, applying Algorithm 8 to the acyclic graph *DG(RTAM_5_1_35)* will mark the *acc_victim* and *car* agent types as dead-end types. All *acc_victim* and *car* agent facts and agent goals will be marked as dead-end agent facts and dead-end agent goals.

However, the nodes mapped to the *ambulance*, *police_car*, *fire_brigade* and *tow_truck* agent types each have outgoing edges. Therefore, applying Algorithm 8 to the acyclic graph *DG(RTAM_5_1_35)* will mark the *ambulance*, *police_car*, *fire_brigade* and *tow_truck* agent types as parent types. All *ambulance*, *police_car*, *fire_brigade* and *tow_truck* agent facts and agent goals will be marked as parent agent facts and parent agent goals.

4.1.6.2 Agent Hierarchical Priorities

Section Overview In this section, we present an agent classification based on the hierarchical position of agents in the agent dependency relationships.

Definition 4.47. *priority(t)* represents the hierarchical priority of an agent type $t_\alpha \in T_d$ when T_d in Π, with t_α mapped to its corresponding node from $DG(\Pi)$.

Another possible classification can be made using the distance of the agent types from each other within $DG(\Pi)$ in order to potentially decompose the problem based on a multi-level hierarchy. While the decomposition and solving procedure described and used in Chapter 6 uses the agent dependency status classification (presented in Section 4.1.6.1), a multi-level hierarchy classification could also be used for more detailed decompositions. The hierarchical priorities extraction procedure (Algorithm 9 which computes in polynomial time) starts by first assigning *priority(t)* = 0 to the PDDL types which correspond to the nodes in $DG(\Pi)$ that do not have any incoming edges (lines 1 to 4). The identified nodes will constitute roots for depth-first search traversals, with one traversal for each root (line 5). During a traversal, each type corresponding to a newly discovered node gets assigned a priority equal to its distance from the root of its respective subgraph (line 16). If a node is reachable via multiple paths with distinct distances from the root in the same subgraph, the node corresponding type gets assigned the priority of the largest distance (lines 13 and 14). A type mapped to a node present in multiple subgraphs with distinct root distances will keep the priority of the largest distance (lines 8 to 10). The dependencies hierarchy grows proportionally to the max distance between the root and the discovered nodes.

Algorithm 9 Algorithm for extracting Agent Hierarchical Priorities from Dependency Relationships

```
Input: $DG(\Pi)$
Output: priority($t_\alpha$) for all $t_\alpha$ in $\Pi$
 1: for all nodes N $\in DG(\Pi)$ do
 2:     agent type $t_\alpha$ = agent type mapped to N
 3:     if N incoming edges == 0 then
 4:         priority($t_\alpha$) = 0
 5:         do DFS from root N
 6:         for all nodes N/ discovered in DFS(N) do
 7:             agent type $t'_\alpha$ = agent type mapped to N/
 8:             if priority($t'_\alpha$) != Null then
 9:                 if (distance between N/ and N) > priority($t'_\alpha$) then
10:                     priority($t'_\alpha$) = distance between N/ and N
11:                 end if
12:             else
13:                 if multiple paths from N to N/ then
14:                     priority($t'_\alpha$) = distance of the longest path
15:                 else
16:                     priority($t'_\alpha$) = distance between N/ and N
17:                 end if
18:             end if
19:         end for
20:     end if
21: end for
```

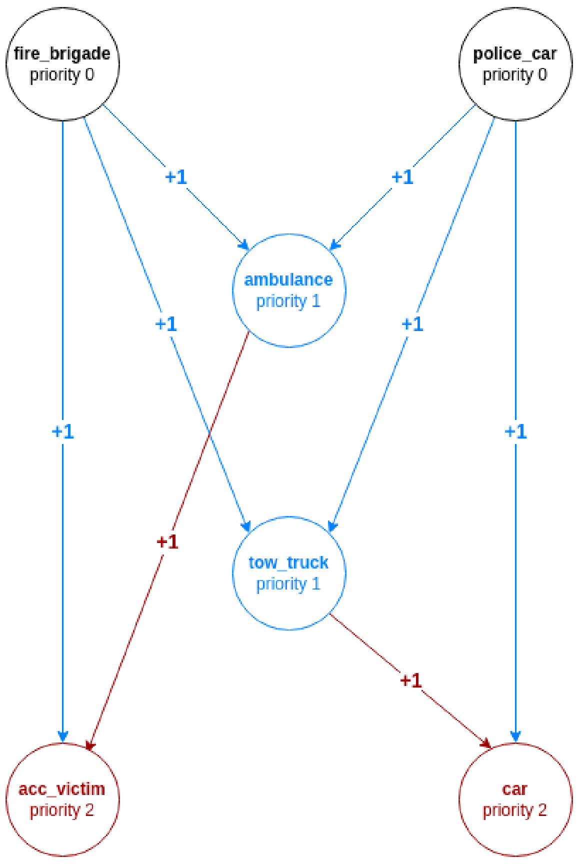


Fig. 4.5 Agent priorities in acyclic graph DG(RTAM_5_1_35)

Example 4.48. Applying Algorithm 9 to the acyclic graph *DG(RTAM_5_1_35)*, we first obtain priority(*fire_brigade*) = 0 and priority(*police_car*) = 0, as the two nodes have no incoming edges (Figure 4.5). By conducting a depth-first search traversal of *DG(RTAM_5_1_35)* starting from root *fire_brigade*, we obtain a subgraph with the following paths:

fire_brigade - *acc_victim*
fire_brigade – *tow_truck* – *car*
fire_brigade – *ambulance* – *acc_victim*

The nodes *tow_truck*, *car* and *ambulance* can be reached via a single path starting from the root, so they get assigned as priorities their respective distance from the root: priority(*tow_truck*) = 1, priority(*car*) = 2 and priority(*ambulance*) = 1. However, *acc_victim* can be reached via two paths starting from the root, therefore it gets assigned priority(*acc_victim*) = 2, the distance of the longest path (*fire_brigade* – *ambulance* – *acc_victim*).

By conducting a depth-first search traversal of *DG(RTAM_5_1_35)* starting from root *police_car*, we obain a subgraph with the following paths:

police_car - *ambulance* - *acc_victim*
police_car – tow_truck – car
police_car – car

Nodes *ambulance*, *acc_victim*, *tow_truck* and *car* were discovered in both subgraphs, therefore we have to assign as priorities the max distance from the root among both subgraphs. However, in this example, all nodes have the same root distances in both subgraphs, so no changes to the priorities are required.

4.2 Relaxation of Propositions and Events

Section Overview In this section, we describe two relaxations of planning problem elements which will be used as decomposition heuristics. The relaxed format allows the extraction of new relevant data from the planning problem which, as we will show throughout our work, can be used for constructing decomposition heuristics. We will show how the relaxed formats reflect how humans use environment similarities as a means to catalogue incomplete information in order to create efficient reasoning models.

4.2.1 Relaxed Propositions and Relaxed Agent Propositions

Definition 4.49. A *relaxed proposition* is a proposition that has at least one of its parameters in a lifted format.

Definition 4.50. A *relaxed agent proposition* is a relaxed proposition that has at least one agent type among its lifted format parameters.

Relaxed propositions are propositions with at least one parameter relaxed to the form of a type. In our work, we relax only propositions that contain one or more agents and we call such propositions *relaxed agent propositions*. Each relaxed agent proposition will have all parameters that are agents replaced with the corresponding agent type. All other parameters will be kept in their original format. The final result is a boolean construct that can encapsulate both objects and types, similarly to a predicate with constants.

Example 4.51. In the RTAM_5_1_35 planning problem, the proposition *(at acc_victim0 accident_location2)* contains the *acc_victim0* agent. To obtain the relaxed agent proposition, we replace *acc_victim0* with its type, *acc_victim*. The final construct is *(at acc_victim accident_location2)*. This construct allows us to encode a state in which we know the location of an accident victim without knowing the exact victim: a state which can exist in real-world environments.

4.2.2 Relaxed Events and Relaxed Agent Events

Definition 4.52. A *relaxed event* is an event that has at least one of its parameters in a lifted format.

Definition 4.53. A *relaxed agent event* is a relaxed event that has at least one agent type among its lifted format parameters.

Similarly to relaxed propositions, *relaxed events* are events with at least one parameter relaxed to the form of a type. In our work, we relax only events that contain or more agents and we call such events *relaxed agent events*. Each relaxed agent event will have all parameters that are agents replaced with their corresponding agent type. All other parameters will be kept in their original format. The final result is a construct capable of encapsulating both objects and types, similar to a partially grounded action [51]. Relaxed events, however, are not limited to objects only belonging to connected maps, as opposed to partially grounded actions where only such objects can be represented.

Example 4.54. In the RTAM_5_1_35 planning problem, a possible event could be *START(confirm_accident police_car0 acc_victim0 accident_location2)*. To obtain the relaxed agent event, we replace the *acc_victim0* agent with its type, *acc_victim*; we also replace the *police_car0* agent with its type, *police_car*. The final construct is *START(confirm_accident police_car acc_victim accident_location2)*. This construct allows us to encode a state in which we know that a police car is used for confirming an accident without knowing the exact victim or the specific police car, a state which can exist in real-world environments.

The relaxations obtained in the previous examples will aid in determining similarities between top-level goals based on the location of police cars and of the accident victims. This in turn will allow us to group the goals in a way that, as we will show in the evaluation section, generates solutions with efficient makespans.

4.3 Landmarks Extraction and Relaxation

Section Overview In this section, we show the landmarks extraction procedure and define the relaxed landmarks. Landmarks will be used in the decomposition procedure of the dead-end agent goals (detailed in Chapter 6).

4.3.1 Landmarks Extraction

Definition 4.55. L_g is the set of landmarks composed of dead-end agent goal g and all unique landmarks obtained when backchaining from g while ignoring time points.

Definition 4.56. An *agent landmark* is a landmark which has at least one agent among its parameters.

Grouping all landmarks extracted by backchaining from a specific top-level goal is a core component of the work presented in this thesis. The landmarks are extracted using a slightly modified version of the backchaining technique described in Karpas et al. 2015 [42]. The procedure (Algorithm 10 which computes in polynomial time) conducts an individual backchaining operation [50, 42] for each top-level goal present in goal state G that is a dead-end agent goal $g_{d\alpha_t}$ (lines 1 to 4). Each $g_{d\alpha_t}$ constitutes the initial landmark of its corresponding backchaining operation.

Algorithm 10 Algorithm for Landmarks Extraction

```
  Input: $\Pi$
  Output: A set of landmarks in $\Pi$ and all $L_g$ sets of each $g_{d\alpha_t}$ in $\Pi$
 1: for all dead-end agent goals $g_{d\alpha_t} \in G$ when $G$ in $\Pi$ do
 2:     create set $L_g$
 3:     add $g_{d\alpha_t}$ to $L_g$
 4:     begin backchaining from $g_{d\alpha_t}$
 5:     for each new derived landmark $l$ do
 6:         if $l$ ignoring time points $\notin L_g$ then
 7:             add $l$ to $L_g$
 8:         else
 9:             stop backchaining
10:         end if
11:     end for
12: end for
13: return all found landmarks and all $L_g$ sets
```

It is important to note that exploring the duplicate landmarks among all top-level goals could potentially yield interesting decompositions, but this avenue represents a future area of research and was not explored in this thesis. In our work, we only backchain from dead-end agent goals as the landmarks-based decomposition technique (Chapter 6) aims to decompose only the dead-end agent goals. In Karpas et al. 2015, all discovered landmarks are stored in the same set and compared against each other during backchaining. However, our decomposition technique relies on identifying duplicate landmarks between dead-end agent goals (detailed in Chapter 6 Section 6.4.1.1). Therefore, we create a set L_g for each dead-end agent goal $g \in G$ where we store g and all its derived fact and action landmarks (lines 5 to 7). Each backchaining operation is halted as soon as a newly derived landmark is already found in L_g if time points are ignored (lines 8 and 9). If we had stored all landmarks in the same set (like the original Karpas et al. 2015 technique), then the landmarks extraction operation would have stopped at the first discovered duplicate landmark and the required duplicate landmarks between dead-end agent goals needed for the decomposition procedure would have been missed. Algorithm 10 always outputs correct landmarks, but it might not identify all landmarks present in a planning problem (due to the inherited incompleteness of backchaining operations).

4.3.2 Relaxed Landmarks and Relaxed Agent Landmarks

Section Overview After the completion of the landmarks extraction procedure, we evaluate all fact and action landmarks to determine which of them are compatible for relaxations and mark the ones that do as *relaxed agent landmarks*.

4.3.2.1 Relaxed Agent Fact Landmarks

Definition 4.57. A *relaxed fact landmark* is a fact landmark formed only of relaxed propositions.

Definition 4.58. A *relaxed agent fact landmark* is a relaxed fact landmark formed only of relaxed agent propositions.

The procedure for relaxing fact landmarks is shown in Algorithm 11, an algorithm with a supportive role which computes in polynomial time. We parse all fact landmarks (line 1) identified in the landmarks extraction procedure (Algorithm 10), search their parameters for agents (line 2) and replace the found agents with their respective agent types (lines 4 to 6).

Example 4.59. During the landmark extraction procedure of the RTAM_5_1_35 planning problem, we identify the fact landmark *(at acc_victim0 accident_location2)*. Parsing the parameters of the landmark, we identify *acc_victim0* as a agent with the *acc_victim* agent type, therefore the landmark is compatible for relaxation. We then replace *acc_victim0* with its type, *acc_victim* to obtain the relaxed agent fact landmark *(at acc_victim accident_location2)*.

Algorithm 11 Algorithm for Relaxing Fact Landmarks

```
Input: All $\alpha_t$, $t_\alpha$ and all fact landmarks extracted from $\Pi$ with Algorithm 10
Output: The relaxed agent fact landmarks corresponding to all fact landmarks extracted
from $\Pi$ with Algorithm 10
1: for all fact landmarks $l$ extracted from $\Pi$ do
2:     if $\exists$ parameter $p \in l$ such that $p$ == agent then
3:         $l$ == relaxed agent fact landmark
4:         for all parameters $p \in l$ do
5:             if type of $p$ == agent type then
6:                 $p$ = type of $p$
7:             end if
8:         end for
9:     end if
10: end for
```

4.3.2.2 Relaxed Agent Action Landmarks

Definition 4.60. A *relaxed action landmark* is an action landmark formed only of relaxed events.

Definition 4.61. A *relaxed agent action landmark* is a relaxed action landmark formed of only relaxed agent events.

Action landmarks are relaxed via the procedure described in Algorithm 12 (an algorithm with a supportive role which computes in polynomial time), similar to the relaxation of fact landmarks. We parse all action landmarks (line 1) identified in the landmarks extraction procedure (Algorithm 10), search their parameters for agents (line 2) and replace the found agents with their respective agent types (lines 4 to 6).

Example 4.62. When running the landmark extraction procedure of planning problem RTAM_5_1_35, we identify the action landmark *START(confirm_accident police_car0 acc_victim0 accident_location2).* Parsing the parameters of the landmark, we identify *acc_victim0* and *police_car0* as agents, therefore the landmark is compatible for relaxation. We then replace *acc_victim0* with its type, *acc_victim*, and replace *police_car0* with its type, *police_car* to obtain the relaxed agent action landmark *START(confirm_accident police_car acc_victim accident_location2).*

Algorithm 12 Algorithm for Relaxing Action Landmarks

```
Input: All $\alpha_t$, $t_\alpha$ and all action landmarks extracted from $\Pi$ with Algorithm 10
Output: The relaxed agent action landmarks corresponding to all action landmarks
extracted from $\Pi$ with Algorithm 10
 1: for all action landmarks $l$ extracted from $\Pi$ do
 2:     if $\exists$ parameter $p \in l$ such that $p ==$ agent then
 3:         $l ==$ relaxed agent action landmark
 4:         for all parameters $p \in l$ do
 5:             if type of $p ==$ agent type then
 6:                 $p =$ type of $p$
 7:             end if
 8:         end for
 9:     end if
10: end for
```

4.3.3 Extracting Relaxed Agent Landmarks in a Connected Map

Section Overview In this section, we present an improved extraction procedure for relaxed landmarks when a connected map is present in the planning problem.

Definition 4.63. A *unique parent agent group* $\mu_{p\alpha_t}$ represents a set of parent agents $p\alpha_t$ in Π where $\mu_{p\alpha_t}$ contains only one $p\alpha_t$ for each parent type $t \in T_d$ when t represents the type of one or more parent agents defined in Π.

Agents provide the option to improve the relaxed agent landmarks extraction process when a connected map [57] is present in a planning problem Π and when there are no irreversible state transitions in Π. If the map is connected, then a single unique parent agent group $\mu_{p\alpha_t}$ is capable of reaching every part of the map and is sufficient to solve all dead-end agent goals [59] (provided there are no irreversible state transitions in Π). Therefore, using a unique parent agent group as the only agents in Π is sufficient to extract at least the same number of landmarks as in the case of using multiple or all parent agents with the same type when backchaining from the same dead-end agent goals. This is possible because regardless if we use multiple parent agents per type or a single parent agent per type when extracting relaxed landmarks, all found agent landmarks based on the same predicate or the same action and with identical non-agent parameters will get reduced to a single landmark when relaxed.

The procedure for extracting landmarks when a connected map is present in Π and when there are no irreversible state transitions in Π(Algorithm 13 which computes in polynomial time and is an optional component of the thesis) removes all but one of the parent agents with the same dynamic type from the planning problem (lines 1 to 7). When removing a parent agent, we remove all propositions, functions and goals which contain one or more parameters consisting of the target parent agent from the planning problem (line 6). Afterwards, we follow the same steps described in Algorithm 10 (line 11). Similarly to Algorithm 10, Algorithm 13 always outputs correct landmarks, but it might not identify all landmarks present in a planning problem (due to the incompleteness of backchaining operations).

Algorithm 13 Algorithm for Landmarks Extraction in a Connected Map

```
Input: $\Pi$
Output: A set of landmarks in $\Pi$ (must be relaxed after) and all $L_g$ sets of each $g_{d\alpha_t}$ in $\Pi$
1: if connected_map ($\Pi$) == true and no irreversible state transitions in $\Pi$ then
2:     for all types $t \in T_d$ when $T_d$ in $\Pi$ do
3:         $n_{\alpha_t}$ = number of $\alpha_t$ in $\Pi$ when $\alpha_t$ is a parent agent
4:         while $n_{\alpha_t} > 1$ do
5:             $\neg\alpha_t$ = random parent agent $\alpha_t$ in $\Pi$
6:             remove $\neg\alpha_t$ and all elements which contain one or more $\neg\alpha_t$ from $\Pi$
7:             $n_{\alpha_t}$ = number of $\alpha_t$ in $\Pi$ when $\alpha_t$ is a parent agent
8:         end while
9:     end for
10: end if
11: run Algorithm 10
```

Example 4.64. In the *RTAM_5_1_35* planning problem, all the *tow_truck* dynamic type agents are : *tow_truck0*, *tow_truck1*, and *tow_truck2*. The agents are used to populate the initial state with facts:

(= (speed tow_truck0) 0.8)
(= (speed tow_truck1) 0.8)
(= (speed tow_truck2) 0.8)
(available tow_truck0)
(available tow_truck1)
(available tow_truck2)
(at tow_truck0 garage_halifax)
(at tow_truck1 garage_huddersfield)
(at tow_truck2 garage_brighouse)

The agents are also used to populate the goal state with goals:

(at tow_truck0 garage_halifax)
(at tow_truck1 garage_huddersfield)
(at tow_truck2 garage_brighouse)

Running the first part of Algorithm 13 on the *RTAM_5_1_35* planning problem and stopping right before running Algorithm 10 will yield a new planning problem with only one random agent of the *tow_truck* parent type: *tow_truck0* for the purpose of this example. All facts with parameters of the *tow_truck* parent type corresponding to the agents that were discarded during the random selection have been removed from the new planning problem. The remaining initial state facts with *tow_truck* dynamic type parameters are:

(= (speed tow_truck0) 0.8)
(available tow_truck0)
(at tow_truck0 garage_halifax)

All goals with parameters of the *tow_truck* parent type corresponding to the agents that were discarded during the random selection have also been removed from the new planning problem. The remaining goal with *tow_truck* dynamic type parameters is:

(at tow_truck0 garage_halifax)

Example 4.65. Running Algorithm 10 on the *RTAM_5_2_35* planning problem discovers the ν action landmark which is a disjunction of multiple *START* events :

ν *= OCCURS (START(confirm_accident police_car0 acc_victim0 accident_location2)*
START(confirm_accident police_car1 acc_victim0 accident_location2)
START(confirm_accident police_car2 acc_victim0 accident_location2)
START(confirm_accident police_car3 acc_victim0 accident_location2)
START(confirm_accident police_car4 acc_victim0 accident_location2))

Backchaining from ν using the third derivation rule [42] means exploring the conditions of each of the events to see if they are common among all disjunctive events. Each event in ν has its conditions corresponding to the parameter format of the *confirm_accident* action (Figure 4.6).

```
(:durative-action confirm_accident
     :parameters (?V - police_car ?P - subject ?A - accident_location)
     :duration (= ?duration 10)
     :condition (and
        (at start (at ?V ?A))
        (at start (at ?P ?A))
        (at start (uncertified ?P)))
     :effect (and
        (at start (not (uncertified ?P)))
        (at end (waiting ?P))
        (at end (certified ?P)))
)
```

Fig. 4.6 RTAM domain *confirm_accident* action

The conditions with the predicate *(at start (at ?subject ?accident_location))* are the same in all events, as the *acc_victim0* and *accident_location2* agents are the condition parameters for all of the events. Therefore, the fact landmark *HOLDS ((at acc_victim0 accident_location2))* gets extracted and a new backchaining operation is started from it. The conditions with the predicate *(at start (uncertified ?subject))* are also the same in all events, as the *acc_victim0* agent is the condition parameter for all of the events. Therefore, the fact landmark *HOLDS ((uncertified acc_victim0))* also gets extracted and a new backchainig operation is started from it.

However, the conditions with the predicate *(at start (at ?police_car ?accident_location))* are not the same in all events, as each event must have its respective *police_carX* agent as a condition parameter. Therefore, a fact landmark with the *(at start (at police_carX accident_location2))* format will not get extracted when backchaining from v.

On the other hand, *RTAM_5_2_35* has a connected map, so Algorithm 13 is compatible for extracting landmarks. Running Algorithm 13 on the *RTAM_5_2_35* planning problem no longer discovers the v disjunctive action landmark as we no longer have multiple agents with the *police_car* dynamic type. Instead, we discover the λ action landmark represented by the only agent with the *police_car* dynamic type - *police_car0* for the purpose of this example:

λ = *OCCURS (START(confirm_accident police_car0 acc_victim0 accident_location2)*

Backchaining from λ using the third derivation rule extracts the *HOLDS ((at acc_victim0 accident_location2))* and *HOLDS ((uncertified acc_victim0))* fact landmarks as before. However, the backchaining also extracts the *HOLDS ((at police_car0 accident_location2))* fact landmark, as we have only one event in λ and, therefore, only one condition with the predicate *(at start (at ?police_car ?accident_location))* in the derivation of λ. The additionally discovered *HOLDS ((at police_car0 accident_location2))* fact landmark triggers its own backchaining operation which adds even more landmarks which in turn will trigger more backchaining operations (details in the Evaluation chapter).

It is important to note that Algorithm 13 can be applied only when the aim is to relax all compatible landmarks, as in this case both ν and λ will get relaxed to the identical *OCCURS (START(confirm_accident police_car acc_victim accident_location2)* relaxed agent action landmark. Therefore, using one or multiple agents for the *police_car* dynamic type will provide the same contribution to the tactical decomposition process (Chapter 6). Also, the additionally discovered *HOLDS ((at police_car0 accident_location2))* will get relaxed to the *HOLDS ((at police_car accident_location2))* relaxed agent fact landmark, which is representative for the relaxation of any of the *police_car* agents, but would not constitute a valid landmark without relaxation.

4.4 Summary of Chapter Contributions

In this chapter, we introduced new formally defined AI planning technical concepts along with automated extraction procedures that generate information to be used as 'advice' for evaluating the difficulty of a problem (described in Chapter 5) and for decomposing and abstracting a planning problem (described in Chapter 6).

Specifically, we used dynamic and static objects to identify and formally define the agents, the inactive dynamic objects, the necessary static objects and the unnecessary static objects of a planning problem (Sections 4.1.1 and 4.1.2). We presented a problem 'cleaning' technique for removing potentially redundant elements that can interfere with the extraction of the 'advice' and with the solution search (Section 4.1.3).

We illustrated a technique for extracting the dependency relationships among agents based on the reachability impact observed when removing all agents with the same type from a planning problem (Section 4.1.4). We also showcased a fast procedure for mitigating potential conflicting agent dependency relationships (Section 4.1.5). Furthermore, we used the agent dependency relationships to create techniques that classify the agents according to dependency status and hierarchical positioning within the dependency graph (Section 4.1.6).

We introduced the 'relaxed proposition' and 'relaxed event' PDDL encodings which support both grounded and lifted parameters in order to allow the representation of partially complete environment information in PDDL (Section 4.2). The encodings were used to construct a new type of landmark 'relaxed' based on the agents it may contain (Section 4.3.2).

Furthermore, we provided a landmarks extraction technique (Section 4.3) that extracts the landmarks from each specific top-level goal in order to identify the duplicate landmarks among top-level goals (which are to be used as criteria for goal clustering from PDDL encoded data, as described in Chapter 6 Section 6.4.1.1). We also presented a faster extraction technique for relaxed landmarks in problems that have a connected map and no irreversible state transitions (Section 4.3.3).

Chapter 5

Estimating the Difficulty of Planning Problems

Chapter Overview In this chapter, we use the agent classifications and the agent dependency relationships defined in the previous chapter as well as the entanglement among objects to identify and formalise potential difficulty metrics for evaluating the difficulty of a planning problem.

5.1 Entangled Objects

Section Overview The objects in a planning problem can be classified based on their roles as parameters of the same executable action of a planning problem. Two or more objects are considered entangled if their instantiated types represent the types of the parameters of the same action in at least one executable action of the planning problem. We only consider executable actions, as actions that are not executable will never expand the state space during a planner solution search.

Definition 5.1. An executable action $a \in A$ when A in Π is an action that is present in at least one action layer of the fully expanded relaxed planning graph of a planning problem Π.

Definition 5.2. An *instantiated type* t_i is a type $t \in T$ that has at least one object instance with type t in a planning problem Π.

Definition 5.3. Two or more instantiated types $t_{i1}, t_{i2}, \dots, t_{in} \in T_d$ are entangled if there is at least one executable action $a \in A$ when A in Π that has at least a parameter with each of the types $t_{i1}, t_{i2}, \dots, t_{in}$. An instantiated type $t_i \in T_d$ can be entangled with itself if there is an action $a \in A$ when A in Π that has two or more parameters with type t_i.

Definition 5.4. Two or more objects are entangled if they are parameters in at least one precondition or at least one effect of at least one executable action $a \in A$ when A in Π.

The above usage of the *entangled* term should not be confused with other AI Planning concepts that use the same word for their identification - such as in the work of Chapra et al. (2009).

Example 5.5. In the DLOG-5-5-10 planning problem, all truck and package agents are entangled objects as they are both parameters in the *LOAD-TRUCK* and *UNLOAD-TRUCK* actions and the two actions are executable actions as they appear in the fully expanded relaxed planning graph of the planning problem. If, however, we consider a version of the DLOG-5-5-10 planning problem that has only one truck located at a new location that has no packages and is disconnected from all other locations that have packages, then the action *LOAD-TRUCK* and *UNLOAD-TRUCK* will not be part of the fully expanded relaxed planning graph of the modified DLOG-5-5-10 planning problem as the truck will never be in the same location as any of the packages to be able to perform a loading operation and will never have a loaded package to perform an unloading operation. In this case, the truck and package agents are not classified as entangled objects even though they are both parameters in the *LOAD-TRUCK* and *UNLOAD-TRUCK* actions as the actions are not executable.

5.2 Dynamic Objects Test Problems

Section Overview This section presents the intuition behind the difficulty estimation approach as well as describes the structure of the test problems used to conduct the difficulty analysis.

The IPC 2014/2018 planning problems and results show that planners generally stop finding solutions to problems that have an increased number of initial state facts and goals relative to the number of initial state facts and goals in the solved problems. However, identifying generic planning problem difficulty metrics is not straightforward as it is possible for a problem which is hard to solve by one type of planner to be easily solved by another planner that uses a different technique. To tackle this issue, we have opted for a specific, rather than a generic, approach for identifying some of the scalability difficulty metrics of planning problems. By focusing on a specific planning problem with a structure similar to many real-world problems when solved by multiple state-of-the-art temporal planners we obtain difficulty metrics that are potentially compatible with other similar planning problems. Therefore, we have created a set of Driverlog test problems in which we systematically modify the number of dynamic objects and attempt to solve the test problems with the Optic, Itsat, Temporal Fast Downward (TFD) and Yahsp3 temporal planners in order to empirically identify some of the difficulty metrics of Driverlog. As we will show in the next chapters, the difficulty metrics obtained from the Driverlog test problems also apply to other planning problems with a similar structure to Driverlog.

All dynamic objects test problems have the static environment of the DLOG-5-5-10 IPC 2014 benchmark problem. The test problems are divided into two batches. In the first batch, we gradually increase the agents and corresponding agent facts in each problem and add an *(at agent location)* agent goal (as the goals in the IPC benchmark problems) for each extra-added agent. In the second batch, we gradually increase the dynamic objects in each problem without adding any corresponding achievable goals for the extra added dynamic objects. The dynamic objects, agents and agent goals are added evenly among all ten locations present in the DLOG-5-5-10 problem with no agent having its initial state location the same as its goal state location. In both batches, the test problems are divided into seven sets which cover all possible combinations of Driverlog dynamic types:

1. Set **P** contains problems focused only on dynamic objects with the package dynamic type in which we gradually increase the number of packages.
2. Set **T** contains problems focused only on dynamic objects with the truck dynamic type in which we gradually increase the number of trucks.
3. Set **D** contains problems focused only on dynamic objects with the driver dynamic type in which we gradually increase the number of drivers.
4. Set **TP** contains problems focused on dynamic objects with the truck dynamic type and package dynamic type in which we gradually increase the number of trucks and packages.
5. Set **DP** contains problems focused on dynamic objects with the driver dynamic type and package dynamic type in which we gradually increase the number of drivers and packages.
6. Set **DT** contains problems focused on dynamic objects with the driver dynamic type and truck dynamic type in which we gradually increase the number of drivers and trucks.
7. Set **DTP** contains problems focused on all three driver, truck and package dynamic types in which we gradually increase the number of drivers, trucks and packages.

Each test problem is executed on a Dell XPS 15 9560 laptop with 32GB total and unrestricted RAM and a threshold of 300 seconds. The purpose of the test is to evaluate the impact of the number of dynamic objects in each test problem on the capacity of planners to find a solution in the allocated time.

5.3 Difficulty Impact of Agents Present in the Goal State

Section Overview In this section, we analyse the impact of the number of agents that are present in the goal state of a planning problem on the difficulty of the problem.

In the first batch, we focused on the impact of adding agents and agent facts together with a corresponding *(at agent location)* goal (as the goals in the IPC benchmark problems) for each added agent. Each test problem will only have the minimum required dynamic objects for the problem to be solvable. For example, all problems in set T will have a driver agent with no corresponding goal as a truck needs a driver to travel between locations. In set D, however, we will have no trucks or packages, as drivers do not need a truck or a package to travel between locations.

Each of the seven sets has gradually increasing problems with a maximum of 60 added agents and 60 corresponding achievable goals per problem regardless of the number of targeted types in the set so we can accurately compare the impact of the type of agents on the number of solved problems in each of the seven sets.

In sets P, T, and D we focus only on one type so we increase each problem by one agent with a total of 60 problems per set with each problem having a score of 1 point (one point per added agent) with a total set score of 60.

In sets TP, DP, and DT we focus on two types so we increase each problem by two agents, one for each type, with a total of 30 problems per set with each problem having a score of 2 points (one point per added agent) with a total set score of 60.

In set DTP we focus on three types so we increase each problem by three agents, one for each type, with a total of 20 problems with each problem having a score of 3 points (one point per added agent) in order to maintain the same number of total agents and the same total score of 60 as in the other sets.

All dynamic objects added in the first batch are agents as they have corresponding achievable goals in the goal state so they will become parameters in the preconditions of the relaxed plan of their respective single dynamic object per dynamic type planning problem Π_{do}. However, the agents in each set might be parent, dead-end, or regular agents depending on specific dependency relationships in each set.

The results of running the Driverlog Test Problems on the selected planners can be seen in Figure 5.1, Table 5.1.

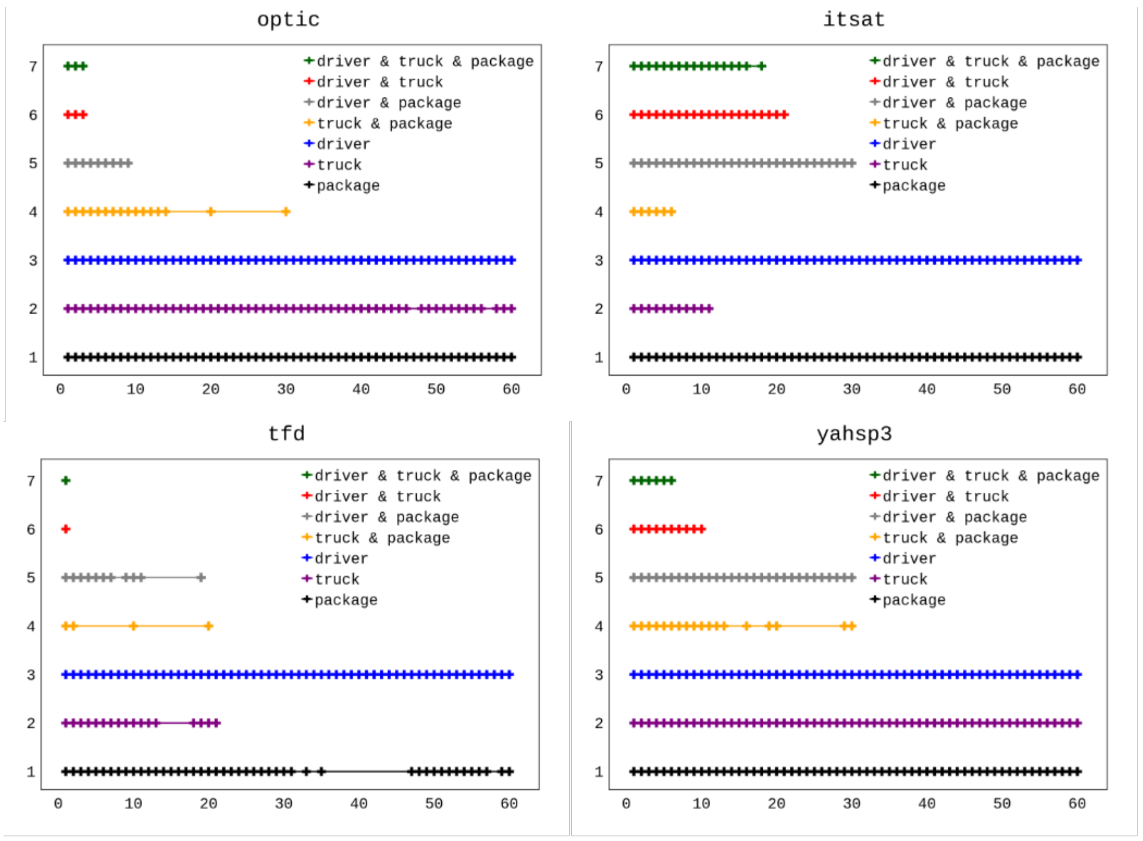


Fig. 5.1 Problems From the First Batch Solved By Each Planner within a Threshold of 300 Seconds. X Axis: Number of Added Agents with corresponding achievable agent goals in the Problems Solved by Each Planner. Y Axis: Set Id.

Set Name	**P**	**T**	**D**	**T P**	**D P**	**D T**	**D T P**
No of Sampled Problems	60	60	60	30	30	30	20
Agent Gradual Increase	1	1	1	2	2	2	3
Goals Gradual Increase	1	1	1	2	2	2	3
Score per Problem	1	1	1	2	2	2	3
Optic Score	60	58	60	32	18	6	9
Itsat Score	60	11	60	12	60	42	51
TFD Score	46	17	60	8	22	2	3
Yahsp3 Score	60	60	60	36	60	20	18
Total Obtained Score	226	146	240	88	160	70	81
Total Possible Score	240	240	240	240	240	240	240
% Solved Problems	**94.17%**	**60.83%**	**100.00%**	**36.67%**	**66.67%**	**29.17%**	**33.75%**

Table 5.1 Results of the solving attempts of the first batch of test problems for each planner when the minimum gradual increase is considered.

The package agents in set P are dead-end agents due to the *driver* $\longmapsto$ *package* and *truck* $\longmapsto$ *package* dependency relationships of each test problem in set P. The results of set P show a minimal decrease in the total possible solved planning problems with a 94.17% success rate. Therefore, increasing the number of dead-end package agents with corresponding achievable agent goals mildly increases the difficulty of a planning problem.

The driver agents in set D are neither parent nor dead-end agents as there are no dependency relationships present in any of the test problems in set D. The results of set D show no decrease in the total possible solved planning problems with a 100.00% success rate. However, the planning time of the test problems in set D tends to increase along with the number of agents per problem for all the planners. To accurately compare the planning time increase among distinct planners, we have normalised the planning times of each planner so that each entry lies in a range between 0 and 1. This has been achieved by applying the following formula to each planning time θ obtained by each planner p:

$$\theta = (\theta - min_p) / (max_p - min_p)$$

In the above formula, min_p is the minimum planning time obtained by p in all test problems from a specific set and max_p is the maximum planning time obtained by p in all test problems from the same specific set. The normalised planning times (Figure 5.2) illustrate a similar rate of increase in all planners at every test problem, which shows that the difficulty increase is universal irrespective of the employed (distinct) solving technique and dependent on the number of added elements. Therefore, increasing the number of driver agents with corresponding achievable agent goals increases the difficulty of a planning problem even if the agents are not in a dependency relationship. The similarity of the solving rate between set D and set P is because the agents added in both sets are entangled only in executable actions with three parameters.

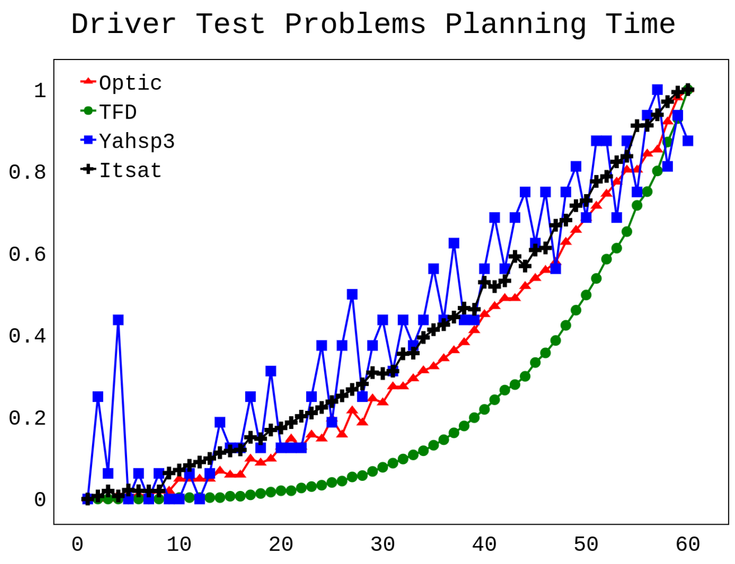


Fig. 5.2 Normalised Planning Time of the Test Problems in Set D. X Axis: Number of Driver Agents per Problem. Y Axis: Normalised Planning Time

The truck agents in set T are also dead-end agents due to the $driver \longmapsto truck$ dependency relationship and due to $truck \longmapsto package$ not being present in the test problems of set T. The results of set T show a significant decrease in the total possible solved planning problems with only a 60.83% success rate. Therefore, increasing the number of dead-end truck agents with corresponding achievable agent goals significantly increases the difficulty in comparison to increasing the number of dead-end package agents with corresponding achievable goals from set P when the total number of agents is the same. The significant difference between the success rate of set T and the rate of sets D and P is because in set T the added truck agents are entangled along three other objects in the *DRIVE-TRUCK* executable action which expands the preprocessing and search operation more than the added package agents in set P and the added driver agents in set D that are only entangled in executable actions with three parameters.

In set DP, the driver agents are parent agents and the package agents are dead-end agents due to the *driver* $\longmapsto$ *package* dependency relationship of each test problem in set DP. The results of set DP show a significant decrease in the total possible solved planning problems with only a 66.67% success rate. Therefore, increasing the driver parent agents with corresponding achievable goals together with the package dead-end agents with corresponding achievable goals significantly increases the difficulty.

In set TP, the truck agents are parent agents and the package agents are dead-end agents due to the *truck* $\longmapsto$ *package* dependency relationship present in each test problem in set TP. The results of set TP show a drastic decrease in the total possible solved planning problems with only a 36.67% success rate. Therefore, increasing the truck parent agents with corresponding achievable goals together with the package dead-end agents with corresponding achievable goals drastically increases the difficulty.

The significant difference between the success rate of set DP and set TP, even though they have the same number of parent types and dead-end types and the same ratio and quantities of agents is because the truck parent agents and package dead-end agents increased in set TP are entangled objects as they are both parameters in the *LOAD-TRUCK* and *UNLOAD-TRUCK* executable actions and exponentially expand the preprocessing and state space during a planner solution search as a consequence. In contrast, the driver parent agents and package dead-end agents increased in set DP are not entangled objects as there is no action in Driverlog that has both the driver and package parameters so the preprocessing and expansion of the state space are not as severe as in set TP.

In set DT, the truck agents are dead-end agents and the driver agents are parent agents due to the *driver* $\longmapsto$ *truck* dependency relationship present in each test problem in set DT. The results of set DT show a drastic decrease in the total possible solved planning problems with only a 29.17% success rate. The driver parent agents and truck dead-end agents increased in set DT are entangled objects as they are both parameters in the four-parameter *DRIVE-TRUCK* executable action and exponentially expand the preprocessing and state space during a planner solution search as a consequence. Therefore, even if we have the same parent to dead-end agent ratio as in sets TP and DP, the difficulty is significantly more increased when we increase two types of entangled objects in an executable action with four parameters than when we increase two types of entangled agents when the executable actions have three parameters when the total number of agents is the same.

In set DTP all three dependency relationships are present, so the packages agents are dead-end agents and the truck and driver agents are parent agents. The results of set DTP show a drastic decrease in the total possible solved planning problems with only a 33.75% success rate. The driver and truck parent agents increased in set DTP are entangled objects and the truck parent agents and package dead-end agents increased in set DTP are also entangled objects. The slightly better results in set DTP than in set DT can be attributed to having fewer added agents entangled in the four-parameter *DRIVE-TRUCK* executable action in set DTP than in set DT.

The overall results in the first batch show that an increase in the number of agents with corresponding achievable agent goals can lead to a significant decrease in the total possible solved planning problems particularly when the agents are entangled objects. The overall results also show that increasing multiple types of agents and achievable corresponding achievable agent goals can lead to a drastic decrease in the total possible solved planning problems particularly when the types of the increased agents are entangled types.

5.4 Difficulty Impact of Agents and Inactive Dynamic Objects not Present in the Goal State

Section Overview In this section, we analyse the impact of the number of agents and inactive dynamic objects that are not present in the goal state of a planning problem on the difficulty of the problem.

In the second batch, we focus on the impact of dynamic objects without corresponding goals on the difficulty of a planning problem. Each test problem has a mandatory agent per dynamic type with corresponding initial state facts and goals. Then, each test problem gets added extra dynamic objects and facts without corresponding goals. Not adding corresponding achievable goals increases the required number of extra added dynamic objects by one order of magnitude in order to obtain a noticeable impact on the difficulty of the test problems in comparison to the first batch.

In sets P, T, and D we focus only on one type so we increase each problem by ten dynamic objects with a total of 60 problems per set with each problem having a score of 10 points (one per extra added agent) with a total set score of 600.

In sets TP, DP, and DT we focus on two types so we increase each problem by two dynamic objects, one for each type, with a total of 30 problems per set with each problem having a score of 20 points (one per extra added dynamic object) with a total set score of 600.

In set DTP we focus on three types so we increase each problem by three dynamic objects, one for each type, with a total of 20 problems with each problem having a score of 30 points (one per extra added dynamic object) in order to maintain the same number of total dynamic objects and the same total score of 600 as in the other sets.

The extra *driver* and *truck* dynamic objects added in the problems of the second batch are parent agents as their types are parent types and there are no elements in any of the problems to prevent any of the parent agents from becoming parameters in the preconditions of the relaxed plan of their respective single dynamic object per dynamic type planning problem Π_{do_t}. The extra added *package* dynamic objects, however, are inactive dynamic objects as they have no corresponding achievable goals in the goal state of the test problems so they will not become parameters in the preconditions of the relaxed plan of their respective single dynamic object per dynamic type planning problem Π_{α_t}. Each set has a maximum of 600 added dynamic objects per problem regardless of the number of targeted agent types in the set so we can accurately compare the impact of the type of agents on the number of solved problems in each of the seven sets.

The results of running the Driverlog Test Problems on the selected planners can be seen in Figure 5.3, Table 5.2.

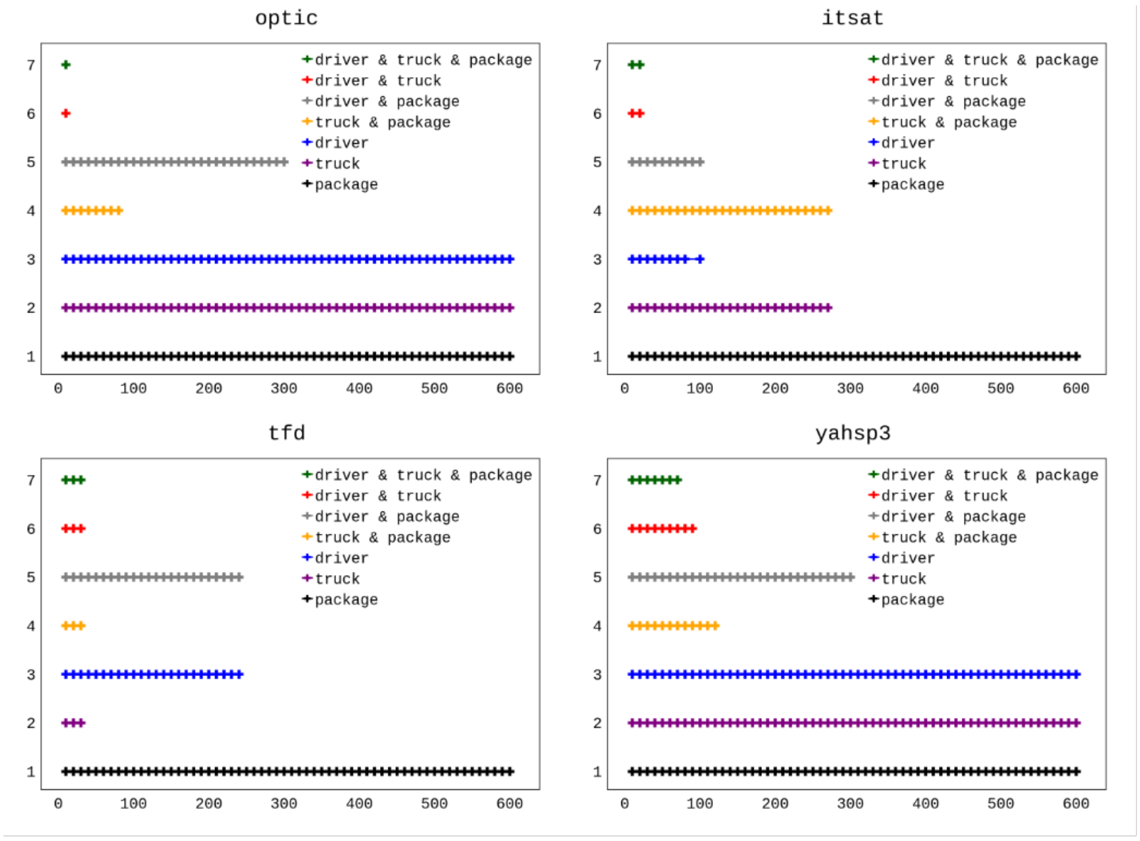


Fig. 5.3 Problems From the Second Batch Solved By Each Planner within a Threshold of 300 Seconds. X Axis: Number of Added Agents without corresponding achievable agent goals in the Problems Solved by Each Planner. Y Axis: Set Id.

Set Name	**P**	**T**	**D**	**T P**	**D P**	**D T**	**D T P**
No of Sampled Problems	60	60	60	30	30	30	20
Object Gradual Increase	10	10	10	20	20	20	30
Goal Gradual Increase	0	0	0	0	0	0	0
Score per Problem	10	10	10	20	20	20	30
Optic Score	600	600	600	160	600	20	30
Itsat Score	600	270	100	540	200	40	60
TFD Score	600	30	240	60	480	60	90
Yahsp3 Score	600	600	600	240	600	180	210
Total Obtained Score	2400	1500	1540	1000	1880	300	390
Total Possible Score	2400	2400	2400	2400	2400	2400	2400
% Solved Problems	**100%**	**62.50%**	**64.17%**	**41.67%**	**78.33%**	**12.50%**	**16.25%**

Table 5.2 Results of the solving attempts of the second batch of test problems for each planner.

The results of set P show no decrease in the total possible solved planning problems with a 100% success rate. However, the planning time of the test problems in set P (Figure 5.4) tends to increase along with the number of inactive dynamic objects per problem for most planners. Therefore, increasing the number of package inactive dynamic types without corresponding achievable agent goals impacts the difficulty of a planning problem.

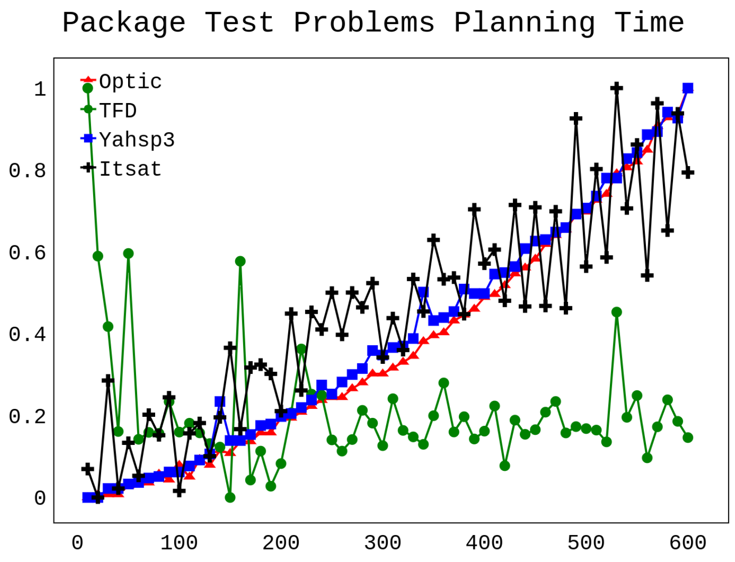


Fig. 5.4 Normalised Planning Time of the Test Problems in Set P. X Axis: Number of Package Inactive Dynamic Objects per Problem. Y Axis: Normalised Planning Time

The results of sets T and D show a significant decrease in the total possible solved planning problems with only a 62.5% and a 64.17% success rate. Therefore, increasing one type of parent agents without corresponding achievable goals significantly increases the difficulty of a planning problem in comparison to increasing the number of one type of inactive dynamic objects without corresponding achievable goals when the total number of dynamic objects is the same. The significant difference between the success rate of sets T and D to that of set P is because in sets T and D the added truck and driver agents are entangled along three other objects in the *DRIVE-TRUCK* executable action which expands the preprocessing and search operation more than the added package inactive dynamic objects in set P that are only entangled in executable actions with three parameters.

The results of set DP show a significant decrease in the total possible solved planning problems with only a 78.3% success rate. Therefore, even if we have a 0.5 ratio of parent agents in comparison to set D where we have the same parent type but with a ratio of 1, increasing one type of parent agents without corresponding achievable goals together with one type of inactive dynamic objects without corresponding achievable goals also impacts the difficulty of a planning problem.

The results of set TP show a drastic decrease in the total possible solved planning problems with only a 41.67% success rate. Therefore, even if we have a 0.5 ratio of parent agents in comparison to set T where we have the same parent type but with a ratio of 1, increasing one type of parent agents without corresponding achievable goals together with one type of inactive dynamic objects without corresponding achievable goals also impacts the difficulty of a planning problem.

The significant difference between the success rate of set DP and set TP is also present in the second batch for the same reason as in the first batch. Even though the two sets have the same number of dynamic types and the same ratio and quantities of agents and inactive dynamic objects, the truck parent agents and package inactive dynamic objects increased in set TP are entangled objects as they are both parameters in the *LOAD-TRUCK* and *UNLOAD-TRUCK* executable actions and exponentially expand the preprocessing and state space during a planner solution search as a consequence. In contrast, the driver parent agents and package inactive dynamic objects increased in set DP are not entangled objects as there is no action in Driverlog that has both the driver and package parameters so the expansion of the preprocessing and state space is not as severe as in set TP. Therefore, even if the package dynamic objects are inactive, they still contribute to the difficulty of the problems in TP due to their entanglement with the truck agents.

The results of set DT show a drastic decrease in the total possible solved planning problems with only a 12.5% success rate. The driver and truck parent agents increased in set DT are entangled objects as they are both parameters in the *BOARD-TRUCK*, *DISEMBARK-TRUCK* and *DRIVE-TRUCK* executable actions and exponentially expand the preprocessing and state space during a planner solution search as a consequence. Therefore, even if we have the same parent agent ratio of 1 as in sets T and D, the difficulty is significantly more increased when we increase two types of entangled objects that are parent agents without corresponding achievable goals in comparison to when we increase only one type of parent agent without corresponding achievable goals when the total number of agents is the same.

The results of set DTP show a drastic decrease in the total possible solved planning problems with only a 16.25% success rate. The driver and truck parent agents increased in set DTP are entangled objects and the truck parent agents and package inactive dynamic objects increased in set TP are also entangled objects. Therefore, even if we have only a 0.66 parent agent ratio in comparison to sets T and D where we have a parent agent ratio of 1, the difficulty is significantly more increased when we increase all three types of entangled objects without corresponding achievable goals in comparison to when we increase only one type of parent agent without corresponding achievable goals when the total number of objects is the same. Also, the slightly better results in set DTP than in set DT can be attributed to having fewer added parent agents entangled in the four-parameter DRIVE-TRUCK executable action in set DTP than in set DT.

The results in the second batch show that an increase in the number of dynamic objects without corresponding achievable agent goals can lead to a decrease in the total possible solved planning problems, particularly when the dynamic objects are entangled objects. The results in the second batch also show that increasing multiple types of dynamic objects without corresponding achievable agent goals can lead to a decrease in the total possible solved planning problems particularly when the types of the increased dynamic objects are entangled types.

Overall, the results in both batches show that the impact of dynamic objects agents without corresponding achievable goals is approximately an order of magnitude lower than the impact of agents with corresponding achievable goals in most of the sets.

5.5 Static Objects Test Problems

Section Overview In this section, we analyse the impact of the number of static objects in a planning problem on the difficulty of the problem.

We will now evaluate the impact of the number of static objects on the difficulty of Driverlog. The static objects in Driverlog are the objects with the *location* static type. We will start with a base test problem that has two locations, reachable by trucks, that are connected with each other via both paths and links. The base test problem also has a single agent with corresponding initial state agent facts and goals for each of the three dynamic types present in Driverlog (Figure 5.5, goals are highlighted in green and the relaxed plan is highlighted in red).

```
(:objects
    S0 - location
    S1 - location

    P0-1 - location

    driver - driver
    truck - truck
    package - obj
)
(:goal (and
    (at driver S1)
    (at truck S0)
    (at package S1)
))
```

```
(:init
    (link S0 S1)
    (link S1 S0)

    (path S0 P0-1)
    (path P0-1 S0)
    (path S1 P0-1)
    (path P0-1 S1)

    (empty truck)

    (at driver S0)
    (at truck S1)
    (at package S0)
)
```

Fig. 5.5 The objects, initial state and goal state of the base test problem.

The single dynamic object per dynamic type planning problem for the driver, truck and package dynamic objects in the base problem will be the same problem as the base problem (as there is only one object per dynamic type) and will have the same relaxed plan as the base problem. Each static object in the base problem is a *necessary static object* as it respects the criteria of being a condition in at least one action in the relaxed plan of at least a single dynamic object per dynamic type planning problem derived from the base problem.

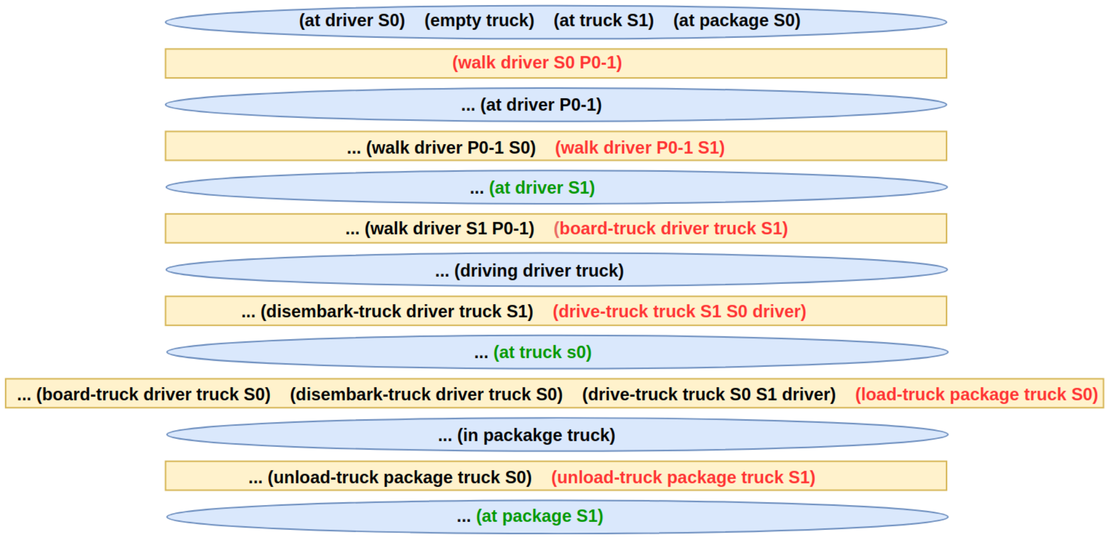
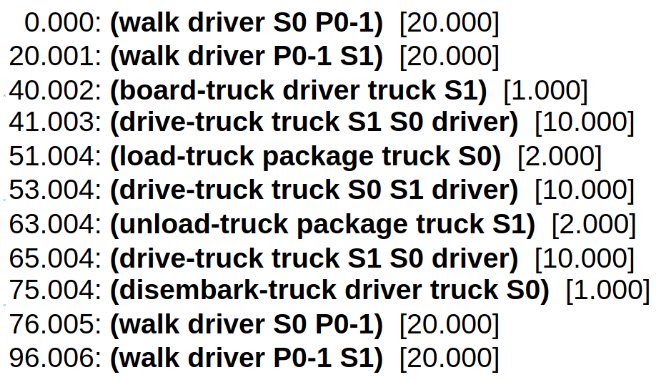

(at driver S0) (empty truck) (at truck S1) (at package S0)
(walk driver S0 P0-1)
... (at driver P0-1)
... (walk driver P0-1 S0) (walk driver P0-1 S1)
... (at driver S1)
... (walk driver S1 P0-1) (board-truck driver truck S1)
... (driving driver truck)
... (disembark-truck driver truck S1) (drive-truck truck S1 S0 driver)
... (at truck s0)
... (board-truck driver truck S0) (disembark-truck driver truck S0) (drive-truck truck S0 S1 driver) (load-truck package truck S0)
... (in packakge truck)
... (unload-truck package truck S0) (unload-truck package truck S1)
... (at package S1)

Fig. 5.6 The relaxed planning graph and relaxed plan (highlighted in red) of the base test problem and of all single dynamic object per dynamic type planning problems derived from the base test problem. '...' represents facts or actions added in the previous fact or action layers. The facts highlighted in green represent the goals of the problem.

The optimal plan for the base test problem can be observed in Figure 5.5.



```
 0.000: (walk driver S0 P0-1)  [20.000]
20.001: (walk driver P0-1 S1)  [20.000]
40.002: (board-truck driver truck S1)  [1.000]
41.003: (drive-truck truck S1 S0 driver)  [10.000]
51.004: (load-truck package truck S0)  [2.000]
53.004: (drive-truck truck S0 S1 driver)  [10.000]
63.004: (unload-truck package truck S1)  [2.000]
65.004: (drive-truck truck S1 S0 driver)  [10.000]
75.004: (disembark-truck driver truck S0)  [1.000]
76.005: (walk driver S0 P0-1)  [20.000]
96.006: (walk driver P0-1 S1)  [20.000]
```

Fig. 5.7 The optimal plan of the base test problem when $\varepsilon = 0.001$.

We then create additional test problems by gradually increasing the number of locations in the base test problem. Each new test problem gets added two new locations reachable by trucks, that are connected with each other and every other location in the problem via both paths and links in order to obtain a set of 20 problems with locations reachable by trucks ranging from 2 to 40. The single dynamic object per dynamic type planning problems derived from each additional test problem will have the same relaxed plan (highlighted in red in Figure 5.8) as the equivalent problems derived from the base test problem: the relaxed plan of the based case problem. Also, the optimal plan for each additional test problem will be the same as the base test problem. This happens because the *drive-truck* and *move* actions in the Driverlog IPC 2014 version have fixed durations and all locations are connected with each other in every problem so there is an equal transportation cost among locations. Therefore, adding extra locations brings no benefit to finding a solution, and all gradually added locations are *unnecessary static objects* (present in the facts highlighted in blue in Figure 5.8) as they do not appear in any condition or effect of any action in the relaxed plan of any single dynamic object per dynamic type planning problem derived from any additional test problem.

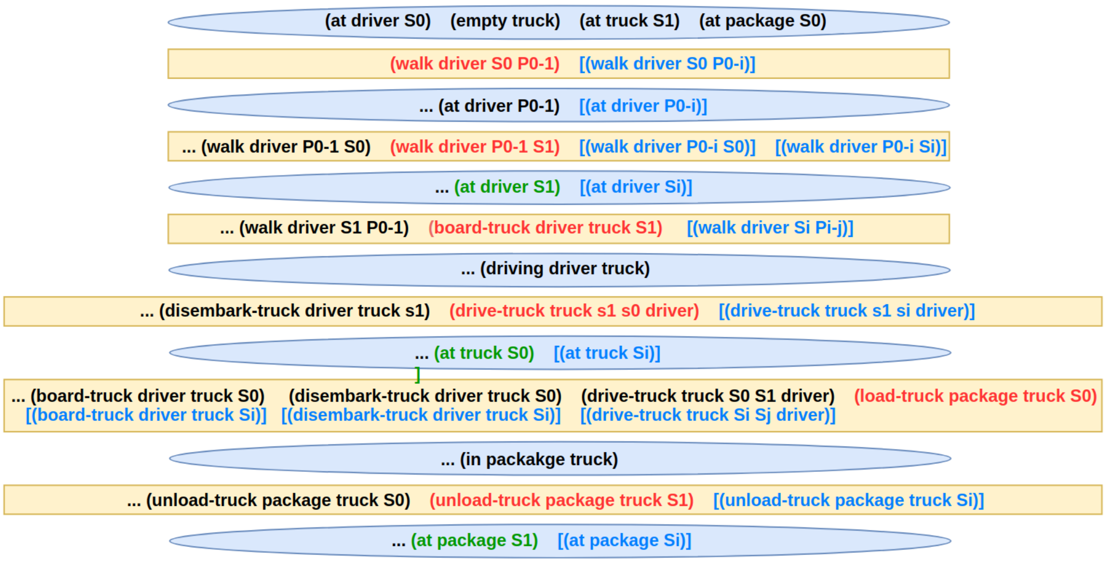

(at driver S0) (empty truck) (at truck S1) (at package S0)
(walk driver S0 P0-1) [(walk driver S0 P0-i)]
... (at driver P0-1) [(at driver P0-i)]
... (walk driver P0-1 S0) (walk driver P0-1 S1) [(walk driver P0-i S0)] [(walk driver P0-i Si)]
... (at driver S1) [(at driver Si)]
... (walk driver S1 P0-1) (board-truck driver truck S1) [(walk driver Si Pi-j)]
... (driving driver truck)
... (disembark-truck driver truck s1) (drive-truck truck s1 s0 driver) [(drive-truck truck s1 si driver)]
... (at truck S0) [(at truck Si)]
... (board-truck driver truck S0) (disembark-truck driver truck S0) (drive-truck truck S0 S1 driver) (load-truck package truck S0)
[(board-truck driver truck Si)] [(disembark-truck driver truck Si)] [(drive-truck truck Si Sj driver)]
... (in packakge truck)
... (unload-truck package truck S0) (unload-truck package truck S1) [(unload-truck package truck Si)]
... (at package S1) [(at package Si)]

Fig. 5.8 The relaxed planning graph and relaxed plan (highlighted in red) of all test problems and of all single dynamic object per dynamic type planning problems derived from all test problems. '...' represents facts or actions added in the previous fact or action layers of the base test problem. Facts and actions in square brackets (highlighted in blue) represent the facts and actions added for the additional test problems ($2 \leq$ i,j $\leq$ max location of additional test problem; i $\neq$ j). The facts highlighted in green represent the goals of the problem.

The test problems are executed with the Optic, Itsat, Temporal Fast Downward (TFD) and Yahsp3 planners on a Dell XPS 15 9560 laptop with 32GB total and unrestricted RAM and unlimited time in order to obtain the makespan and planning time of the first plan and best plan found by each planner for each test problem.

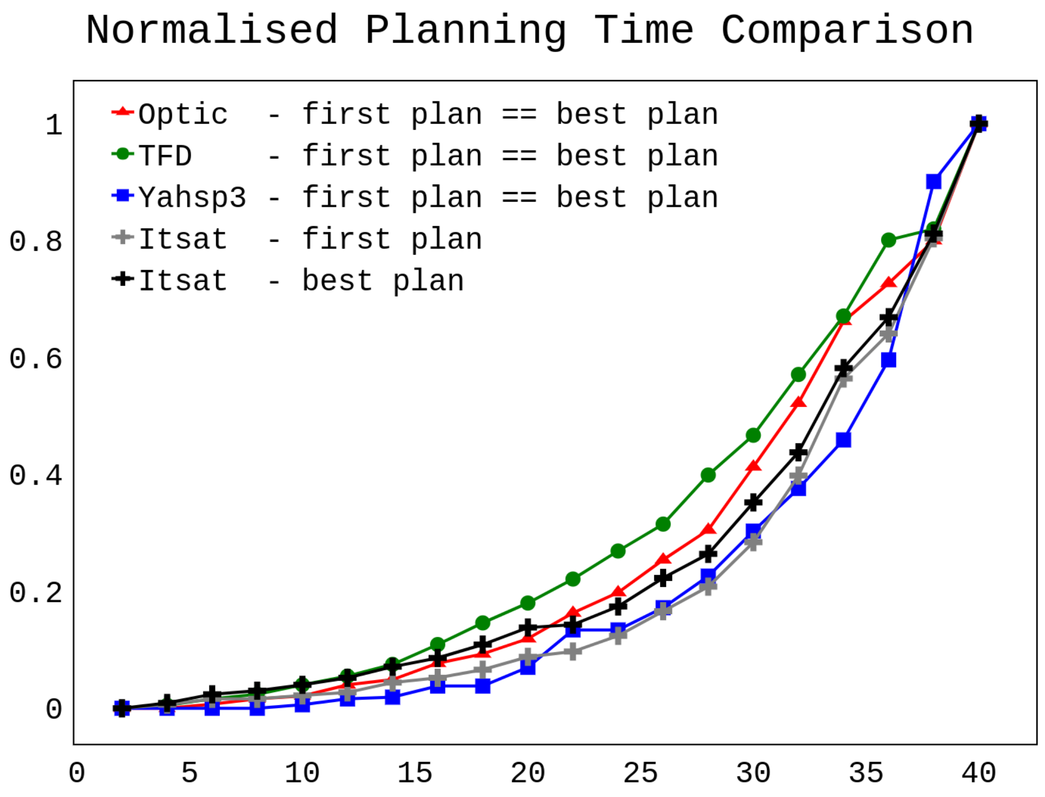


Fig. 5.9 Normalised Planning Time Comparison of Test Problems. X Axis: Number of Locations reachable by trucks. Y Axis: Normalised Planning Time

Optic, TFD and Yahsp3 have their first found plan as their respective best plan while Itsat obtained distinct first and best plans (Table 5.3). The planning time results in all cases (Figure 5.9) clearly show a similar monotonic growth in the normalised planning time as we gradually increase the number of locations per additional test problem even though the locations are unnecessary static objects. The increase in planning time happens because of the entanglement of the gradually added unnecessary static objects with the agents and with each other in the *WALK* and *DRIVE-TRUCK* executable actions as well as the engagement with the agents in all other actions (which are all executable actions in all test problems) from the Driverlog domain. Therefore, increasing the number of unnecessary static objects that are parameters in executable actions can increase the difficulty of a planning problem.

Id	Loc	**Optic**	**TFD**	**Yahsp3**	**Itsat First**		**Itsat Best**	
-	-	Cost	Cost	Cost	Cost	Time	Cost	Time
1	2	116.006	116.010	116.120	156.120	0.463	116.100	0.484
2	4	116.006	116.010	156.140	166.130	0.567	116.100	0.761
3	6	116.006	116.010	156.140	176.160	0.755	116.100	1.232
4	8	116.006	116.010	156.140	226.210	0.757	116.100	1.439
5	10	116.006	116.010	156.140	276.180	0.862	116.100	1.752
6	12	116.006	116.010	156.140	396.240	0.965	116.100	2.104
7	14	116.006	116.010	156.140	516.300	1.270	116.100	2.720
8	16	116.006	116.010	156.140	376.360	1.411	116.100	3.178
9	18	116.006	116.010	156.140	186.170	1.674	116.100	3.901
10	20	116.006	116.010	156.140	476.460	2.074	116.100	4.809
11	22	116.006	116.010	156.140	196.160	2.251	116.100	4.972
12	24	116.006	116.010	156.140	546.530	2.746	116.100	5.952
13	26	116.006	116.010	156.140	466.450	3.524	116.100	7.481
14	28	116.006	116.010	156.140	166.150	4.282	116.100	8.761
15	30	116.006	116.010	156.140	406.370	5.681	116.100	11.505
16	32	116.006	116.010	156.140	426.410	7.782	116.100	14.215
17	34	116.006	116.010	156.140	126.110	10.842	116.100	18.716
18	36	116.006	116.010	156.140	196.160	12.258	116.100	21.440
19	38	116.006	116.010	156.140	386.370	15.259	116.100	25.932
20	40	116.006	116.010	156.140	876.860	18.868	116.100	31.825

Table 5.3 The makespan (cost) of the first and best plan found by each planner for each test problem. Optic, TFD and Yahsp3 have their first found plan as their respective best plan in all test problems. Itsat has distinct first and best plans found in all test problems so we added the planning time (in seconds) for each plan for comparison.

Optic and TFD were able to find the optimal plan (considering each planner-specific epsilon value) as the first found plan (Table 5.3) in all test problems. However, Yahsp3 was able to find the optimal plan (considering its specific epsilon value) only in the base test problem while in all other problems only found a plan with a 34.45% higher cost than the optimal plan even though it was able to perform a complete solution search. Therefore, increasing the number of unnecessary static objects that are parameters in executable actions can affect the quality of the best plan found by Yahsp3 even when it can perform a complete solution search.

Itsat was able to find the optimal plan (considering its specific epsilon value) as the best plan in all test problems (Table 5.3). However, there is a considerable difference between the first plans and best plans found by Itsat in almost all test problems. If, for example, we run the test problem with 40 locations with a solution search threshold of 20 seconds, the outputted solution would have a 655.26% higher cost than the optimal plan (a similar behaviour is observed in the tests performed in the Evaluation Chapter in multiple planners). Therefore, the number of unnecessary static objects that are parameters in executable actions can affect the quality of the solutions found by Itsat when a complete solution search is not possible.

5.6 Defining the Difficulty of a Planning Problem

Section Overview In this section, we formally define the difficulty metrics obtained from the analysis of the test results and discuss their applicability.

Definition 5.6. $N(\alpha)$ represents the *total number of agents* α in Π and $N(\neg\alpha)$ represents the *total number of inactive dynamic objects* $\neg\alpha$ in Π. $N(\alpha)$ and $N(\neg\alpha)$ are a potential difficulty metrics of planning problem Π.

The results show that the total number of agents $N(\alpha)$ affects the difficulty of a planning problem. Therefore, $N(\alpha)$ is a potential difficulty metric for planning problems with a similar structure to Driverlog when solved with the sample planners used in our test or with other similar planners. The performed tests show that this metric is particularly useful for problems where the parent agents are present in goal state facts. In such problems, our tests have shown that a reduced number of parent agents can drastically decrease the difficulty of a problem. This metric can also be useful even if there are no parent agents present in the goal state facts, particularly in situations where more subtle difficulty variations are relevant.

The results of the test problems also show that the total number of inactive dynamic objects $N(\neg\alpha)$ affects the difficulty of a planning problem. Therefore, $N(\neg\alpha)$ is a potential difficulty metric for planning problems with a similar structure to Driverlog when solved with the sample planners used in our test or with other similar planners. The performed tests show that this metric is useful in situations where more subtle difficulty variations are relevant.

The $N(\alpha)$ and $N(\neg\alpha)$ metrics cumulatively cover all dynamic objects in a planning problem. However, considering that our empirical results (Sections 5.3 and 5.4) show that the number of α in Π has a higher influence on the solving difficulty than the number of $\neg\alpha$ in Π, we defined a separate metric for each.

Definition 5.7. $N(\Phi)$ represents the *total number of necessary static objects* $N(\Phi)$ in Π. $N(\Phi)$ is a potential difficulty metric of planning problem Π.

The results of the test problems also show that the total number of unnecessary static objects $N(\neg\Phi)$ affects the difficulty of a planning problem. Therefore, $N(\neg\Phi)$ is a potential difficulty metric for planning problems with a similar structure to Driverlog when solved with the sample planners used in our test or with other similar planners. The performed tests show that this metric is useful in situations where more subtle difficulty variations are relevant.

Definition 5.8. $N(\neg\Phi)$ represents the *total number of unnecessary static objects* $N(\neg\Phi)$ in Π which have their type as a parameter it at least one precondition a_{pre} or in at least one effect a_{eff} of at least one executable action in Π. $N(\neg\Phi)$ is a potential difficulty metric of planning problem Π.

While no tests explicitly show that the total number of necessary static objects $N(\Phi)$ impacts the difficulty of a planning problem, we can presume from the static objects test problems that $N(\Phi)$ can also potentially affect the difficulty of a planning problem and infer that $N(\Phi)$ is a potential difficulty metric for planning problems with a similar structure to Driverlog when solved with the sample planners used in our test, or with other similar planners.

It is important to note that definitions 5.7 and 5.8 do not cumulatively cover all possible static objects in a planning problem, as a problem can have unnecessary static objects that are never part of operator parameters. Such objects do not influence in any way the solving difficulty of the problem and have been discarded from the $N(\neg\Phi)$ metric in order to preserve the accuracy of the metric. To illustrate this, let us consider a scenario in which the Driverlog running examples (described in Chapter 2 Section 2.1.5.1) have been modified to also contain unnecessary static objects that will never become parameters in any of the operators in Driverlog. In this scenario, the Driverlog running examples also contain the *palm_tree* type along with a very large number of objects with the *palm_tree* type. However, regardless of the number of added *palm_tree* objects, the solving difficulty will always be the same as in the unmodified running examples, as there are no operators in Driverlog that have parameters with the *palm_tree* type. Therefore, we constrained the $N(\neg\Phi)$ metric to only take into account unnecessary static objects that can become operator parameters in the considered planning problem, as objects that can not become operator parameters (such as the Driverlog palm trees) can not influence the solving process.

Definition 5.9. $N(t_i)$ represents the *total number of entangled types*. $N(t_i)$ is a potential difficulty metric of planning problem Π.

The results of the test problems also show that the total number of entangled types $N(t_i)$ affects the difficulty of a planning problem. Therefore, $N(t_i)$ is a potential difficulty metric for planning problems with a similar structure to Driverlog when solved with the sample planners used in our test or with other similar planners. The performed tests show that this metric is particularly useful for problems where we have multiple entangled parent types. In such problems, our tests have shown that a reduced number of parent agents can drastically decrease the difficulty of a problem.

It is important to note that the above difficulty metrics may not apply to all possible temporal planners when solving the Driverlog planning problem. It is also possible that the difficulty metrics may not apply to the sample planners used in our tests when solving a problem with a different structure to Driverlog. However, our inductive approach to empirically identify some of the difficulty elements in a specific planning problem with a common structure when solved by a specific range of state-of-the-art temporal planners that have performed well in the past IPCs yields difficulty metrics that are likely very relevant to other similarly structured problems when solved with the same planners used for identifying the metrics or with other similar planners. The metrics will serve as an optimisation guide for constructing the recursive agents and landmarks decomposition procedure that, as we will show in the Evaluation chapter, increases the scale and improves the solution quality of the solvable problems.

5.7 Summary of Chapter Contributions

In this chapter, we presented a framework for evaluating the difficulty of a planning problem.

Specifically, we introduced and formally defined the concept of entanglement among types and objects.

We utilised a planning problem with a structure similar to many real-world problems (Driverlog) to construct a series of tests in order to evaluate the impact of the number of objects, the number of types and the entanglements among them on the difficulty of a planning problem.

Furthermore, we performed an analysis of the results which showed that the number of agents, inactive dynamic objects, necessary static objects, unnecessary static objects and entangled types all affect the difficulty of a planning problem to various degrees and can be used as metrics for estimating the difficulty of a planning problem.

Chapter 6

Recursive Agents and Landmarks Strategic-Tactical Planning

Chapter Overview The difficulty metrics defined in Chapter 5 are used as a guide for creating a recursive decomposition procedure for temporally expressive numeric planning problems based on the agents, the agent dependency relationships, the agent classifications, the landmarks and the relaxed landmarks defined in Chapter 4.

6.1 High-level Description

Section Overview In this section, we will provide a high-level description (Figure 6.1) of the whole procedure for recursive agents and landmarks strategic-tactical planning (RALSTP) as an intuitive introduction to the detailed and formal description (Figure 6.26) provided in the rest of the chapter.

Our recursive agent-based temporal decompositions procedure for solving a large-scale temporal numeric planning problem Π consists of trying to reduce the total number of inactive dynamic objects $N(\neg\alpha)$, the total number of dead-end agents $N(d\alpha)$, the total number of parent agents $N(p\alpha)$ and the total number of entangled types $N(t_i)$ difficulty metrics of a planning problem as much as possible, with as few such decompositions as possible, by recursively increasing the decompositions until we either find a solution or encounter an error. The solution is formed by recombining the decompositions taking into account the makespan of temporal plans and durations of durative actions (detailed in Section 6.4.3.2) for mitigating any potential constraint violations among the decompositions while allowing temporal expressiveness and numeric components.

The procedure begins by considering the unmodified original planning problem Π as the instance planning problem of the base case instance (described in Section 6.2). As the dependency relationships allow us to create a partial order between dead-end agent goals and parent agent goals, we first try to achieve the dead-end agent goals of the instance planning problem Π without considering the parent agent goals in Π and use the state where all the dead-end agent goals are solved as the initial state from which we start searching for a plan which achieves all parent agent goals in Π.

The separation of the dead-end agent goals from the parent agent goals not only reduces the difficulty of achieving the dead-end agent goals by decreasing or eliminating the parent agent goals from the search operation of the dead-end agent goals but also reduces the difficulty of achieving parent agent goals by potentially having a lower total number of entangled types $N(t_i)$ in the parent agent goals problem due to no longer having to consider the dead-end type of the solved dead-end agent goals.

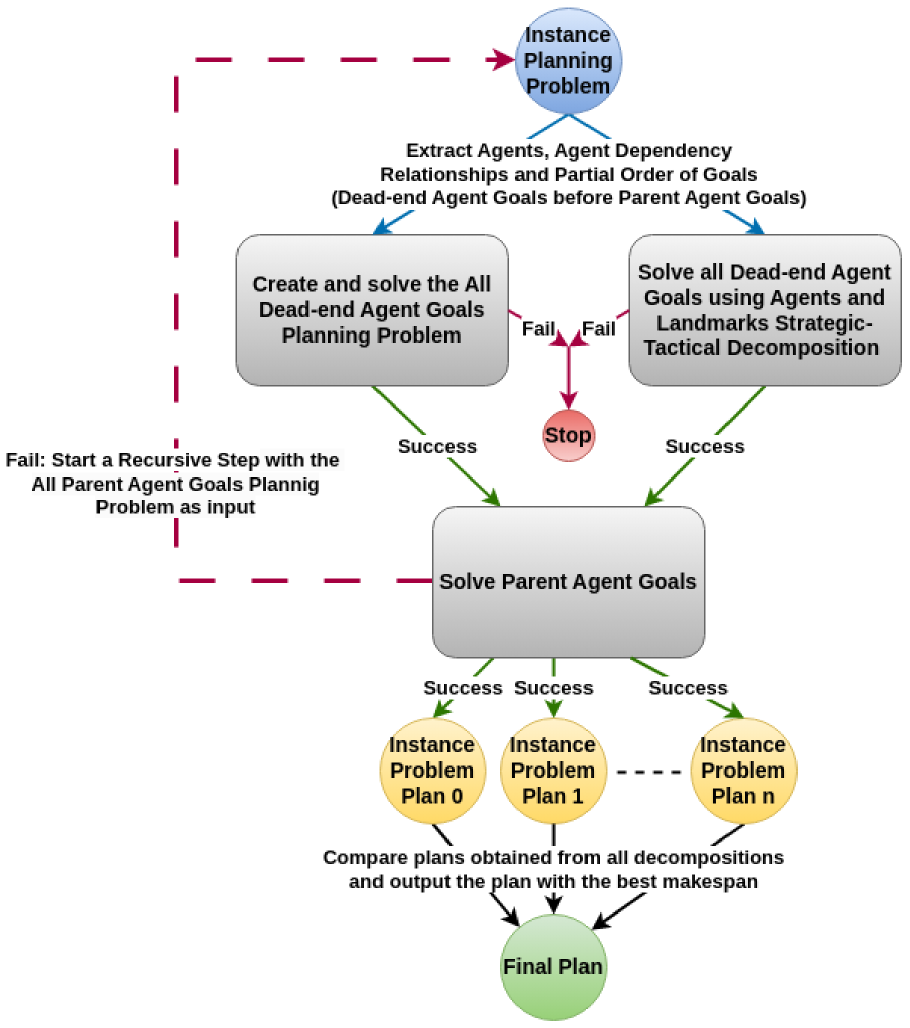

Instance Planning Problem
Extract Agents, Agent Dependency Relationships and Partial Order of Goals (Dead-end Agent Goals before Parent Agent Goals)
Create and solve the All Dead-end Agent Goals Planning Problem
Solve all Dead-end Agent Goals using Agents and Landmarks Strategic-Tactical Decomposition
Fail
Fail
Stop
Success
Success
Fail: Start a Recursive Step with the All Parent Agent Goals Plannig Problem as input
Solve Parent Agent Goals
Success
Success
Success
Instance Problem Plan 0
Instance Problem Plan 1
Instance Problem Plan n
Compare plans obtained from all decompositions and output the plan with the best makespan
Final Plan

Fig. 6.1 High-level Flowchart for Solving a Planning Problem using Recursive Agents and Landmarks Strategic-Tactical Planning

However, even though decompositions can have a significant impact in decreasing the difficulty of a planning problem, our procedure aims to efficiently solve a planning problem with as few decompositions as possible since every decomposition increases the chance of reducing the quality of the solution. For example, splitting the dead-end agent goals into multiple sub-problems and solving them individually can potentially decrease the solution quality as a planner will not be able to consider information that is not present in the sub-problem it tries to solve. Therefore, to try to minimise the potential loss in solution quality due to decompositions, we solve all dead-end agent goals with two types of decompositions. First, we attempt to solve all dead-end agent goals in Π in a single problem that does not have any parent agent goals (described in Section 6.3). The all dead-end agent goals problem is solved as a stand-alone temporal planning problem using an off-the-shelf numeric temporally expressive planner, so the problem can involve temporal expressiveness and numeric components. Then, we also split all dead-end agent goals in Π among multiple sub-problems using a landmarks and agents strategic-tactical dead-end agent goals decomposition (described in Section 6.4). The strategic-tactical decompositions can also involve temporal expressiveness and numeric components (detailed in Sections 6.4.2.6 and 6.4.3).

After achieving all dead-end agent goals $g_{d\alpha_t}$ in Π, either by solving them in a single problem or by solving them in multiple sub-problems using landmarks [38] and agents decompositions along with strategic-tactical planning [15], we use each state where all $g_{d\alpha_t}$ have been achieved (from the obtained distinct decompositions) as the initial state of a new planning problem Π' with the goal state G in Π' containing only the parent agent goals of the instance planning problem Π (described in Section 6.5). Each Π' corresponding to each decomposition will become less difficult than Π as all dead-end agents from Π can be eliminated from each Π' (therefore potentially reducing the total number of entangled types $N(t_i)$ in Π') due to the partial order obtained from the dependency relationships of Π guaranteeing that the dead-end agents of Π are not required for achieving the parent agent goals of Π. Π' is solved as a stand-alone numeric temporal planning problem using an off-the-shelf temporally expressive numeric planner. Therefore, Π' can involve temporal expressiveness and numeric components.

If we find a solution for Π', we merge the corresponding all dead-end agent goals plan of Π with the plan for Π' by increasing the start time of each durative action in the plan for Π' with the makespan of the corresponding all dead-end agent goals plan of Π and output a plan that solves all goals in Π for each decomposition that was able to output a dead-end agent goals plan and a parent agent goals plan (described in Section 6.5.1).

We then compare all plans that solve all goals in Π obtained from all decompositions and output as the final plan the plan with the best makespan (described in Section 6.6.

In case we can not find a solution for Π', we start a recursive step (detailed in Section 6.5.2) with Π' as the instance planning problem for the recursive step. Π' does not have the same dependency relationships as Π because Π' does not have any of the dead-end agents or dead-end agent goals of Π. Therefore, we extract the dependency relationships specific to Π' and use the new dependency relationships to create the dead-end agent goals decompositions and corresponding parent agent goals problems specific to Π' in order to attempt to solve the newly obtained problems identically to how we solved the previous dead-end agent goals decompositions and corresponding parent agent goals problems derived from Π. The capacity for temporal expressiveness and numeric components is maintained in each recursive step in every individual dead-end agent and parent agent decomposition (detailed in Section 6.5.2). The recursive steps continue until we find a solution by concatenating all plans obtained during the procedure. A recursive step will terminate if all dead-end agent goals in the respective recursive step are not solvable with any of the decompositions.

Example 6.1. In this example, we will consider $\Pi_{(drivers,trucks,packages)}$, a planning problem built using the IPC 2014 Driverlog domain which contains multiple driver, truck and package *(at agent location)* goals in its goal state. The dependency relationships of a Driverlog problem which has at least one driver, one truck and one package goal (such as $\Pi_{(drivers,trucks,packages)}$) are:

$$driver \longmapsto truck$$
$$driver \longmapsto package$$
$$truck \longmapsto package$$

The dependency relationships dictate that the package goals will be dead-end agent goals in $\Pi_{(drivers,trucks,packages)}$ while the driver and truck goals will be parent agent goals in $\Pi_{(drivers,trucks,packages)}$. The dependency relationships dictate a partial order among dead-end agent and parent agent goals, which in $\Pi_{(drivers,trucks,packages)}$ translates to all package goals must be achieved before starting the search for truck or driver goals.

Therefore, when applying our recursive decomposition procedure, we first attempt to solve the package goals in a single $\Pi_{(packages)}$ planning problem that contains all package goals from $\Pi_{(drivers,trucks,packages)}$ and no truck or driver goals from $\Pi_{(drivers,trucks,packages)}$. Then, we attempt to solve all package goals in $\Pi_{(drivers,trucks,packages)}$ by splitting them among multiple $\Pi_{(packages)}$ sub-problems using a landmarks and agents decomposition along with strategic-tactical planning [15]. Afterwards, for each successfully obtained $plan_{packages}$ that when executed on the initial state of $\Pi_{(drivers,trucks,packages)}$ outputs a state that has all *package* goals solved, we create a new planning problem $\Pi_{(drivers,trucks)}$ that has as its initial state a state where all package goals have been achieved and in the goal state all the driver and truck goals from $\Pi_{(drivers,trucks,packages)}$. We then remove all package agents from $\Pi_{(drivers,trucks)}$ as the dependency relationships of $\Pi_{(drivers,trucks,packages)}$ guarantee that package agents are not required for solving driver or truck goals. Afterwards, we attempt to solve each obtained $\Pi_{(drivers,trucks)}$ problem, and, if successful, we merge the corresponding $plan_{packages}$ with the plan for $\Pi_{(drivers,trucks)}$ by increasing the start time of each durative action in the plan for $\Pi_{(drivers,trucks)}$ with the makespan of the corresponding $plan_{packages}$ in order to obtain a $plan_{packages-drivers-trucks}$ plan that solves all goals in $\Pi_{(drivers,trucks,packages)}$ for each successfully obtained $plan_{packages}$ that had its corresponding $\Pi_{(drivers,trucks)}$ problem solved. We then output as the final plan for $\Pi_{(drivers,trucks,packages)}$ the $plan_{packages-drivers-trucks}$ that has the best makespan among all the $plan_{packages-drivers-trucks}$ plans obtained from all decompositions.

However, if we can not find a solution for a $\Pi_{(drivers,trucks)}$ problem corresponding to a state that has all *package* goals solved, we start a recursive step with $\Pi_{(drivers,trucks)}$ as the instance planning problem for the recursive step (and repeat the same procedure previously applied to $\Pi_{(drivers,trucks,packages)}$). The only dependency relationship of a Driverlog problem which has at least one driver, one truck goal and no package goals is:

$$driver \longmapsto truck$$

The other dependency relationships present in $\Pi_{(drivers,trucks,packages)}$ no longer exist in $\Pi_{(drivers,trucks)}$ as all package agents have been eliminated from $\Pi_{(drivers,trucks)}$. Therefore, the dependency relationships in $\Pi_{(drivers,trucks)}$ dictate that the truck goals will be the dead-end agent goals in $\Pi_{(drivers,trucks)}$ while the driver goals will be the parent agent goals in $\Pi_{(drivers,trucks)}$ and that the partial order between the goals in $\Pi_{(drivers,trucks)}$ translates to all truck goals must be achieved before starting the search for the driver goals.

Therefore, we first attempt to solve the truck goals in $\Pi_{(drivers,trucks)}$ (which are the new dead-end agent goals) in a single $\Pi_{(trucks)}$ planning problem that contains all truck goals from $\Pi_{(drivers,trucks)}$ and none of the driver goals in $\Pi_{(drivers,trucks)}$ and then attempt to solve all truck goals in $\Pi_{(drivers,trucks)}$ by splitting them among multiple $\Pi_{(trucks)}$ sub-problems using a landmarks and agents decomposition along with strategic-tactical planning [15]. Afterwards, for each successfully obtained $plan_{trucks}$ that when executed on the initial state of $\Pi_{(drivers,trucks)}$ outputs a state that has all *truck* goals solved, we create a new planning problem $\Pi_{(drivers)}$ that has as its initial state a state where all truck goals have been achieved and in the goal state all the driver goals from $\Pi_{(drivers,trucks)}$. We then remove all truck agents from $\Pi_{(drivers)}$ as the dependency relationships of $\Pi_{(drivers,trucks)}$ guarantee that truck agents are not required for solving driver goals. Afterwards, we attempt to solve each obtained $\Pi_{(drivers)}$ problem, and, if successful, we merge the corresponding

$plan_{trucks}$ plan with the plan for $\Pi_{(drivers)}$ by increasing the start time of each durative action in the plan for $\Pi_{(drivers)}$ with the makespan of the corresponding $plan_{trucks}$ in order to obtain a $plan_{drivers-trucks}$ plan that solves all goals in $\Pi_{(drivers,trucks)}$ for each successfully obtained $plan_{trucks}$ that had its corresponding $\Pi_{(drivers)}$ problem solved. The $plan_{drivers-trucks}$ with the best makespan among all obtained $plan_{drivers-trucks}$ plans is merged with the corresponding $plan_{packages}$ achieved in the previous recursive step by increasing the start time of each durative action the plan for $\Pi_{(drivers,trucks)}$ with the makespan of the $plan_{packages}$ in order to obtain a $plan_{packages-drivers-trucks}$ that solves all goals in $\Pi_{(drivers,trucks,packages)}$ for each successfully obtained $plan_{packages}$ that had its corresponding $\Pi_{(drivers,trucks)}$ problem solved in the current recursive step. We then output as the final plan for $\Pi_{(drivers,trucks,packages)}$ the $plan_{packages-drivers-trucks}$ that has the best makespan among all the $plan_{packages-drivers-trucks}$ plans obtained from all decompositions.

6.2 Starting a RALSTP Instance

Section Overview In this section, we describe how RALSTP instances are started.

Definition 6.2. Π represents the unmodified original temporally expressive numeric planning problem we want to solve.

Definition 6.3. An *instance planning problem* Π_i represents a planning problem provided as the input of a RALSTP instance. An instance planning problem can be the unmodified original planning problem Π we want to solve or a sub-problem derived from Π.

Definition 6.4. A *decomposition arborescence* $T(\Pi) = (N, E)$ represents the arborescence obtained from applying our recursive agent and landmarks decomposition procedure to a planning problem Π intended for solving. The root node $r \in N$ maps Π. A child node $cn \in N$ maps the instance planning problem Π_i at the start of a recursive step (detailed in Section 6.5.2). A directed edge $e \in E$ connects a parent node $pn \in N$ (tail of e) with a child of pn (head of e). A directed edge e maps the plan that achieves all the dead-end agent goals of the instance problem Π_i mapped by the parent node pn at the tail of e.

Having presented a high-level description of the decomposition procedure, our thesis continues with a detailed description of all components and processes. The procedure begins by creating a decomposition arborescence $T(\Pi)$ for the unmodified original planning problem Π we want to solve. Π is mapped to the root of $T(\Pi)$

To start a RALSTP instance, we must provide an instance planning problem Π_i as input. Only a single RALSTP instance can run at any given time. If the instance we are starting is the first initialised instance, the instance planning problem Π_i of the RALSTP instance is the unmodified original planning problem Π we are trying to solve. Otherwise, the instance planning problem Π_i of a successive RALSTP instance (started by a recursive step, detailed in Section 6.5.2) will be a sub-problem derived from Π. Therefore, the solving process of Π begins by starting a RALSTP instance with Π as the instance planning problem Π_i.

A RALSTP instance begins by cleaning its instance planning problem Π_i with the procedure in Algorithm 5. Then, we apply the steps in Algorithm 6 to extract the agent dependency relationships of Π_i and the steps in Algorithm 8 to obtain the dependency status of all agent types, the agents, the agent facts and the agent goals in Π_i. If we are unable to classify the agents according to their agent dependency status (described in Section 4.1.6.1 from Chapter 4) or there are no dead-end agent goals $g_{d\alpha_t}$ in Π_i or there are goals in Π_i that contain a dead-end agent as well as a parent agent or the reunion of the set of all dead-end agent goals $g_{d\alpha_t}$ in Π_i with the set of all parent agent goals $g_{p\alpha_t}$ in Π_i is not equal to G in Π_i, then Π_i is unsolvable with RALSTP and the instance is stopped. This is done to prevent the unnecessary algorithmic overhead (detailed in the evaluation from Chapter 7) that would come with the execution of the entire RALSTP procedure on an incompatible problem. Only if the agent dependency status classification is successful and the two compared goal sets are equal do we allow the instance to continue executing. If allowed to continue, the instance will proceed with the decomposition of Π_i into sub-problems.

6.3 Creating and Solving the All Dead-end Agent Goals Planning Problem

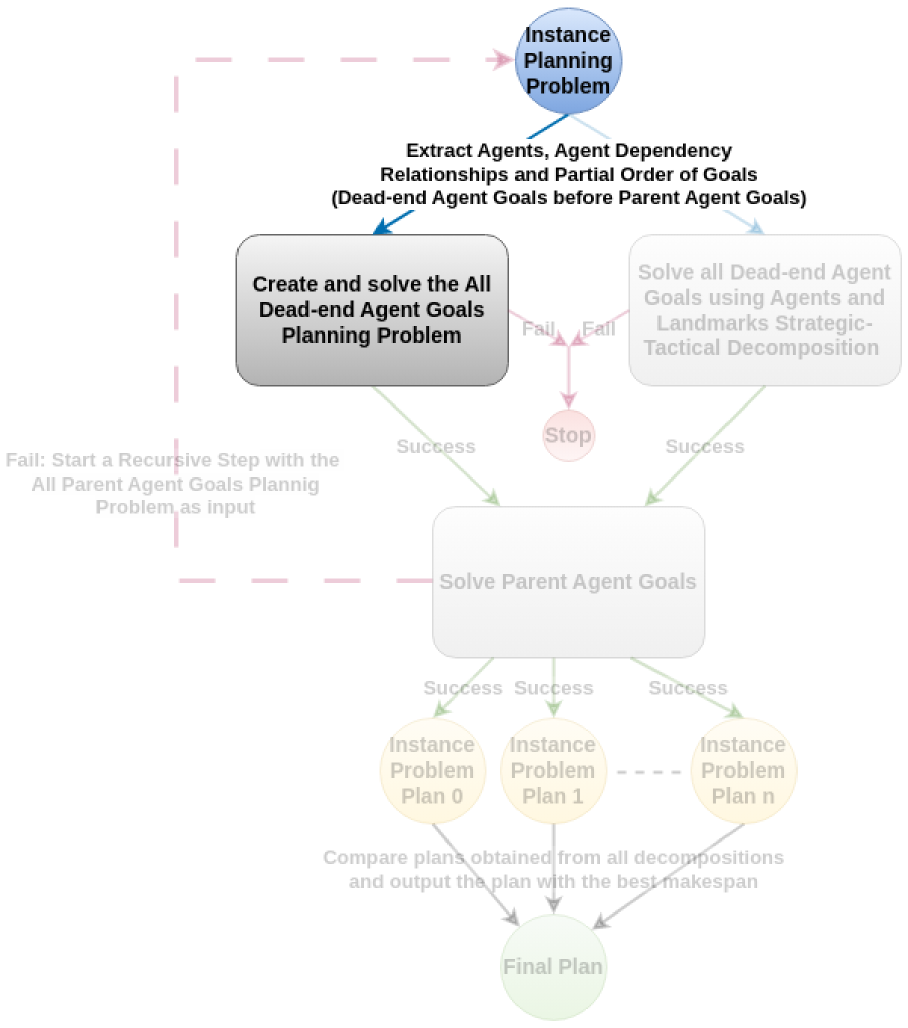

Instance Planning Problem
Extract Agents, Agent Dependency Relationships and Partial Order of Goals (Dead-end Agent Goals before Parent Agent Goals)
Create and solve the All Dead-end Agent Goals Planning Problem
Solve all Dead-end Agent Goals using Agents and Landmarks Strategic-Tactical Decomposition
Fail
Fail
Stop
Success
Success
Fail: Start a Recursive Step with the All Parent Agent Goals Plannig Problem as input
Solve Parent Agent Goals
Success
Success
Success
Instance Problem Plan 0
Instance Problem Plan 1
Instance Problem Plan n
Compare plans obtained from all decompositions and output the plan with the best makespan
Final Plan

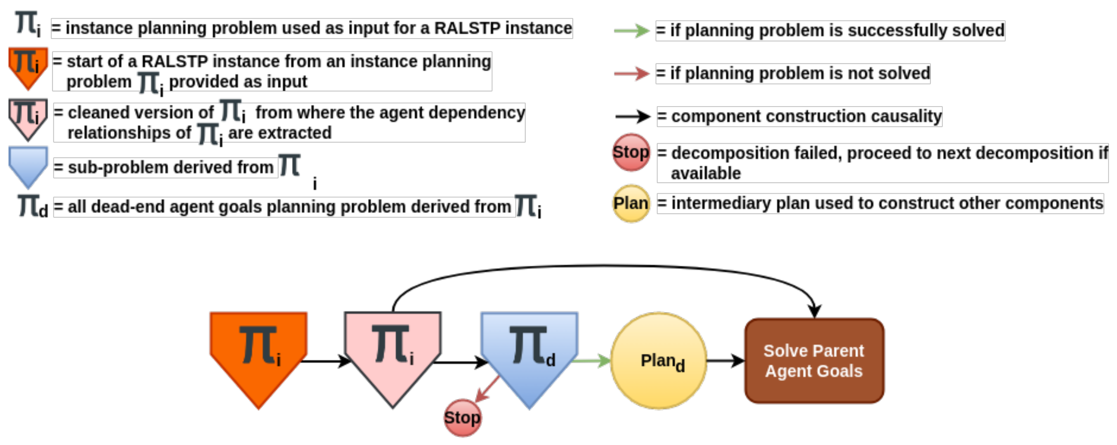
Fig. 6.2 Location of Section 6.3 on the High-Level Flow Chart

Section Overview In this section, we present the procedure for creating and solving the all dead-end agent goals planning problem derived from the agent dependency relationships and classifications of an instance planning problem Π_i.

Definition 6.5. An *all dead-end agent goals planning problem* is defined as a tuple $\Pi_d := \{P,V,A,I,G\}$ where $\{P,V,A,I\}$ in Π_d are inherited from the instance planning problem Π_i from which Π_d is derived and G in Π_d contains only all the dead-end agent goals g in Π_i and none of the parent agent goals from Π_i.

Definition 6.6. An *all dead-end agent goals plan* $plan_d$ is a plan that achieves all the dead-end agent goals of an instance planning problem Π_i when executing the plan from the initial state of Π_i.

After obtaining the agent dependency relationships and classifications specific to instance planning problem Π_i (defined in Section 6.2), we focus on achieving the dead-end agent goals of Π_i without considering the parent agent goals of Π_i. The dependency relationships and agent dependency status are used to create a partial order between the goals of Π_i which dictates that all dead-end agent goals in Π_i must be achieved before the parent agent goals in Π_i as at least one parent agent from each parent type in Π_i is required for achieving dead-end agent goals in Π_i but no dead-end agents in Π_i are required for achieving any of the parent agent goals in Π_i.



Πi = instance planning problem used as input for a RALSTP instance
Πi = start of a RALSTP instance from an instance planning problem Πi provided as input
Πi = cleaned version of Πi from where the agent dependency relationships of Πi are extracted
= sub-problem derived from Πi
Πd = all dead-end agent goals planning problem derived from Πi
= if planning problem is successfully solved
= if planning problem is not solved
= component construction causality
Stop = decomposition failed, proceed to next decomposition if available
Plan = intermediary plan used to construct other components
Πi
Πi
Πd
Stop
Plan_d
Solve Parent Agent Goals

Fig. 6.3 Flowchart of the Creation And Solving Attempt of the All Dead-end Agent Goals Planning Problem

The procedure for creating and solving the all dead-end agent goals planning problem Π_d is described in Algorithm 29, which has a time complexity based on the chosen solver (the creation of Π_d is computed in polynomial time). The procedure starts by creating an all dead-end agent goals planning problem Π_d from the instance planning problem Π_i with

Π_d identical in every aspect to Π_i except for the goal state G in Π_d which is populated only by all the dead-end agent goals in Π_i without any of the parent agent goals of Π_i (line 1). Then, we attempt to solve Π_d (line 5). If successful (Figure 6.3), we proceed to the parent agent goals-solving procedure (described in section 6.5) to which we will provide $plan_d$ as input (line 8). Π_d is solved as a stand-alone numeric temporal planning problem using an off-the-shelf temporally expressive numeric planner. Therefore, Π_d can involve temporal expressiveness and numeric components.

Algorithm 14 Algorithm for creating and solving the All Dead-end Agent Goals Planning Problem Π_d

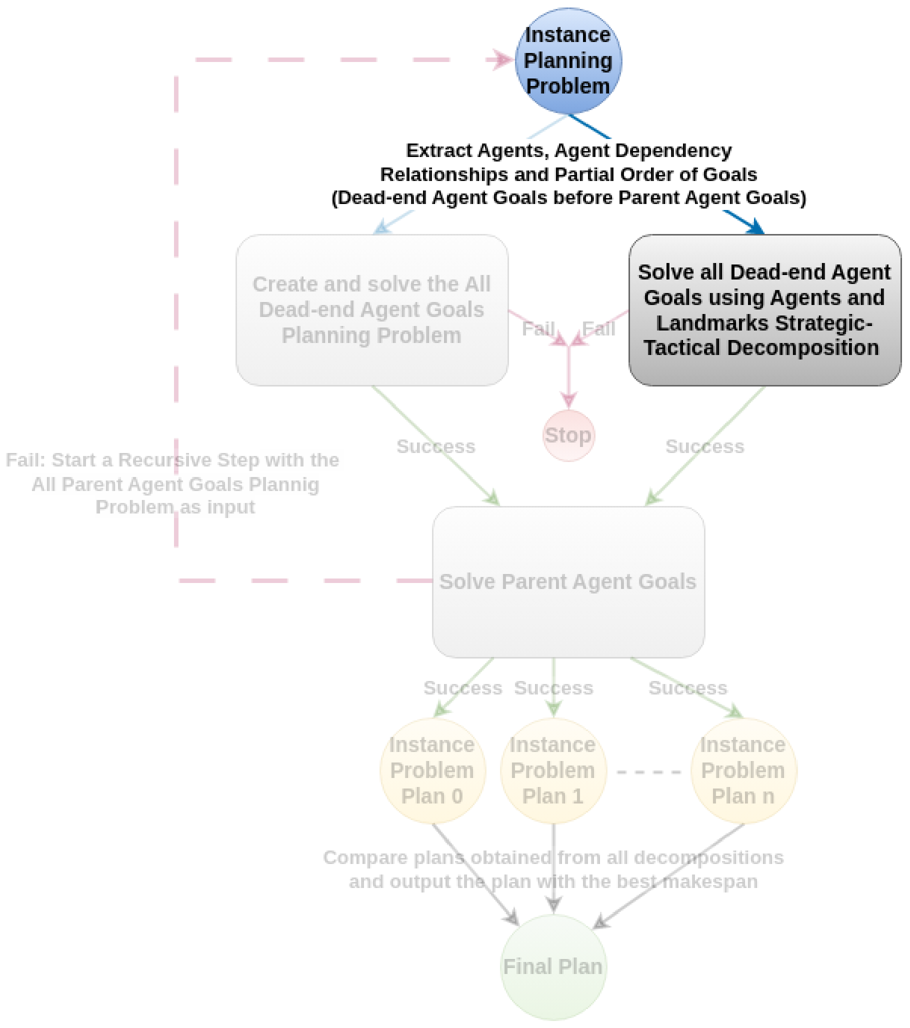
```
Input: $\Pi_i$
Output: $plan_d$ and final state of solved($\Pi_d$) or failure
1: create the all dead-end agent goals planning problem $\Pi_d := \{P,V,A,I,G\}$ where $\{P,V,A,I\}$ are the same as in $\Pi_i$ and $G$ is empty
2: for all dead-end agent goals $g_{d\alpha_t}$ in $\Pi_i$ do
3:     add $g_{d\alpha_t}$ to $G$
4: end for
5: solved($\Pi_d$) = attempt to solve $\Pi_d$
6: if solved($\Pi_d$) == true then
7:     all dead-end agent goals plan $plan_d$ = plan of solved($\Pi_d$)
8:     return $plan_d$)
9: else
10:    return failure
11: end if
```

Example 6.7. In the DLOG-5-5-10 Driverlog planning problem, we have ten dead-end agents with the *package* dead-end type: *package1, package2, package3, package4, package5, package6, package7, package8, package9, package10*. DLOG-5-5-10 has an *(at package location)* dead-end agent goal for each of the *package* agents except *package1, package3* and the goal for *package7* is achieved in the initial state.

When considering the DLOG-5-5-10 Driverlog planning problem as the instance planning problem Π_i, the cleaned version of DLOG-5-5-10 no longer has the *package1, package3, package7* agents, agent facts and agent goals, as *package1, package3, package7* are inactive dynamic objects (as described in Example 4.16).

The all dead-end agent goals planning problem Π_d derived from the cleaned version of DLOG-5-5-10 is identical to DLOG-5-5-10 except it has none of the driver parent agent goals of DLOG-5-5-10 in the goal state G in Π_d, none of the truck parent agent goals of DLOG-5-5-10 in the goal state G in Π_d and only all the package dead-end agent goals of DLOG-5-5-10 in the goal state G in Π_d. If the solving attempt of Π_d outputs an all dead-end agent goals plan $plan_d$, we then use $plan_d$ as input for the parent agent goals solving procedure in Algorithm 29 (detailed in Section 6.5).

6.4 Agents and Landmarks Strategic-Tactical Decomposition and Abstraction



Instance Planning Problem
Extract Agents, Agent Dependency Relationships and Partial Order of Goals (Dead-end Agent Goals before Parent Agent Goals)
Create and solve the All Dead-end Agent Goals Planning Problem
Solve all Dead-end Agent Goals using Agents and Landmarks Strategic-Tactical Decomposition
Fail
Fail
Stop
Success
Success
Fail: Start a Recursive Step with the All Parent Agent Goals Plannig Problem as input
Solve Parent Agent Goals
Success
Success
Success
Instance Problem Plan 0
Instance Problem Plan 1
Instance Problem Plan n
Compare plans obtained from all decompositions and output the plan with the best makespan
Final Plan

Fig. 6.4 Location of Section 6.4 on the High-Level Flow Chart

Section Overview In this section, we will show the procedure for using agents, landmarks and strategic-tactical planning to decompose and solve the dead-end agent goals of a planning problem. We will first provide a higher-level description of the technique, which will be followed by a more detailed description of each component used in the technique.

Regardless of the solving success of the all dead-end agent goals planning problem Π_d and of the success of the parent agent goals solving procedure when $plan_d$ for Π_d is the input (described in section 6.5), a RALSTP instance will also attempt to solve the dead-end agent goals using our modified version of strategic tactical planning [15]. This is done not only to potentially reduce the difficulty of achieving all the dead-end agent goals in case Π_d is too difficult to solve, but also to potentially obtain better solutions to the instance planning problem than the solution obtained by using the outputs of a successfully solved Π_d.

STP can increase the scale of solvable planning problems by dividing the difficulty of Π_i among multiple sub-problems. The STP technique described in Buksz et al. (2018) has been modified from a manual domain-engineer-dependent and time-consuming procedure into a fast automatic process that requires no human input. STP uses as inputs the same elements used for creating the all dead-end agent goals planning problem Π_d: the cleaned version of planning problem Π_i, the agent dependency relationships of Π_i, the dependency status of all agent types as well as the agents, the agent facts and the agent goals in Π_i. The technique decomposes the dead-end agent goals into multiple smaller sub-problems (described in Section 6.4.2) that will have a potentially lower difficulty than the all dead-end agent goals planning problem due to having a smaller total number of dead-end agents $N(d\alpha)$ and due to having only the minimum total number of parent agents $N(p\alpha)$ necessary for solving a dead-end agent goal. The dead-end agent goals are arranged using a two-level hierarchical structure in which the goals are first grouped into dead-end agent goal sets at the lower level and then the dead-end agent goal sets are further arranged into one or more contextual decompositions at the higher level (described in Section 6.4.1). The contextual grouping is based on possible favourable agent configurations (described in Section 6.4.1.2) as well as on the output of the landmarks contextual sorting procedure (described in Section 6.4.1.1) which attempts to replicate to how humans catalogue the specifics of a problem in distinct ways according to different contextual perspectives. A tactical planning problem (described in Section 6.4.2) will be created for each dead-end agent goal set in a contextual decomposition. A strategic problem (described in Section 6.4.3) will be created for each contextual decomposition that had all its corresponding tactical planning problems successfully solved in order to mitigate any eventual constraint violations among the tactical plans in the respective context. The final state of each successfully solved strategic problem will contain as facts all the dead-end agent goals of the instance planning problem and will become the initial state of the parent agent goals planning problem.

6.4.1 Decomposing Dead-end Agent Goals into Dead-end Agent Goal Sets and Contextual Decompositions

Section Overview The dead-end agent goals of a planning problem are grouped into dead-end agent goal sets based on the duplicate landmarks found between the dead-end agent goals (described in Section 6.4.1.1) and based on the maximum quantity of unique parent agent groups with no common agents among them (described in Section 6.4.1.2). The dead-end agent goal sets will be used to form contextual decompositions.

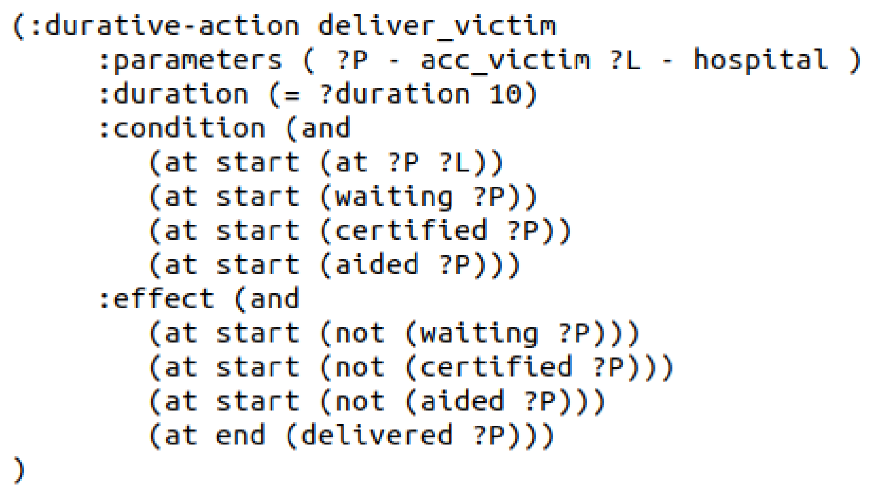
6.4.1.1 Decomposing Dead-end Agent Goals Using the Landmarks Contextual Sorting Procedure

Section Overview In this section, we describe the dead-end agent goals decomposition based on the duplicate landmarks found between the dead-end agent goals.

The agents and landmarks strategic tactical decomposition procedure starts with extracting the landmarks from Π_i using the steps described in Section 4.3 from Chapter 4. The landmark-based decomposition procedure is designed to replicate human contextual understanding by evaluating the similarities among the dead-end agent goals from the number of landmarks and relaxed landmarks that are found in common between dead-end agent goals. The more landmarks are found in common between two or more goals (such as the common location in the RTAM Introduction example) the bigger the possibility that there is a contextual connection between the goals.

In effect, the landmark-based decomposition method groups dead-end agent goals into similarity goal sets based on the landmarks that are found in common between the L_g sets obtained after the landmarks extraction procedure. The intention is to potentially obtain contextual decompositions with the highest makespan among all tactically solved similarity goal sets lower than the highest makespan among the tactically solved random dead-end agent goal sets in the maximum quantity of unique parent agent groups contextual decomposition.

Example 6.8. In the *RTAM_5_2_35* planning problem, we have the set Ξ = { *acc_victim2, acc_victim3, acc_victim4, acc_victim11, acc_victim22, acc_victim28* } representing dead-end agents located at the *accident_location0* location in the initial state. We also have a *(delivered ?acc_victim)* dead-end agent goal in the goal state for each of the agents in Ξ. In order to achieve the dead-end agent goals of the agents in Ξ we need to execute the *deliver_victim* action (Figure 6.5) for each agent in Ξ. Therefore, we need the condition *(aided ?acc_victim)* to be true for each agent in Ξ. If we use a unique parent agent group as the only agents for achieving the goals, an *ambulance* agent would have to travel via the *move* action (Figure 4.1) to a single location (*accident_location0*) in order to execute the *first_aid* action (Figure 4.2) responsible for making the *(aided ?acc_victim)* conditions true for all agents in set Ξ.



```
(:durative-action deliver_victim
     :parameters ( ?P - acc_victim ?L - hospital )
     :duration (= ?duration 10)
     :condition (and
        (at start (at ?P ?L))
        (at start (waiting ?P))
        (at start (certified ?P))
        (at start (aided ?P)))
     :effect (and
        (at start (not (waiting ?P)))
        (at start (not (certified ?P)))
        (at start (not (aided ?P)))
        (at end (delivered ?P)))
)
```

Fig. 6.5 RTAM domain *deliver_victim* action

If, however, we create a set of *acc_victim* agents which are at different locations in the initial state, a single *ambulance* agent would have to travel to multiple locations to start the *first_aid* actions and the extra *move* actions necessary for travelling to multiple locations could potentially increase the makespan for achieving the respective *(delivered ?acc_victim)* dead-end agent goals. Therefore, being able to group the dead-end agent goals into goal sets based on the location of the dead-end agents in the initial state could potentially result in more efficient makespans for solving the location-based goal sets in comparison to solving randomly obtained goal sets.

Identifying and Scoring Similarity Goal Sets

Definition 6.9. A *dead-end agent goal set* $G_d \subseteq G$ is a set that can contain only dead-end agent goals.

Definition 6.10. A *similarity goal set* $Sim_d \subseteq G$ is a dead-end agent goal set G_d formed of all dead-end agent goals $g_{d\alpha_t} \in G$ that have at least one identical landmark or relaxed landmark l in their respective backchaining L_g sets when the time points of the landmarks and relaxed landmarks are ignored.

Definition 6.11. $score(l)$ represents the score of a landmark or relaxed landmark l and is the total number of L_g sets where $l \in L_g$ after Algorithm 10 is executed on a planning problem.

Definition 6.12. Set $\Lambda(Sim_d)$ represents the intersection of all L_g sets corresponding to all goals $g_{d\alpha_t}$ in a similarity goal set Sim_d.

Definition 6.13. $score(Sim_d)$ represents the score of a similarity goal set Sim_d and is the sum of the scores of all landmarks and all relaxed landmarks found in $\Lambda(Sim_d)$.

Definition 6.14. Set Γ is the set of all Sim_d sets.

Definition 6.15. A *contextual decomposition* $Context_d$ is a set of dead-end agent goal sets G_d in which no duplicate dead-agent goals $g_{d\alpha_t} \in G$ are found in any of the dead-end agent goal sets $G_d \in Context_d$ and where the union of all dead-end agent goals $g_{d\alpha_t}$ in all $G_d \in Context_d$ is equal to all dead-end agent goals in G.

Definition 6.16. Set Σ is the set of all contextual decompositions.

The landmark scoring procedure (Algorithm 15 which computes in polynomial time) starts by first identifying the landmarks and relaxed landmarks that are found in common between each dead-end agent goal corresponding L_g set (Figure 6.6) in order to determine potential similarities (such as the location of the dead-end agents in example 6.8) between dead-end agent goals and group the goals into similarity goal sets (lines 1 to 4).

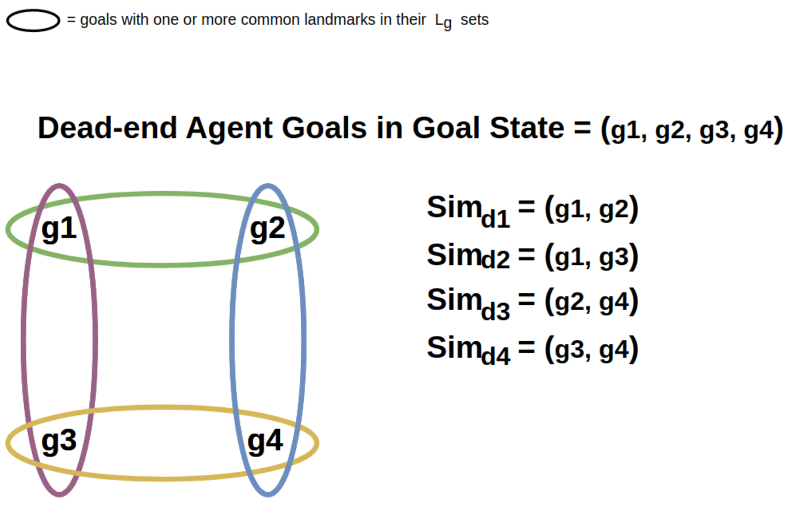

= goals with one or more common landmarks in their L_g sets
Dead-end Agent Goals in Goal State = (g1, g2, g3, g4)
g1
g2
g3
g4
Sim_{d1} = (g1, g2)
Sim_{d2} = (g1, g3)
Sim_{d3} = (g2, g4)
Sim_{d4} = (g3, g4)

Fig. 6.6 Basic Example of Creating Similarity Goal Sets from landmarks found in common between L_g sets corresponding to dead-end agent goals

We start by scoring each extracted landmark based on the number of L_g sets it was found in during the landmarks extraction procedure (Algorithm 13). We have chosen to weight a fact landmark double to an action landmark because a START-END action landmark pair portrays the same event or the same disjunction of events at the start and end time points and provides identical information for our decomposition heuristic (lines 5 to 8).

Algorithm 15 Algorithm for Scoring Landmarks

```
Input: all landmarks $l$ and all $L_g$ sets extracted from $\Pi_i$
Output: $score(l)$ of each landmark $l$ extracted from $\Pi_i$
1:  for all extracted landmarks $l$ do
2:      score($l$) = 0
3:      for all sets $L_g$ do
4:          if $l \in L_g$ then
5:              if $l$ == fact landmark then
6:                  score($l$) += 2
7:              else if $l$ == action landmark then
8:                  score($l$) += 1
9:              end if
10:         end if
11:     end for
12: end for
13: return $score(l)$ of each landmark $l$ extracted from $\Pi_i$
```

We then create a similarity goal set Sim_d (Algorithm 16 which computes in polynomial time) for each maximum collection of unique dead-end agent goals that have at least one identical landmark in their respective backchaining L_g sets (line 10) when the time points of the landmarks are ignored.

Algorithm 16 Algorithm for creating the Similarity Goal Sets

```
Input: score(l) of each landmark l extracted from $\Pi_i$
Output: $\Gamma$ (the set of all similarity goal sets $Sim_d$)
1: for all scored landmarks $l$ do
2:     if score($l$) > 1 then
3:         create similarity goal set $Sim_d$
4:         for all sets $L_g$ do
5:             $g_{d\alpha_t}$ = dead-end agent goal responsible for extracting $L_g$
6:             if $l \in L_g$ then
7:                 add $g_{d\alpha_t}$ to $Sim_d$
8:             end if
9:         end for
10:        if $Sim_d \notin \Gamma$ and size($Sim_d$) $\geq 2$ then
11:            add $Sim_d$ to $\Gamma$
12:        end if
13:    end if
14: end for
15: return $\Gamma$
```

Afterwards, we compute the score of each Sim_d set (Algorithm 17 which computes in polynomial time) by adding the individual score of each fact and action landmarks that are found in $\Lambda(Sim_d)$ (line 2 and 3).

Algorithm 17 Algorithm for Scoring the Similarity Goal Sets

```
Input: $\Gamma$ (the set of all similarity goal sets $Sim_d$)
Output: scored $\Gamma$
1: for all sets $Sim_d \in \Gamma$ do
2:     score($Sim_d$) = 0
3:     for all landmarks $l \in \Lambda(Sim_d)$ do
4:         score($Sim_d$) += score($l$)
5:     end for
6: end for
7: return scored $\Gamma$
```

Example 6.17. When backchaining from the following nine dead-end agent goals {*(delivered car0), (delivered car7), (delivered car8), (delivered car10), (delivered car15), (delivered car18), (delivered car19), (delivered car23), (delivered car28)* } during the execution of Algorithm 13 on the *RTAM_5_1_35* planning problem we obtain the following duplicate landmarks:

HOLDS ((at car accident_location0)) - fact landmark

OCCURS (START(confirm_accident police_car car accident_location0)) - action landmark

OCCURS (END(confirm_accident police_car car accident_location0)) - action landmark

The landmarks are found in all nine L_g sets corresponding to the nine dead-end agent goals. Therefore, the score of the fact landmark will be 18 and the score of each action landmark will be 9 (Algorithm 15). The *(delivered ?car)* dead-end agent goals meet the condition of having at least one landmark in common in their respective L_g sets so they qualify for grouping into a duplicate landmark dead-end agent goal set Sim_d (Algorithm 16). Since the three landmarks are the only duplicate landmarks between the L_g sets of the goals in Sim_d, score(Sim_d) is 36, the sum of the scores of all three duplicate landmarks (Algorithm 17). The end result is a goal set with not only all the cars from a specific location in the initial state (such as the decomposition in the RTAM Introduction example but algorithmically obtained without human intuition) but also with identical mandatory *confirm_accident* operations performed by parent agents onto dead-end agents after the initial state of *RTAM_5_1_35*.

Creating Contextual Decompositions from the Similarity Goal Sets

The aim of our procedure is to arrange the similarity goal sets into multiple contextual decompositions which are expected to provide better solutions than the contextual decomposition obtained from the maximum quantity of unique parent agent groups decomposition (described in Section 6.4.1.2). This is similar to how humans use the available context and information to generate a variety of alternative solutions to a particular problem.

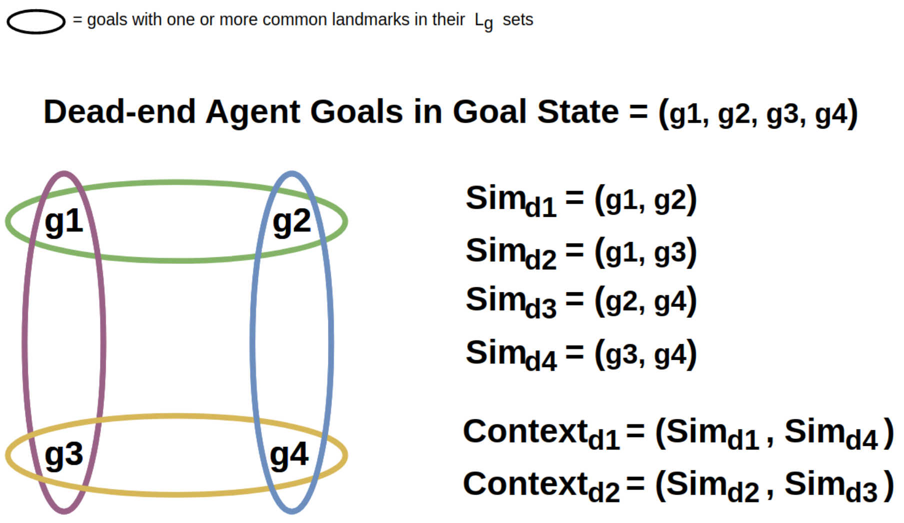

= goals with one or more common landmarks in their L_g sets
Dead-end Agent Goals in Goal State = (g1, g2, g3, g4)
g1
g2
g3
g4
Sim_{d1} = (g1, g2)
Sim_{d2} = (g1, g3)
Sim_{d3} = (g2, g4)
Sim_{d4} = (g3, g4)
$Context_{d1}$ = (Sim_{d1} , Sim_{d4})
$Context_{d2}$ = (Sim_{d2} , Sim_{d3})

Fig. 6.7 Basic Example of Creating Distinct Contextual Decompositions from the same Similarity Goal Sets

The previously obtained similarity goal sets can vary in size and can have common dead-end agent goals among them depending on the landmarks and relaxed landmarks identified in the landmarks extraction procedure. Therefore, we combine the similarity goal sets into all possible contextual decompositions so that each contextual decomposition contains all the dead-end agent goals of the planning problem and has no duplicate goals among any of its similarity goal sets (Figure 6.7). The procedure for creating the contextual decompositions is described in Algorithm 18 (which computes in polynomial time) and starts by sorting all similarity goal sets into two sorted sets: one based on the score of the similarity goal sets (line 1) and one based on the size of the similarity goal sets (line 2). We then create a contextual decomposition starting from each similarity goal set in the two sorted sets (lines 4 and 5).

Algorithm 18 Algorithm for Creating the Landmarks-based Contextual Decompositions

```
Input: scored $\Gamma$ (the set of all scored similarity goal sets) and $G$ in $\Pi_i$
Output: $\Sigma$ (the set of all contextual decompositions)
1: sorted_score($\Gamma$) = sort all $Sim_d \in \Gamma$ by highest score first
2: sorted_size($\Gamma$) = sort all $Sim_d \in \Gamma$ by largest size first
3: for all sorted($\Gamma$) $\in$ {sorted_score($\Gamma$), sorted_size($\Gamma$)} do
4:    for all $Sim_d \in$ sorted($\Gamma$) do
5:       create a new contextual decomposition $Context_d$
6:       add $Sim_d$ to $Context_d$ // (the starting similarity goal set of the contextual decom-
         position)
7:       for all $Sim_{d'}$ in sorted($\Gamma$) do
8:          if $Sim_{d'}$ is after $Sim_d$ in sorted($\Gamma$) then
9:             if $Sim_d$ and $Sim_{d'}$ do not have any common goals then
10:               add $Sim_{d'}$ to $Context_d$
11:            end if
12:         end if
13:      end for
14:      add $Context_d$ to $\Sigma$
15:   end for
16: end for
17: for all $Context_d \in \Sigma$ do
18:   if union of all goals in all $Sim_d \in Context_d$ != all dead-end agent goals $g_{d\alpha_t} \in G$
      then
19:      remove $Context_d$ from $\Sigma$
20:   end if
21: end for
22: return $\Sigma$
```

Each contextual decomposition will contain the starting similarity goal set it was initially created from (line 6) along with all following similarity goal sets in the respective sorted set from which the contextual decomposition was created provided there are no overlapping dead-end agent goals among any of the sets within a contextual decomposition (lines 8 to 10). For example, if a goal set Sim_{d1} is part of a contextual decomposition $Context_d$ and contains one or multiple goals that are also part of a goal set Sim_{d2} and Sim_{d1} is in front of Sim_{d2} in their respective sorted set, we do not add Sim_{d2} to $Context_d$. The procedure ends by eliminating each contextual decomposition $Context_d$ that has any of the instance planning problem dead-end agent goals missing from the union of all similarity goal sets in $Context_d$ (lines 17 to 19).

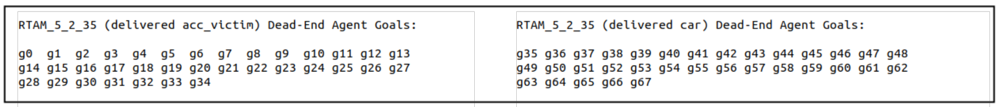
Example 6.18. In the *RTAM_5_2_35* planning problem, we have 68 dead-end agent goals (Figure 18 A). Goals g0 to g34 have dead-end agents with the *acc_victim* dead-end type as parameters and goals g35 to g67 have dead-end agents with the *car* dead-end type as parameters. Applying algorithm Algorithm 18 on *RTAM_5_2_35* yields 15 valid contextual decompositions. Figures 6.8 B and 6.8 C represent two of the contextual decompositions of *RTAM_5_2_35* that were created using the sorting based on the size of the identified similarity goal sets. Both contextual decompositions are valid as every dead-end agent goal in the goal state of RTAM_5_2_35 is found only once within the similarity goal sets of each contextual decomposition.

Algorithm 18 uses the available contextual information to construct distinct solving strategies similar to how humans interpret the available information and context of a given problem to create multiple potential solving strategies. An example is provided in Figure 18 B which contains a straight-forward decomposition based on the similarity of the agents in the *(delivered subject)* dead-end agent goals (with the former similarity set containing all *(delivered ?acc_victim)* goals and the latter similarity set containing all *(delivered ?car)* goals) while Figure 18 C contains a more comprehensive decomposition that takes into account:

- similarities between the location of the dead-end agents in the initial state - identified via identical *(at ?car ?location)* and *(at ?acc_victim ?location)* relaxed agent fact landmarks between all dead-end agent goals in RTAM_5_2_35
- similarities between the mandatory rescue operations performed by parent agents onto dead-end agents at identical locations - identified via identical *confirm_accident* and *first_aid* relaxed agent action landmarks between all dead-end agent goals in RTAM_5_2_35
- similarities between the paths of the mandatory movement operations of parent agents for achieving the goals - identified via identical *move* relaxed agent action landmarks between all dead-end agent goals in RTAM_5_2_35

An alternative method for decomposing dead-end agent goals can be designed by identifying the "location" generic type using TIM [57] and decomposing the dead-end agent goals based on their location parameters in the initial state. However, our relaxed landmarks method for identifying similarities and creating contexts between dead-end

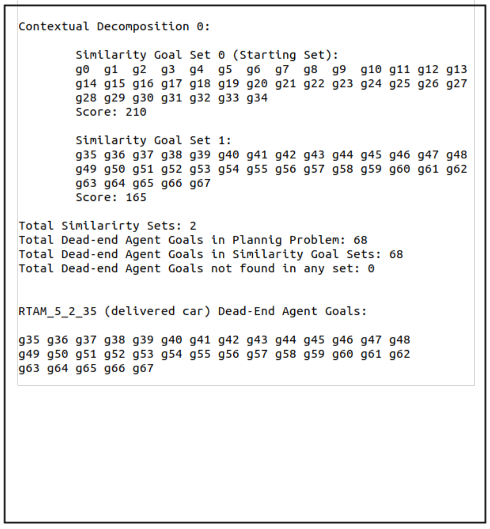
A



```
RTAM_5_2_35 (delivered acc_victim) Dead-End Agent Goals:

g0  g1  g2  g3  g4  g5  g6  g7  g8  g9  g10 g11 g12 g13
g14 g15 g16 g17 g18 g19 g20 g21 g22 g23 g24 g25 g26 g27
g28 g29 g30 g31 g32 g33 g34
```

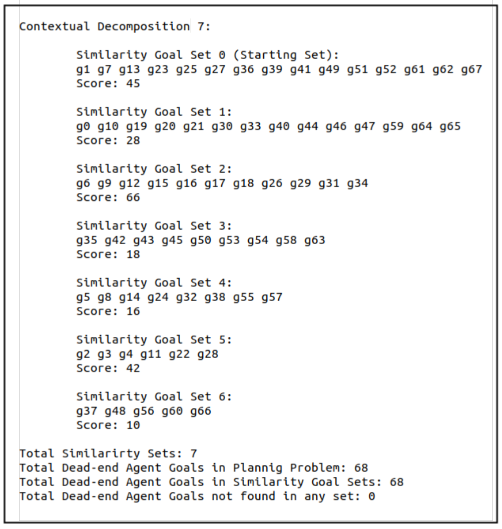
```
RTAM_5_2_35 (delivered car) Dead-End Agent Goals:

g35 g36 g37 g38 g39 g40 g41 g42 g43 g44 g45 g46 g47 g48
g49 g50 g51 g52 g53 g54 g55 g56 g57 g58 g59 g60 g61 g62
g63 g64 g65 g66 g67
```

B



```
Contextual Decomposition 0:

        Similarity Goal Set 0 (Starting Set):
        g0  g1  g2  g3  g4  g5  g6  g7  g8  g9  g10 g11 g12 g13
        g14 g15 g16 g17 g18 g19 g20 g21 g22 g23 g24 g25 g26 g27
        g28 g29 g30 g31 g32 g33 g34
        Score: 210

        Similarity Goal Set 1:
        g35 g36 g37 g38 g39 g40 g41 g42 g43 g44 g45 g46 g47 g48
        g49 g50 g51 g52 g53 g54 g55 g56 g57 g58 g59 g60 g61 g62
        g63 g64 g65 g66 g67
        Score: 165

Total Similarirty Sets: 2
Total Dead-end Agent Goals in Plannig Problem: 68
Total Dead-end Agent Goals in Similarity Goal Sets: 68
Total Dead-end Agent Goals not found in any set: 0


RTAM_5_2_35 (delivered car) Dead-End Agent Goals:

g35 g36 g37 g38 g39 g40 g41 g42 g43 g44 g45 g46 g47 g48
g49 g50 g51 g52 g53 g54 g55 g56 g57 g58 g59 g60 g61 g62
g63 g64 g65 g66 g67
```

C



```
Contextual Decomposition 7:

        Similarity Goal Set 0 (Starting Set):
        g1 g7 g13 g23 g25 g27 g36 g39 g41 g49 g51 g52 g61 g62 g67
        Score: 45

        Similarity Goal Set 1:
        g0 g10 g19 g20 g21 g30 g33 g40 g44 g46 g47 g59 g64 g65
        Score: 28

        Similarity Goal Set 2:
        g6 g9 g12 g15 g16 g17 g18 g26 g29 g31 g34
        Score: 66

        Similarity Goal Set 3:
        g35 g42 g43 g45 g50 g53 g54 g58 g63
        Score: 18

        Similarity Goal Set 4:
        g5 g8 g14 g24 g32 g38 g55 g57
        Score: 16

        Similarity Goal Set 5:
        g2 g3 g4 g11 g22 g28
        Score: 42

        Similarity Goal Set 6:
        g37 g48 g56 g60 g66
        Score: 10

Total Similarirty Sets: 7
Total Dead-end Agent Goals in Plannig Problem: 68
Total Dead-end Agent Goals in Similarity Goal Sets: 68
Total Dead-end Agent Goals not found in any set: 0
```

Context$_{d0}$
Context$_{d7}$
6
9
Optimal Decomposition of All Dead-end Agent Goals in the RTAM_5_2_35 Goal State

Fig. 6.8 Valid Contextual Decompositions of the *RTAM_5_2_35* Planning Problem with distinct Similarity Goal Sets

agent goals is not limited to agent locations in the initial state and can potentially identify other types of connections besides a common location (such as the mandatory rescue and mandatory movement operations in example 6.18), can identify connections both in the initial state and beyond the initial state of a planning problem and can identify both factual and operational connections between the dead-end agent goals of a planning problem.

6.4.1.2 Decomposing Dead-end Agent Goals Using the Maximum Quantity of Unique Parent Agent Groups

Section Overview In this section, we describe the dead-end agent goals decomposition based on the maximum quantity of unique parent agent groups.

Definition 6.19. *total_agents(t)* is the total number of agents of a specific parent type $t \in T_d$ that are present in a planning problem.

Definition 6.20. M is the set of the maximum quantity of unique parent agent groups $\mu_{p\alpha_t}$ where an α_t is found in only one of the unique parent agent groups $\mu_{p\alpha_t}$ in M. There can be distinct configurations of M within a planning problem based on all possible configurations of $\mu_{p\alpha_t}$ but the size of M will always be equal to the minimum among all *total_agents(t)* for all parent types $t \in T_d$.

The maximum quantity of unique parent agent groups with no common agents among them in a planning problem is also used as a contextual decomposition criterion. The intent is to deploy all possible or as close to all possible agents in solving a problem, as usually the more deployed resources the better the results [59]. The method (Algorithm 19 which computes in polynomial time) uses the size of M (lines 1 to 9) as the total number of dead-end agent goal sets in a contextual decomposition (line 12). The intent is to utilise as many unique parent agents groups with no common agents among them as possible (all $\mu_{p\alpha_t} \in M$) concurrently for solving all dead-end agent goals. The method evenly splits all dead-end agent goals among all dead-end agent goal sets at random (line 12).

Algorithm 19 Algorithm for creating the Maximum Quantity of Unique Parent Agent Groups Contextual Decomposition

```
Input: $T_d$ in $\Pi_i$, all $\alpha_t$ in $\Pi_i$, $G$ in $\Pi_i$ and $\Sigma$ (the set of all contextual decompositions)
Output: updated $\Sigma$ (which now also contains the maximum quantity of unique parent agent groups contextual decomposition)
 1: for all parent types $t \in T_d$ do
 2:     total_agents(t) = 0
 3:     for all $\alpha_t$ in $\Pi$ do
            total_agents(t)++
 4:     end for
 5: end for
 6: size($M$) = $\infty$
 7: for all parent types $t \in T_d$ do
 8:     if total_agents(t) < size($M$) then
 9:         size($M$) = total_agents(t)
10:     end if
11: end for
12: divide all dead-end agent goals $g_{d\alpha_t} \in G$ evenly and randomly among a size($M$) number
    of $G_d$ sets
13: create a new contextual decomposition $Context_d$ and add to it all $G_d$ sets
14: add $Context_d$ to $\Sigma$
```

Example 6.21. In the *RTAM_5_2_35* planning problem, we have the following parent agents for the dynamic types $t \in T_d$:

ambulance0 ambulance1 ambulance2 ambulance3 - ambulance
fire_brigade0 fire_brigade1 fire_brigade2 - fire_brigade
police_car0 police_car1 police_car2 police_car3 police_car4 - police_car
tow_truck0 tow_truck1 tow_truck2 tow_truck3 tow_truck4 tow_truck5 tow_truck6 - tow_truck

Therefore, the total_score(t) for the above dynamic types are:

total_score(ambulance) = 4
total_score(fire_brigade) = 3
total_score(police_car) = 5
total_score(ambulance) = 7

The minimum total_score(t) is 3. Therefore, applying Algorithm 19 to the *RTAM_5_2_35* planning problem will yield a contextual decomposition $Context_d$ composed of three dead-end agent goal sets G_d in which all of the 68 dead-end agent goals of the planning problem are evenly and arbitrarily divided.

6.4.2 Creating and Solving the Dead-end Agents Goals Tactical Planning Problems

Section Overview A tactical planning problem [15] (described in Section 6.4.2) will be created for each dead-end agent goal set or similarity goal set in a contextual decomposition $Context_d$. The unique parent agent groups (described in Section 6.4.1.2) will be methodically spread across all tactical problems from $Context_d$ according to an efficiency-focused agent resource management procedure in order to permit concurrency among the tactical planning problems of $Context_d$. The tactical planning problems are created in several steps that will be described in the following subsections.

6.4.2.1 Initial Creation of the Tactical Planning Problems of a Contextual Decomposition

Section Overview In this section, we describe the procedure for the initial creation of the tactical planning problems Π_t from the dead-end agent goal sets G_d of a contextual decomposition $Context_d$ (described in Section 6.4.1) and the reasoning behind the design choices.

Definition 6.22. An action a is an *initial action* if there is another action a' in Π_i where $pre_{a'} \subseteq pre_a$ and $eff_{a'} = eff_a$.

Definition 6.23. An *tactical planning problem* constructed from a corresponding dead-end agent goal set G_d is defined as a tuple $\Pi_t := \{P, V, A, I, G\}$ where $\{P, V, A\}$ in Π_t are initially inherited from an instance planning problem Π_i from which Π_t is derived. I in Π_t is initially populated by all the facts from I in Π_i except for all facts which contain a parameter formed of a dead-end agent that is not part of any of the parameters of the goals in G_d. $\{P, A, I\}$ in Π_t are customised to allow only the agents from a single unique parent agent group $\mu_{p\alpha_t}$ in Π_i among the parameters of the initial actions $a \in A$ when solving Π_t. The *(stp_complete_mission)* proposition has been added to P in Π_t and represents the only goal in the goal state G in Π_t. The goals in G_d are added as action preconditions necessary to be achieved in order to satisfy G in Π_t.

Specification 6.24. The *(stp_selected_parent_dynamic_type ?parent_dynamic_type - parent_dynamic_type)* template is used to create and add a new proposition to P in Π_t for each parent type t in Π_i that represents the type or supertype (provided the supertype is a dynamic type) of a parent agent $p\alpha_t$ in Π_i. Each proposition will be constructed according to the parent type it represents.

Specification 6.25. The *(stp_not_selected_parent_dynamic_type)* template is used to create and add a new proposition p to P in Π_t for each parent type t in Π_i. A *(stp_not_selected-_parent_dynamic_type)* fact is created and added to I in Π_t for each newly created proposition p. Each proposition and corresponding fact will be constructed according to the parent type they represent.

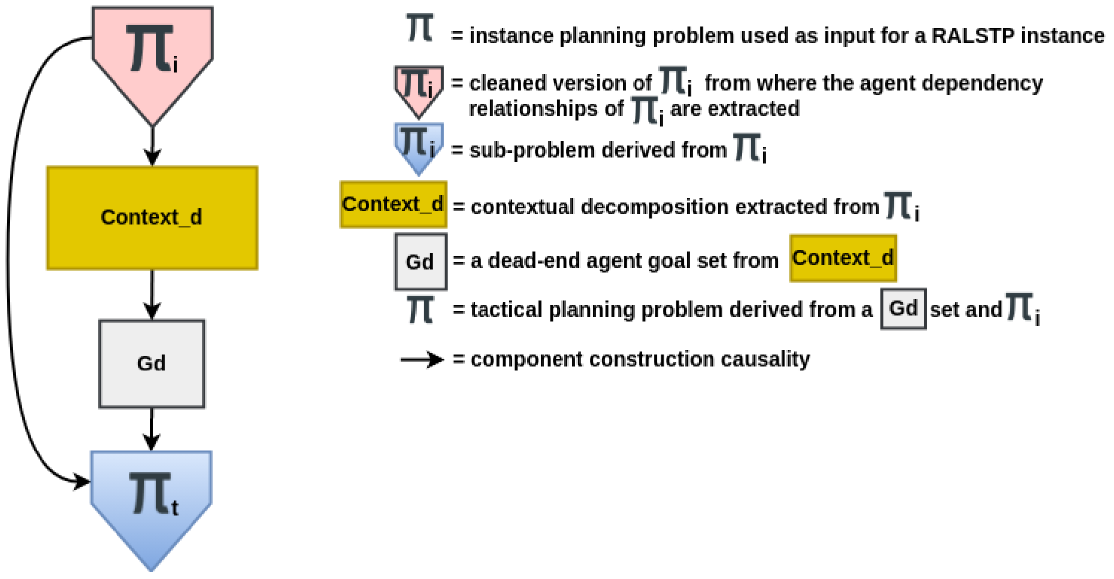
Specification 6.26. The *stp_select_parent_dynamic_type* action name template is used to create and add a new instantaneous durative action $a := \{pre_a,\ eff_a\}$ to the action set A in Π_t for each parent type t in Π_i. Each action name, parameter, condition, and effect will be constructed according to the parent type action a represents. Each action a will have *?parent_dynamic_type - parent_dynamic_type* as the only parameter. Each action a will have *(stp_not_selected_parent_dynamic_type)* as a positive condition in pre_a and *(not (stp_not_selected_parent_dynamic_type))* as negative effect in eff_a. Each action a will also have an *(stp_selected_parent_dynamic_type ?parent_dynamic_type - parent_dynamic_type)* positive effect in eff_a for the parent type it represents as well as for each of the supertypes of the parent type it represents provided that the supertypes are also dynamic types.

Specification 6.27. A *(stp_selected_parent_dynamic_type ?parent_dynamic_type - parent-_dynamic_type)* positive condition is created and added to the precondition $pre_{\vdash a}$ of every durative initial action $a \in A$ when A in Π_t if a has parameters which contain the parent type or supertype (provided the supertype is also a dynamic type) of a parent agent $p\alpha_t$ in Π_i, one condition for each such parameter in a. Each condition will be constructed according to the parent type of its respective parameter. If a is an instantaneous durative action then each condition will be added to pre_a.

A tactical planning problem Π_t (Figure 6.9) is constructed for each dead-end agent goal set G_d in a given contextual decomposition $Context_d$ in order to divide the difficulty of solving all dead-end agent goals of the instance planning problem derived from the instance problem among multiple tactical planning problems. By dividing the dead-end agent goal sets among tactical planning problems we aim to obtain problems less difficult than the all dead-end agent goals planning problem Π_d as the total number of dead-end agents $N(d\alpha)$ in each tactical planning problem Π_t is potentially smaller than the equivalent difficulty metric of Π_d. Solving all tactical planning problems corresponding to a contextual decomposition will cumulatively achieve all the dead-end agent goals of the instance planning problem (detailed in Section 6.4.2.6).

The procedure for the initial creation of a tactical planning problem is shown in Algorithm 20, which computes in polynomial time. Each tactical planning problem Π_t is initially created from $\{P, V, A\}$ in Π_i (line 1) and has the initial state I in Π_t initially populated by all the facts from I in Π_i except for the facts which contain one or more parameters formed of a dead-end agent that is not part of any of the parameters of the goals in G_d (lines 3 to 5). The objects in each Π_t will be defined as constants in order to allow the incorporation of facts and goals from the instance planning problem as conditions to the actions of the tactical planning problems (line 2).

The goal state of each tactical planning problem Π_t is populated only by the *(stp_complete_mission)* goal, without any of the goals in the dead-end agent goal set G_d corresponding to Π_t (lines 8 and 9). The goals in G_d will be added as action preconditions necessary to be achieved in order to satisfy the goal state of Π_t (detailed in Section 6.4.2.2).



Π_i
Context_d
Gd
Π_t
Π = instance planning problem used as input for a RALSTP instance
Π_i = cleaned version of Π_i from where the agent dependency relationships of Π_i are extracted
Π_i = sub-problem derived from Π_i
Context_d = contextual decomposition extracted from Π_i
Gd = a dead-end agent goal set from Context_d
Π = tactical planning problem derived from a Gd set and Π_i
= component construction causality

Fig. 6.9 Flowchart of Creating a Tactical Planning Problem

The tactical planning problems have no inactive dynamic objects as each Π_t was derived from the instance planning problem Π_i which had all its inactive dynamic objects removed when it was cleaned. The difficulty of tactical planning problems is further decreased by reducing the total number of parent agents $N(p\alpha)$ to the minimum required for respecting the constraints imposed by the agent dependency relationships for achieving dead-end agent goals. This is accomplished by modifying and supplementing the propositions, facts and actions in Π_t to force a planner to consider only the agents in a single unique parent agent group $\mu_{p\alpha_t}$ among the parameters of the initial actions of Π_t and, in effect, to reduce the total number of parent agents $N(p\alpha)$ in Π_t while still allowing a planner to chose which $\mu_{p\alpha_t}$ it uses for solving Π_t. The planner is forced to select a unique parent agent group $\mu_{p\alpha_t}$ and use only the agents in the selected $\mu_{p\alpha_t}$ when attempting to solve Π_t. This is accomplished by adding new *stp_select_parent_dynamic_type* actions and new *(stp_not_selected_parent_dynamic_type)* facts to each Π_t and by adding *(stp_selected_parent_dynamic_type ?parent_dynamic_type - parent_dynamic_type)* positive conditions to all initial actions in each Π_t (lines 10 to 15).

Algorithm 20 Algorithm for the Initial Creation of a Tactical Planning Problem

```
Input: $\Pi_i$ and a dead-end agent goal set $G_d \in Context_d$ when $Context_d$ in $\Pi_i$
Output: a tactical planning problem $\Pi_t$
1: create new tactical planning problem $\Pi_t := \{P, V, A, I, G\}$ where $\{P, V, A\}$ in $\Pi_i$
2: define all objects in $\Pi_t$ as constants
3: for all facts $f \in I$ when $I$ in $\Pi_i$ do
4:    if $f$ does not contain any parameter formed of a dead-end agent that is part of a
      dead-end agent goal outside of $G_d$ then
5:       add $f$ to $I$ in $\Pi_t$
6:    end if
7: end for
8: add the (stp_complete_mission) proposition to $P$ in $\Pi_t$
9: add (stp_complete_mission) as the only goal in $G$ in $\Pi_t$
10: add the (stp_selected_parent_dynamic_type ?parent_dynamic_type - parent_dynamic_type) propositions to $P$ in $\Pi_t$ (as described in Specification 6.24)
11: add the (stp_not_selected_parent_dynamic_type) propositions to $P$ in $\Pi_t$ (as described
    in Specification 6.25)
12: add the (stp_not_selected_parent_dynamic_type) facts to $I$ in $\Pi_t$ (as described in
    Specification 6.25)
13: add the stp_select_parent_dynamic_type actions to $A$ in $\Pi_t$ ((as described in Specifica-
    tion 6.26)
14: for all initial actions $a \in A$ when $A$ in $\Pi_t$ do
15:    add the (stp_selected_parent_dynamic_type ?parent_dynamic_type - par-
       ent_dynamic_type) positive conditions to $pre_a$(as described in Specification 6.27)
16: end for
17: return $\Pi_t$
```

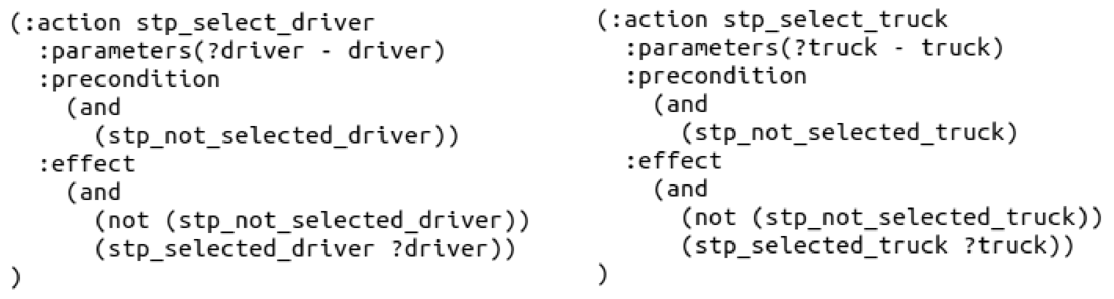
Example 6.28. Set {*(at package9 s9), (at package10 s7)*} represents the dead-end agent goal set $G_d \in Context_d(DLOG-5-5-10)$ used in the creation the tactical planning problem Π_t for this example. Running Algorithm 20 with DLOG-5-5-10 and {*(at package9* *s9), (at package10 s7)*} as inputs will add all initial state facts in DLOG-5-5-10 to Π_t except for the facts which contain one or more parameters formed of a dead-end agent distinct from *(package9)* or *(package10)*.

In the Driverlog DLOG-5-5-10 planning problem, we have only the following parent types: *{driver, truck}*. The *driver* and *truck* parent types do not have any supertypes that are dynamic types as well as found in at least one parameter of at least one initial action. Therefore, when applying Algorithm 20 to the DLOG-5-5-10 planning problem we obtain and add the following propositions to each tactical planning problem Π_t:

(stp_selected_driver ?driver - driver)
(stp_not_selected_driver)
(stp_selected_truck ?truck - truck)
(stp_not_selected_truck)

The initial state I of each Π_t will get populated with two extra facts: *(stp_not_selected_driver)* and *(stp_not_selected_truck)*.

The actions *stp_select_driver* and *stp_select_truck* (Figure 6.10) will be constructed from the *stp_select_parent_dynamic_type* template and the *{driver, truck}* parent types and added to each action set A of each Π_t.

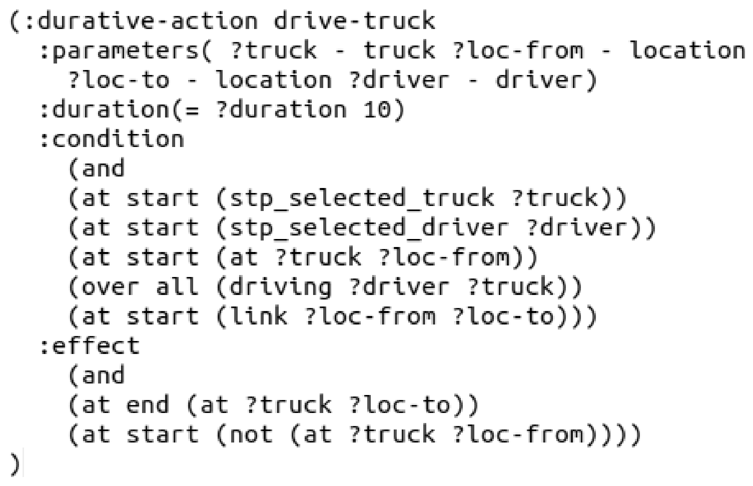


```
(:action stp_select_driver
  :parameters(?driver - driver)
  :precondition
    (and
      (stp_not_selected_driver))
  :effect
    (and
      (not (stp_not_selected_driver))
      (stp_selected_driver ?driver))
)
```

```
(:action stp_select_truck
  :parameters(?truck - truck)
  :precondition
    (and
      (stp_not_selected_truck)
  :effect
    (and
      (not (stp_not_selected_truck))
      (stp_selected_truck ?truck))
)
```

Fig. 6.10 The *stp_select_driver* and *stp_select_truck* actions implemented from the *stp_select_parent_dynamic_type* template and the *{driver, truck}* parent types in the Driverlog Domain

The durative action *drive-truck* (Figure 6.11) is an initial action as it is found in the action set of the DLOG-5-5-10 planning problem. The action contains both a *driver* parameter and a *truck* parameter. Therefore, it will get two new positive conditions added to $pre_{\vdash drive-truck}$: *(stp_selected_truck ?truck)* and *(stp_selected_driver ?driver)*. All other initial actions a in Π_t that contain any *driver* parameters or any *truck* parameters will also get added a corresponding *(stp_selected_driver ?driver)* positive condition or *(stp_selected_truck ?truck)* positive condition to their precondition $pre_{\vdash a}$ (or to pre_a if a is an instantaneous durative action) for every *driver* or *truck* parameter in a.



```
(:durative-action drive-truck
  :parameters( ?truck - truck ?loc-from - location
    ?loc-to - location ?driver - driver)
  :duration(= ?duration 10)
  :condition
    (and
    (at start (stp_selected_truck ?truck))
    (at start (stp_selected_driver ?driver))
    (at start (at ?truck ?loc-from))
    (over all (driving ?driver ?truck))
    (at start (link ?loc-from ?loc-to)))
  :effect
    (and
    (at end (at ?truck ?loc-to))
    (at start (not (at ?truck ?loc-from))))
)
```

Fig. 6.11 Modified *drive_truck* initial action that allows only selected agents among its parameters

The added modifications will force a planner to select and use only one of the existing parent agents with the *driver* parent type and only one of the existing parent agents with the *truck* parent type when solving the goals of the tactical planning problem derived from the DLOG-5-5-10 planning problem.

6.4.2.2 Adding Agent Goals to Tactical Planning Problems

Section Overview In this section, we described how the goals in the dead-end agent goals set G_d corresponding to a particular tactical planning problem Π_t and the goals of all parent agent goals are added as action preconditions in Π_t.

Specification 6.29. The *(stp_parent_dynamic_type_complete)* template is used to create and add a new proposition to P in Π_t for each parent type t in Π_i. Each proposition will be constructed according to the parent type it represents.

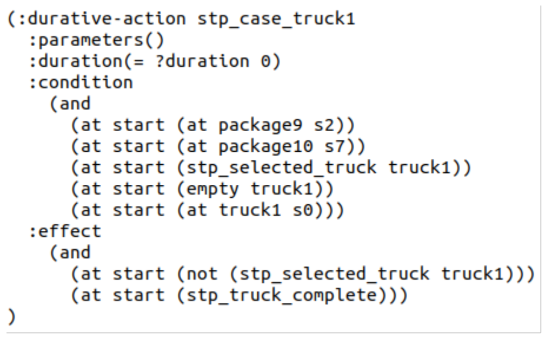
Specification 6.30. The *stp_case_constant* action name template is used to create and add a new instantaneous durative action $a := \{pre_a, eff_a\}$ to the action set A in Π_t for each parent agent $p\alpha_t$ in Π_i. Each action name, condition, and effect will be constructed according to the parent type and constant action a represents. Each action a will have no parameters and a *(stp_selected_parent_dynamic_type constant)* positive condition in precondition pre_a. Each action a will have as positive conditions in precondition pre_a all goals from the corresponding dead-end agent goal set G_d of Π_t.

Each action a will also have as positive conditions in precondition pre_a all parent agent goals $g_{p\alpha_t}$ in Π_i that have the constant action a represents as the only parent agent $p\alpha_t$ in Π_i in the parameters of $g_{p\alpha_t}$ if such goals exist. If there are no parent agent goals with the constant action a represents as the only parent agent, action a will have as conditions in precondition pre_a all propositions and functions representing all facts $f_{p\alpha_t} \in I$ when I in Π_i that have the constant action a represents as the only parent agent $p\alpha_t$ in Π_i in the parameters of $f_{p\alpha_t}$.

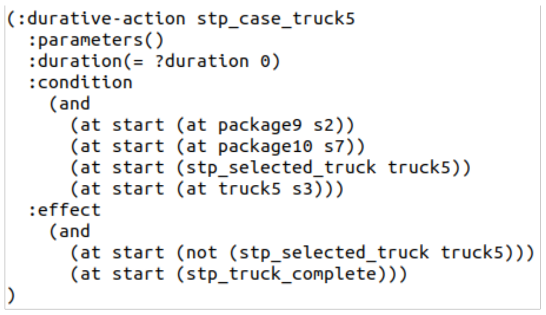
Each action a will have *(not (stp_selected_parent_dynamic_type constant)* as a negative effect in eff_a so that only one such action can be executed per parent type. Each action a will also have *(stp_parent_dynamic_type_complete))* as a positive effect in eff_a which will be required for satisfying conditions required for achieving the goal state of Π_t.

Specification 6.31. The instantaneous durative action *complete_tactical_mission* $:=$ $\{pre_{complete\_tactical\_mission}, eff_{complete\_tactical\_mission}, dur_{complete\_tactical\_mission}\}$ is created and added to A in Π_t. The action has no parameters and has all goals from the corresponding dead-end agent goal set G_d of Π_t as positive conditions in precondition $pre_{complete\_tactical\_mission}$. The action will also have a *(stp_parent_dynamic_type_complete)* positive condition in precondition $pre_{complete\_tactical\_mission}$ for each parent type t in Π_i. The action will have *(stp_complete_mission)* as a positive effect in $eff_{complete\_tactical\_mission}$.

The dead-end agent goals in the goal set G_d corresponding to Π_t are added to Π_t in the form of action preconditions necessary to be achieved in order to satisfy the goal state of Π_t. This is accomplished by supplementing the action set A in Π_t with *stp_case_constant* instantaneous durative actions (Figure 6.12). The actions will have as part of their preconditions all goals in the dead-end agent goal set G_d corresponding to Π_t.



```
(:durative-action stp_case_truck1
  :parameters()
  :duration(= ?duration 0)
  :condition
    (and
      (at start (at package9 s2))
      (at start (at package10 s7))
      (at start (stp_selected_truck truck1))
      (at start (empty truck1))
      (at start (at truck1 s0)))
  :effect
    (and
      (at start (not (stp_selected_truck truck1)))
      (at start (stp_truck_complete)))
)
```

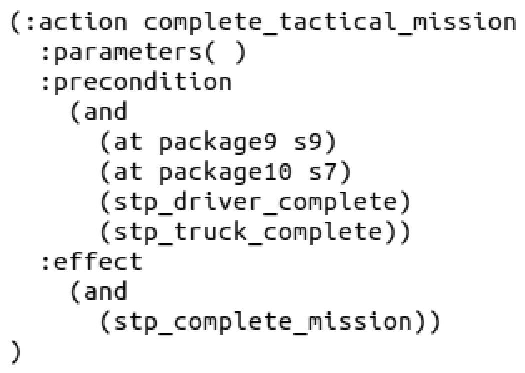


```
(:durative-action stp_case_truck5
  :parameters()
  :duration(= ?duration 0)
  :condition
    (and
      (at start (at package9 s2))
      (at start (at package10 s7))
      (at start (stp_selected_truck truck5))
      (at start (at truck5 s3)))
  :effect
    (and
      (at start (not (stp_selected_truck truck5)))
      (at start (stp_truck_complete)))
)
```

Fig. 6.12 The *stp_case_truck1* and *stp_case_truck5* actions implemented from the *stp_case_constant* template and the *truck1, truck5* parent agents defined as constants in a Tactical Planning Problem derived from the DLOG-5-5-10 Planning Problem

The *stp_case_constant* actions are designed to take advantage of the mandatory selection of a unique parent agent group $\mu_{p\alpha_t}$ necessary for solving a tactical planning problem Π_t in order to potentially decrease the solving difficulty of the parent agent goals in the planning problem from which Π_t was derived. This is accomplished by modifying Π_t in a way that forces a planner to achieve the parent agent goals associated with the parent agents in the selected $\mu_{p\alpha_t}$ in order to reach the goal state of Π_t. Achieving parent agent goals in the solutions of the tactical planning problems will either reduce the number of remaining parent agent goals to be achieved in the parent agent goals problem or eliminate the necessity of the parent agent goals planning problem altogether if all parent agent goals are achieved in the solutions of the tactical planning problems. Therefore, each tactical planning problem Π_t is modified to force a planner to achieve all parent agent goals $g_{p\alpha_t}$ in Π_i which contain in their parameters any of the parent agents $p\alpha_t$ present in the unique parent agent group $\mu_{p\alpha_t}$ in Π_t selected by the planner when solving Π_t. The modifications are obtained by applying Algorithm 21 (which computes in polynomial time) to a tactical planning problem Π_t.

The procedure adds an *stp_case_constant* action to A in Π_t for each parent agent $p\alpha_t$ in Π_i (line 3). The goals in the dead-end agent goal set G_d for which Π_t was created are added as positive conditions in all the *stp_case_constant* actions added to A in Π_t. Each *stp_case_constant* action will also have as positive conditions all parent agent goals $g_{p\alpha_t}$ in Π_i where $p\alpha_t$ is the constant action *stp_case_constant* represents and is the only parent agent in $g_{p\alpha_t}$ if any such goals exist. If there are no such goals for a specific parent agent constant $p\alpha_t$, the *stp_case_constant* action corresponding to $p\alpha_t$ will contain all initial state facts $f_{p\alpha_t}$ in Π_i where $p\alpha_t$ is the constant action *stp_case_constant* represents and is the only parent agent in $f_{p\alpha_t}$.



```
(:action complete_tactical_mission
  :parameters( )
  :precondition
    (and
      (at package9 s9)
      (at package10 s7)
      (stp_driver_complete)
      (stp_truck_complete))
  :effect
    (and
      (stp_complete_mission))
)
```

Fig. 6.13 The *complete_tactical_mission* action added to a tactical planning problem derived from dead-end agent goal set {*(at package9 s9), (at package10 s7)*}

The action set A in Π_t has also been supplemented with the *complete_tactical_mission* instantaneous durative action (line 3). *complete_tactical_mission* (Figure 6.13) is the only action in Π_t that has the *(stp_complete_mission)* goal (the only goal in the goal state of each Π_t) as a positive effect. The *complete_tactical_mission* action can only be executed if $pre_{complete\_tactical\_mission}$ is satisfied. $pre_{complete\_tactical\_mission}$ is achieved only if a planner first selects a single parent agent pair $\mu_{p\alpha_t}$ in Π_t and executes an *stp_case_constant* action for each parent agent in $\mu_{p\alpha_t}$. $\mu_{p\alpha_t}$ must be selected before executing any *stp_case_constant* action as all $pre_{stp\_case\_constant}$ have a *(stp_selected_parent_dynamic_type constant)* positive condition which can only be satisfied if a planner first selects $\mu_{p\alpha_t}$. For executing the *stp_case_constant* actions compatible with the selected $\mu_{p\alpha_t}$, a planner must first pass through the states where each $pre_{stp\_case\_constant}$ is true, hence achieving all dead-end agent goals in the G_d corresponding to Π_t (defined as positive conditions in $pre_{stp\_case\_constant}$), all parent agent goals defined as positive conditions in $pre_{stp\_case\_constant}$ if any such goals exist and all allowed $f_{p\alpha_t}$ facts defined as positive conditions in $pre_{stp\_case\_constant}$.

The tactical planning problem structure described above will force a planner to solve all the goals in the dead-end agent goal set G_d for which Π_t was created as well as all parent agent goals associated with the parent agents in the unique parent agent group $\mu_{p\alpha_t}$ selected by the planner when solving Π_t.

Algorithm 21 Algorithm for Adding Goals to a Tactical Planning Problem

```
Input: a tactical planning problem $\Pi_t$
Output: updated $\Pi_t$ (which now contains the goal in the form of action preconditions)
1: add the (stp_parent_dynamic_type_complete) propositions to $P$ in $\Pi_t$ (as described in Specification 6.29)
2: add the stp_case_constant actions to $A$ in $\Pi_t$ (as described in Specification 6.30)
3: add the complete_tactical_mission action to $A$ in $\Pi_t$ (as described in Specification 6.31)
4: return updated $\Pi_t$
```

Example 6.32. In the Driverlog DLOG-5-5-10 planning problem, we have only the following parent types that represent the type of the parent agents: *{driver, truck}*. Therefore, when applying Algorithm 21 on the DLOG-5-5-10 planning problem we obtain and add the following new propositions to each tactical planning problem:

(stp_driver_complete)
(stp_truck_complete)
(stp_complete_mission)

In this example, we will focus only on the *truck* parent type and only on the *truck1, truck5* parent agents from all parent types and parent agents found in DLOG-5-5-10. The following facts are all the f_{truck1} facts in the initial state of DLOG-5-5-10: {*(at truck1 s0), (empty truck1)*}. *truck1* has no g_{truck1} goals in the goal state of DLOG-5-5-10. The following facts are all the f_{truck5} facts in the initial state of DLOG-5-5-10: {*(at truck5 s8), (empty truck5)*}. Parent agent goal *(at truck5 s3)* is the only g_{truck5} goal in the goal state of DLOG-5-5-10. Set {*(at package9 s9), (at package10 s7)*} represent the dead-end agent goal set $G_d \in Context_d(DLOG\text{-}5\text{-}5\text{-}10)$ used in the creation the tactical planning problem Π_t for this example.

When applying Algorithm 21 to Π_t we add to A in Π_t the action *stp_case_truck1* (Figure 6.12), an action which is created by applying the *truck1* constant to the *stp_case_constant* template. The action has in $pre_{stp\_case\_truck1}$ the *(at truck1 s0) and (empty truck1)*} initial state facts as a positive condition, as the constant *truck1* represented by the action is not part of any of the goals in the goal state of DLOG-5-5-10. When applying Algorithm 21 to Π_t we also add to A in Π_t the action *stp_case_truck5* (Figure 6.12), an action which is created by applying the *truck5* constant to the *stp_case_constant* template. However, this action does not have in $pre_{stp\_case\_truck5}$ any of the f_{truck5} initial state facts as positive conditions and instead has the *(at truck5 s3)* parent agent goal as a positive condition, as the constant *truck5* represented by the action is one of the parameters of parent agent goal *(at truck5 s3)*.

Π_t will also get added the *complete_tactical_mission* action (Figure 6.13) that has the *(at package9 s9)* and *(at package10 s7)* dead-end agent goals in G_d defined as positive conditions in the $pre_{complete\_tactical\_mission}$ precondition. The *complete_tactical_mission* action also has the *stp_truck_complete* as a precondition. To achieve the goal state G in Π_t, a planner will be forced to execute the *complete_tactical_mission* action, which can only be executed if *stp_truck_complete* is first achieved by executing an *stp_case_truck* action. Executing an *stp_case_truck* can only be possible if the *(at package9 s9)* and *(at package10 s7)* goals defined as preconditions in each $pre_{stp\_case\_truck}$ are previously achieved.

6.4.2.3 Adding Parent Agent Availability Constraints To Tactical Planning Problems

Section Overview In this section, we present an efficient parent agent allocation technique for tactical planning problems by applying constraints to parent agents in a process executed outside the solution search.

Definition 6.33. A *parent agent constraint*, $constraint(p\alpha_t)$, is a boolean variable linked to a parent agent $p\alpha_t$ in Π_i.

Definition 6.34. $available(p\alpha_t)$ is a parent agent constraint assigned to every parent agent $p\alpha_t$ in Π_i. All $available(p\alpha_t)$ parent agent constraints are initially set to true when starting the strategic-tactical procedure for all parent agents $p\alpha_t$ in Π_i. An $available(p\alpha_t)$ parent agent constraint will be set to false if its corresponding parent agent $p\alpha_t$ is a parameter in any initial action present in the plan of the latest successfully solved tactical planning problem. If all parent agents $p\alpha_t$ in Π_i of a particular parent type t have had their $available(p\alpha_t)$ constraint set to false after a constraint update we reset the $available(p\alpha_t)$ constraint for all parent agents with parent type t to true before creating the next tactical planning problem.

Specification 6.35. The *(stp_free_parent_dynamic_type ?parent_dynamic_type - parent_dynamic_type)* template is used to create and add a new proposition to P in Π_t for each parent type t in Π_i. A *(stp_free_parent_dynamic_type ?parent_dynamic_type - parent_dynamic_type)* fact is created and added to I in Π_t for each $p\alpha_t$ that has its $available(p\alpha_t)$ parent agent constraint set to true. Each proposition and corresponding fact will be constructed according to the parent type and parent agent they represent.

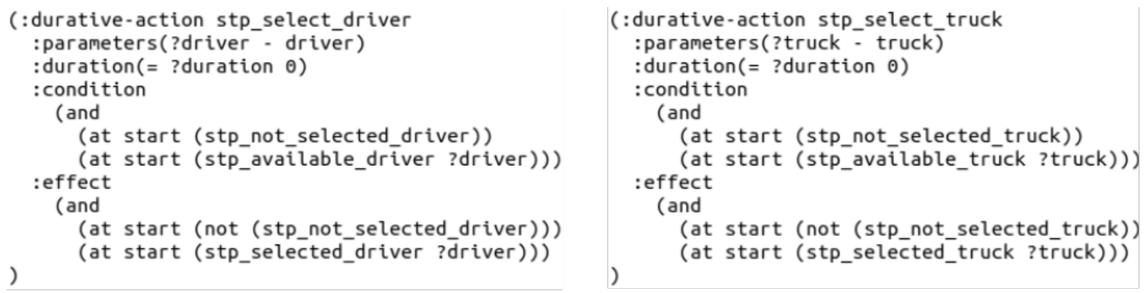
Specification 6.36. A *(stp_free_parent_dynamic_type ?parent_dynamic_type - parent_dynamic_type)* positive condition is created and added to the $pre_{stp\_select\_parent\_dynamic\_type}$ precondition of every *stp_select_parent_dynamic_type* action $\in A$ when A in Π_t. Each condition will be constructed according to the parent type represented by the *stp_select_parent_dynamic_type* action to which the condition is added.

The union of the initial states of all tactical planning problems in a given contextual decomposition $Context_d$ in Π_i will contain all the facts in the initial state as the all dead-end agent goals planning problem Π_d. The tactical planning problems will be sequentially executed on a planner based on the size of the corresponding dead-end agent goal sets (largest sets first) with the result of cumulatively achieving all the dead-end agent goals of the instance planning problem (details of the solving order and methodology will be provided in Section 6.4.2.6). Our procedure aims to achieve all dead-end agent goals using as many parent agents as possible, as usually utilising all the available resources for solving a problem produces higher quality solutions [59]. However, if we attempt to solve the tactical planning problems in the current format, we risk not having an even distribution of parent agents among the tactical problems, as it is currently possible for a planner to repeatedly select the same parent agents for solving all tactical planning problems instead of selecting distinct agents for each problem. This happens because the tactical planning problems are isolated from each other and do not exchange information about which parent agents are selected by the planner when solving a tactical planning problem. To mitigate this issue, we have created a procedure which parses the plan of a tactical planning problem and determines which parent agents were selected by the planner as initial action parameters in order to deny those agents from being repetitively used in solving the remaining tactical planning problems. In effect, we have added the $available(p\alpha_t)$ parent agent constraint to all parent agents $p\alpha_t$ in Π_i and modified the structure of tactical planning problems to only allow parent agents that have their $available(p\alpha_t)$ parent agent constraint set to true to be selected by a planner (Algorithm 22 from Section 6.4.2.4 which computes in polynomial time).

Forcing a planner to select only parent agents with favourable parent agent constraints is achieved by adding *(stp_free_parent_dynamic_type ?parent_dynamic_type - parent_dynamic_type)* as a positive condition in every $pre_{stp\_select\_parent\_dynamic\_type}$ agent selection action (lines 1 to 3).

Expressing the $available(p\alpha_t)$ parent agent constraints in a tactical planning problem Π_t is done by adding a *(stp_free_parent_dynamic_type ?parent_dynamic_type - parent_dynamic_type)* fact to I in Π_t for each parent agent $p\alpha_t$ that has $available(p\alpha_t)$ set to true (lines 16 to 17).

Example 6.37. In the Driverlog DLOG-5-5-10 planning problem, we have the following parent types: *{driver, truck}*. To be able to implement the *available* parent agent constraints in DLOG-5-5-10, we add the *(stp_free_driver ?driver)* proposition and the *(stp_free_truck ?truck)* proposition in all tactical planning problems derived from DLOG-5-5-10. We also add the *(stp_free_driver ?driver)* positive condition to $pre_{stp\_select\_driver}$ and the *(stp_free_truck ?truck)* positive condition to $pre_{stp\_select\_truck}$ in all tactical planning problems derived from derived from DLOG-5-5-10 (Figure 6.14).



```
(:durative-action stp_select_driver
  :parameters(?driver - driver)
  :duration(= ?duration 0)
  :condition
    (and
      (at start (stp_not_selected_driver))
      (at start (stp_available_driver ?driver)))
  :effect
    (and
      (at start (not (stp_not_selected_driver)))
      (at start (stp_selected_driver ?driver)))
)
```

```
(:durative-action stp_select_truck
  :parameters(?truck - truck)
  :duration(= ?duration 0)
  :condition
    (and
      (at start (stp_not_selected_truck))
      (at start (stp_available_truck ?truck)))
  :effect
    (and
      (at start (not (stp_not_selected_truck))
      (at start (stp_selected_truck ?truck)))
)
```

Fig. 6.14 The *stp_select_driver* and *stp_select_truck* actions equipped with the *(stp_free_parent_dynamic_type ?parent_dynamic_type)* parent agent constraint positive condition

The initial state of DLOG-5-5-10 is populated by the following parent agents with the *driver* parent type: $\{$*driver1*, *driver2*, *driver3*, *driver4*, *driver5*$\}$. The initial state of DLOG-5-5-10 is also populated by the following parent agents with the *truck* parent type: $\{$*truck1*,*truck2*,*truck3*,*truck4*,*truck5*$\}$. At the start of the strategic-tactical procedure, we mark all *available* parent agent constraints of all parent agents with the *truck* and *driver* parent type to true. Therefore, when we create the first tactical planning problem Π_t, we will add a *(stp_free_truck truckX)* fact to I in Π_t for all each of the five parent agents with the *truck* parent type and a *(stp_free_driver driverX)* fact I in Π_t for each of the five parent agents with the *driver* parent type.

6.4.2.4 Adding Parent Agent Goal State Constraints To Tactical Planning Problems

Section Overview In this section, we introduce an additional constraint to the parent agent selection procedure in order to allow the completion of parent agent goals at the tactical level.

Definition 6.38. *in_goal_state*$(p\alpha_t)$ is a parent agent constraint assigned to every parent agent $p\alpha_t$ in Π_i. All *in_goal_state*$(p\alpha_t)$ parent agent constraints are initially set to true for all parent agents $p\alpha_t$ in Π_i that represent at least one parameter of at least one parent agent goal $g_{p\alpha_t}$ in Π_i and set to false for all parent agents that are not found among any of the parameters of any parent agent goal $g_{p\alpha_t}$ in Π_i when starting the strategic-tactical procedure. An *in_goal_state*$(p\alpha_t)$ parent agent constraint will be set to false if its corresponding parent agent $p\alpha_t$ was selected by a planner during the successful solving of a tactical planning problem. An *in_goal_state*$(p\alpha_t)$ parent agent constraint can never be reset to true if it was previously set to false.

In order to force a planner to select the parent agents that are part of the $g_{p\alpha_t} \in G$ goals when G in Π_i we have added the *in_goal_state*$(p\alpha_t)$ parent agent constraint to all parent agents (Algorithm 22 which computes in polynomial time) and set it to true for all parent agents $p\alpha_t$ that are part of the $g_{p\alpha_t}$ parent agent goals. Therefore, when creating a tactical planning problem Π_t we only add *(stp_free_parent_dynamic_type ?parent_dynamic_type - parent_dynamic_type)* facts I in Π_t for parent agents $p\alpha_t$ with type t that have the $in\_goal\_state(p\alpha_t)$ parent agent constraint set to true provided that there is at least one such $p\alpha_t$ among all parent agents with parent type t (lines 5 to 11). If no such parent agents exist for a parent type t we will consider the $available(p\alpha_t)$ parent agent constraint and add *(stp_free_parent_dynamic_type ?parent_dynamic_type - parent_dynamic_type)* facts for all parent agents $p\alpha_t$ with type t that have the $available(p\alpha_t)$ parent agent constraint set to true (lines 14 to 17).

Algorithm 22 Algorithm for adding all Parent Agent Constraints to a Tactical Planning Problem

```
Input: all parent agents $p\alpha_t$ in $\Pi_i$, $T_d$ in $\Pi_i$ and a tactical planning problem $\Pi_t$ to which the goal has been added in the form of action preconditions
Output: updated $\Pi_t$ (which now contains the in_goal_state and available parent agent constraints)
 1: add the (stp_free_parent_dynamic_type ?parent_dynamic_type - parent_dynamic_type) propositions to $P$ in $\Pi_t$ (as described in Specification 6.35)
 2: for all stp_select_parent_dynamic_type actions in $A$ in $\Pi_t$ do
 3:     add the (stp_free_parent_dynamic_type ?parent_dynamic_type - parent_dynamic_type) positive conditions to $pre_{stp\_select\_parent\_dynamic\_type}$ (as described in Specification 6.36)
 4: end for
 5: for all parent types $t \in T_d$ do
 6:     goal_state_parent_agent_added = false
 7: end for
 8: for all parent agents $p\alpha_t$ in $\Pi_i$ do
 9:     if $in\_goal\_state(p\alpha_t)$ == true then
10:         add (stp_free_t $p\alpha_t$) fact to $I$ in $\Pi_t$
11:         goal_state_parent_agent_added = true
12:     end if
13: end for
14: if goal_state_parent_agent_added == false then
15:     for all parent agents $p\alpha_t$ in $\Pi_i$ do
16:         if $available(p\alpha_t)$ == true then
17:             add (stp_free_t $p\alpha_t$) fact to $I$ in $\Pi_t$
18:         end if
19:     end for
20: end if
21: return $\Pi_t$
```

Example 6.39. In this example, we will focus on only two parent agents from DLOG-5-5-10: *truck1* and *truck5* with parent type *truck*. When running Algorithm 22 on the DLOG-5-5-10 planning problem the *available(truck1)* and *available(truck5)* parent agent constraints will be initialised to true. However, the *in_goal_state(truck1)* parent agent constraint will be initialised to false as there is no g_{truck1} goal in the goal state of DLOG-5-5-10 and only the *in_goal_state(truck5)* will be initialised to true as goal *(at truck2 s9)* is a g_{truck5} goal in the goal state of DLOG-5-5-10. When creating a tactical planning problem Π_t obtained from a dead-end agent goal set G_d, fact *(stp_free_truck truck5)* will be added to I in Π_t as *in_goal_state(truck5)* parent agent constraint is true. However, no such fact will get added for *truck1* as, even though it has its *available(truck1)* parent agent constraint true, there is at least one parent agent with the same parent type as *truck1* that has the *in_goal_state* parent agent constraint set to true: *truck5* with *in_goal_state(truck5)* parent agent constraint true. Therefore, a planner will be forced to select *truck5* but will not be able to select *truck1* when solving Π_t. The same procedure is applied to all other parent agents in DLOG-5-5-10 which results in a planner being forced to first select only parent agents that have $g_{p\alpha_t}$ in the goal state of DLOG-5-5-10 for solving tactical planning problems if such agents exist.

6.4.2.5 Relaxed Tactical Planning Problems

Section Overview In this section, we present a relaxation to the tactical planning problems that can provide an exponential decrease in the solving difficulty in comparison to tactical problems without relaxation.

Definition 6.40. A *mandatory parent agent group* $\omega_{p\alpha_t}$ is the set of all parent agents that appear as a parameter in at least one initial action present in the relaxed plan of a tactical planning problem Π_t.

Definition 6.41. A *relaxed tactical planning problem* constructed from a corresponding tactical planning problem Π_t and a mandatory parent agent group $\omega_{p\alpha_t}$ is defined as a tuple $\Pi_r := \{P,V,A,I,G\}$ where $\{P,V,A\}$ in Π_r are initially inherited from an instance planning problem Π_i from which Π_t is derived. I in Π_r has all the non-parent agent facts from I in Π_i except for all dead-end agent facts which contain a parameter formed of a dead-end agent that is not part of any of the parameters of the goals in G_d corresponding to Π_t. I in Π_r has all parent agent facts $f_{p\alpha_t}$ in Π_i corresponding to all $p\alpha_t \in \omega_{p\alpha_t}$. $G \in \Pi_r$ has all parent agent goals $g_{p\alpha_t}$ in Π_i corresponding to all parent agents in $\omega_{p\alpha_t}$ that have their goal state constraint *in_goal_state*$(p\alpha_t)$ set to true. If there are no $g_{p\alpha_t}$ in Π_i corresponding to a particular parent agent $p\alpha_t \in \omega_{p\alpha_t}$ or if *in_goal_state*$(p\alpha_t)$ is false, $G \in \Pi_r$ will have as corresponding goals for $p\alpha_t$ all parent agent facts $f_{p\alpha_t}$ in Π_i corresponding to $p\alpha_t \in \omega_{p\alpha_t}$.

To further reduce the difficulty of a tactical planning problem Π_t, we run a reachability analysis on Π_t and parse the relaxed plan in order to mark the parent agents used in the relaxed plan as the mandatory parent agent group $\omega_{p\alpha_t}$ of Π_t (Algorithm 23 which computes in polynomial time). We then create a relaxed tactical planning problem Π_r from $\omega_{p\alpha_t}$ of Π_t and Π_i to which we only add as parent agents (with corresponding goals according to each parent agent goal state constraint) the parent agents from $\omega_{p\alpha_t}$ (lines 10 to 21).

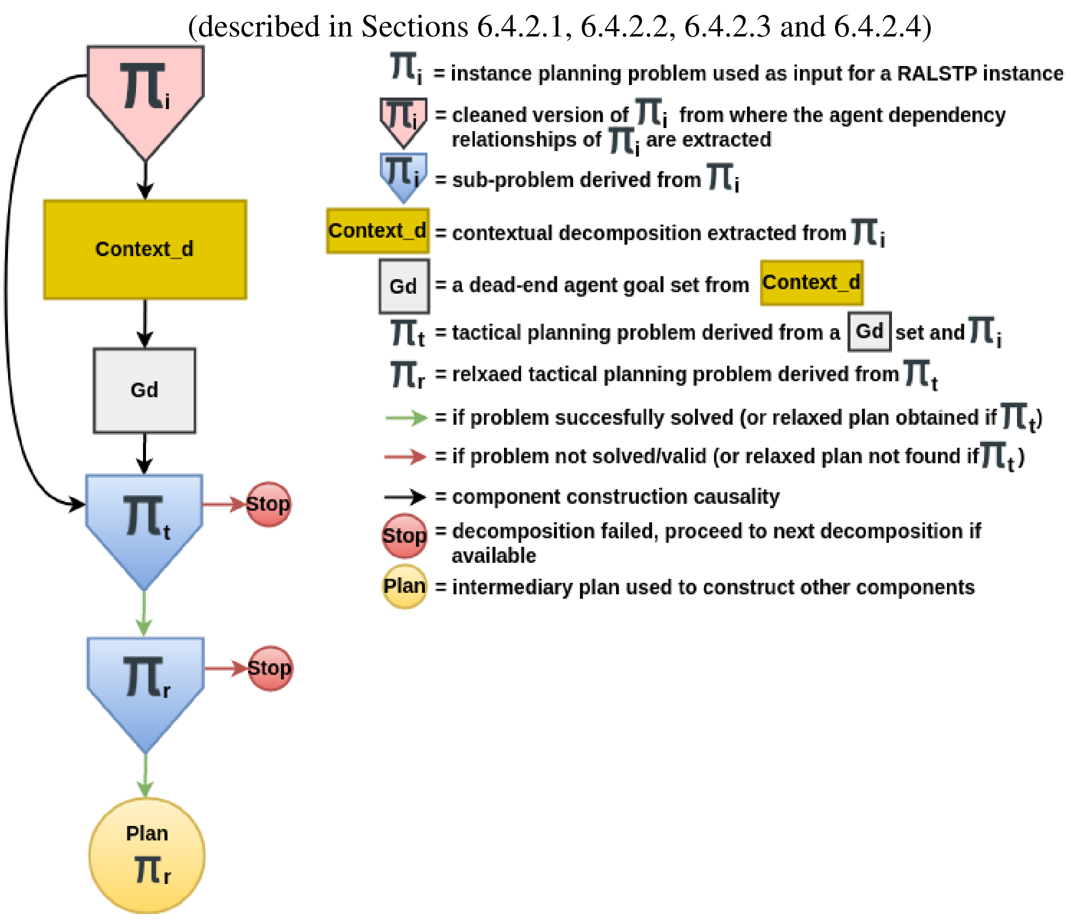

(described in Sections 6.4.2.1, 6.4.2.2, 6.4.2.3 and 6.4.2.4)
π_i
Context_d
Gd
π_t
Stop
π_r
Stop
Plan π_r
π_i = instance planning problem used as input for a RALSTP instance
π_i = cleaned version of π_i from where the agent dependency relationships of π_i are extracted
π_i = sub-problem derived from π_i
Context_d = contextual decomposition extracted from π_i
Gd = a dead-end agent goal set from Context_d
π_t = tactical planning problem derived from a Gd set and π_i
π_r = relxaed tactical planning problem derived from π_t
= if problem succesfully solved (or relaxed plan obtained if π_t)
= if problem not solved/valid (or relaxed plan not found if π_t)
= component construction causality
Stop = decomposition failed, proceed to next decomposition if available
Plan = intermediary plan used to construct other components

Fig. 6.15 Flowchart of Creating And Solving a Relaxed Tactical Planning Problem

A relaxed tactical planning problem Π_r (Figure 6.15) can be much easier to solve in comparison to the (regular) tactical planning problem it was created from, as selecting a mandatory parent agent group via the reachability heuristic prior to the solution search (lines 1 to 9) can exponentially reduce the state space during the solution search by no longer having multiple parent agents of the same type acting as entangled objects in Π_r. This allows a planner to only search for the dead-end agent goals in Π_r, which, as shown in the difficulty evaluation of the test problems in set DT at Section 5.4 from Chapter 5, is a much easier problem to solve than a problem where we also have the parent agent selection as part of the solution search (as shown in Section 7.1 from Chapter 7). This is particularly useful in problems where we have numerous entangled parent types with multiple parent agent instances for each parent type, as such problems could exponentially increase their difficulty with the addition of every new type and agent if they are solved without relaxation.

Algorithm 23 Algorithm for Creating a Relaxed Tactical Planning Problem

```
Input: $\Pi_i$, a tactical planning problem $\Pi_t$ to which the in_goal_state and available parent agent
    constraints have been added and a unique parent agent group $\mu_{p\alpha_t}$ in $\Pi_i$ (can be Null)
Output: a relaxed tactical planning problem $\Pi_r$ or error
 1: if $\mu_{p\alpha_t}$ == Null then
 2:     attempt to get the relaxed plan of $\Pi_t$
 3:     if not able to obtain the relaxed plan of $\Pi_t$ then
 4:         return error "$\Pi_t$ unsolvable with current parent agent constraints" (this error will redirect
            RALSTP to Algorithm 24, the parent agent constraints mitigation procedure)
 5:     else
 6:         $\mu_{p\alpha_t}$ = the unique parent agent group in the relaxed plan of $\Pi_t$
 7:     end if
 8: end if
 9: mark $\mu_{p\alpha_t}$ as the mandatory parent agent group $\omega_{p\alpha_t}$ of $\Pi_t$
10: create new tactical planning problem $\Pi_r := \{P,V,A,I,G\}$ where $\{P,V,A\}$ in $\Pi_i$
11: for all facts $f \in I$ when $I$ in $\Pi_i$ do
12:     if $f$ is not a parent agent fact and does not contain any parameter formed of a dead-end agent
        that is part of a dead-end agent goal outside of $G_d$ corresponding to $\Pi_t$ then
13:         add $f$ to $I$ in $\Pi_r$
14:     end if
15:     if $f$ is a parent agent fact $f_{p\alpha_t}$ and $pa$ is in $\omega_{p\alpha_t}$ of $\Pi_r$ then
16:         add $f$ to $I$ in $\Pi_r$
17:         if $in\_goal\_state(p\alpha_t)$ is true then
18:             add all $g_{p\alpha_t}$ parent agent goals in $\Pi_i$ to $G$ in $\Pi_r$
19:         end if
20:         if there are no $g_{p\alpha_t}$ parent agent goals or $in\_goal\_state(p\alpha_t)$ is false then
21:             add $f$ to $G$ in $\Pi_r$
22:         end if
23:     end if
24: end for
25: return $\Pi_r$
```

Problems Unsolvable due to Parent Agent Constraints

It can be the case that a tactical planning problem Π_t is found unsolvable during its reachability analysis [9] with the parent agents allowed by the parent agent constraints at a given time but can be solved with the parent agents that are currently restricted, for example in situations where we don't have a connected map and the only parent agent $p\alpha_t$ that could be used to solve a specific goal has its $in\_goal\_state(p\alpha_t)$ and $available(p\alpha_t)$ parent agent constraints set to false. Therefore, if Π_t is unsolvable, we use the procedure described in Algorithm 24 to re-create the tactical planning problem with incrementally more relaxed parent agent constraints and re-attempt to obtain the relaxed plan. The efficient mitigation of the parent agent constraints procedure cycles through the parent agent constraints in order to try to remove as few constraints for as few parent types as possible. The procedure first re-creates and re-attempts to solve a problem found unsolvable during its reachability analysis by ignoring the $in\_goal\_state(p\alpha_t)$ parent agent constraints of parent agents $p\alpha_t$ in Π_i for one or more parent types at a time (lines 2 to 10). The procedure gradually cycles through all possible parent type combinations until all $in\_goal\_state(p\alpha_t)$ for all $p\alpha_t$ in Π_i are ignored. If the problem is found unsolvable during its reachability analysis with all $in\_goal\_state(p\alpha_t)$ parent agent constraints ignored, we then re-create Π_t and re-attempt to get the relaxed plan while gradually ignoring the $available(p\alpha_t)$ parent agent constraints of parent agents $p\alpha_t$ in Π_i for one or more parent types at a time (lines 2 to 10). The procedure gradually cycles through all possible parent type combinations until all $available(p\alpha_t)$ for all $p\alpha_t$ in Π_i are ignored. If a problem is unsolvable with all parent agent constraints removed we return an error which specifies the problem and the contextual decomposition the problem was derived from and proceed to the next decomposition if available (lines 26 and 27). Otherwise, the procedure will use the unique parent agent group $\mu p\alpha_t$ in the relaxed plan of the re-created tactical planning problem (lines 16 to 18) as the mandatory parent agent group $\omega_{p\alpha_t}$ to be used for relaxing Π_t (by using $\omega_{p\alpha_t}$ as input in Algorithm 23).

Algorithm 24 Efficient Mitigation of the Parent Agent Constraints

```
Input: $p\alpha_t$ in $\Pi_i$, the $in\_goal\_state(p\alpha_t)$ and $available(p\alpha_t)$ of all $p\alpha_t$ in $\Pi_i$ and a tactical planning problem $\Pi_t$ to which the in_goal_state and available parent agent constraints have been added
Output: a unique parent agent group $\mu_{p\alpha_t}$ in $\Pi_i$
1: solvable($\Pi_t$) = False
2: for all $constraint(p\alpha_t) \in \{in\_goal\_state(p\alpha_t), available(p\alpha_t)\}$ do
3:     for current_no_of_types = 0; current_no_of_types < total_no_of_types; current_no_of_types++ do
4:         for start_type = 0; start_type < total_no_of_types; start_type++ do
5:             if start_type + current_no_of_types < total_no_of_types then
6:                 count = 0
7:                 for all parent types $t \in T_d$ when $T_d$ in $\Pi_t$ do
8:                     if count >= start_type and count <= start_type + current_no_of_types then
9:                         for all parent agents $p\alpha_t$ in $\Pi_t$ with type $t$ do
10:                            ignore $constraint(p\alpha_t)$
11:                            add (stp_free_t $p\alpha_t$) fact to $I$ in $\Pi_t$ unless $available(p\alpha_t)$ is not ignored and prevents it
12:                        end for
13:                    end if
14:                    count++
15:                end for
16:                solvable($\Pi_t$) = attempt to obtain the relaxed plan of $\Pi_t$
17:                if solvable($\Pi_t$ == True then
18:                    return $\mu_{p\alpha_t}$ in the relaxed plan of $\Pi_t$
19:                else
20:                    delete all (stp_free_t $p\alpha_t$) facts added at line 11 from $I$ in $\Pi_t$
21:                end if
22:            end if
23:        end for
24:    end for
25: end for
26: if solvable($\Pi_t$) == False then
27:     return error "$\Pi_t$ unsolvable $Context_d$ unsolvable" (this error will redirect RALSTP to the next decomposition if available)
28: end if
```

Each individual problem re-creation and relaxed plan extraction attempt is cost-effective as it has a run time equal to the reachability analysis of each newly created problem. However, the efficient mitigation of the parent agent constraints procedure is NP-Hard as in the worst case would exponentially increase the run time according to the number of parent types. In case of a planning problem with a large number of parent types, we can apply a weaker procedure executable in polynomial time which incrementally ignores the constraints of parent types without cycling through all possible parent type combinations. The rapid mitigation of the parent agent constraints procedure (detailed in Algorithm 25), however, is not guaranteed to remove the minimum number of constraints for as few parent agent dynamic types as possible so it might yield poorer quality solutions due to the potentially more inefficient allocation of unique parent agent groups.

Algorithm 25 Rapid Mitigation of the Parent Agent Constraints

```
Input: $p\alpha_t$ in $\Pi_i$, the $in\_goal\_state(p\alpha_t)$ and $available(p\alpha_t)$ of all $p\alpha_t$ in $\Pi_i$ and a tactical
planning problem $\Pi_t$ to which the in_goal_state and available parent agent constraints have been
added
Output: a unique parent agent group $\mu_{p\alpha_t}$ in $\Pi_i$
 1: solvable($\Pi_t$) = False
 2: for all $constraint(p\alpha_t) \in \{in\_goal\_state(p\alpha_t), available(p\alpha_t)\}$ do
 3:     for all parent types $t \in T_d$ when $T_d$ in $\Pi_t$ do
 4:         for all parent agents $p\alpha_t$ in $\Pi_t$ with type $t$ do
 5:             ignore $constraint(p\alpha_t)$
 6:             add (stp_free_t $p\alpha_t$) fact to $I$ in $\Pi_t$ unless $available(p\alpha_t)$ is not ignored and prevents it
 7:         end for
 8:         solvable($\Pi_t$) = attempt to obtain the relaxed plan of $\Pi_t$
 9:         if solvable($\Pi_t$ == True then
10:             return $\mu_{p\alpha_t}$ in the relaxed plan of $\Pi_t$
11:         end if
12:     end for
13: end for
14: if solvable($\Pi_t$) == false then
15:     return error "$\Pi_t$ unsolvable $Context_d$ unsolvable" (this error will redirect RALSTP to the
        next decomposition if available)
16: end if
```

Example 6.42. In this example, we will showcase the procedure for the rapid mitigation of the parent agent constraints (Algorithm 25). Some of the agents in the initial state of DLOG-5-5-10 are: {*driver1*, *driver2*, *driver4*} with the *driver* parent type and {*truck1 truck2*, *truck4*} with the *truck* parent type. The goal state of DLOG-5-5-10 contains parent agent goals with the *driver2*, *driver4*, *truck2* and *truck4* parent agents *(at driver2 s0), (at driver4 s1), (at truck2 s9), (at truck4 s7)* but does not contain any parent agent goals with the *driver1* and *truck1* parent agents.

At the start of the RALSTP procedure, we set the {*available(driver1)*, *available(driver2)*, *available(driver4)*,*available(truck1)*, *available(truck2)*, *available(truck4)* } parent agent constraints as true. We also set the { *in_goal_state(driver2)*, *in_goal_state(driver4)*, *in_goal_state(truck2)*, *in_goal_state(truck4)* } parent agent constraints as true, as the {*driver2*, *driver4*, *truck2*, *truck4*} parent agents are found in at least a parameter of at least one goal in the goal state of DLOG-5-5-10.

The parent agent constraints force the relaxed plan to have a unique parent agent group only from the {*driver2*, *driver4*, *truck2*, *truck4*} parent agents for solving a tactical planning problem Π_t, *{driver2, truck2}* for the purpose of this example, as even though all agents have their *available* constraints set to true, only the agents with the *in_goal_state* constraints set to true are considered for the parent types that have at least one parent agent with its *in_goal_state* constraint set to true. Therefore, the *available(driver2)*, *available(truck2)*, *in_goal_state(driver2)* and *in_goal_state(truck2)* parent agent constraints will be set to false after solving Π_t.

The updated parent agent constraints force the relaxed plan to have a unique parent agent group only from the { *driver4*, *truck4*} parent agents for solving the next tactical planning problem, $\Pi_{t'}$, as again only the agents with the *in_goal_state* constraints set to true are considered for the parent types that have at least one parent agent with its *in_goal_state* constraint set to true. If $\Pi_{t'}$ is unsolvable with only the {*driver4*,*truck4*,} parent agents, the procedure for the rapid mitigation of the parent agent constraints will first ignore all the $in\_goal\_state(driver)$ constraints and allow the relaxed plan to have a unique parent agent group from the { *driver1*, *driver4*, *truck4*} parent agents, as even though $in\_goal\_state(driver4)$ is set to false, the $in\_goal\_state(driver)$ constraints are ignored and only the $available(driver)$ and $in\_goal\_state(truck)$ constraints are considered.

If $\Pi_{t'}$ is unsolvable with the { *driver1*, *driver4*, *truck4*} parent agents, the procedure for the rapid mitigation of the parent agent constraints will then also ignore all the $in\_goal\_state(truck)$ constraints and allow the relaxed plan to have a unique parent agent group from the { *driver1*, *driver4*, *truck1*, *truck4*} parent agents, as all the *in_goal_state* constraints are ignored and only the *available* constraints are considered.

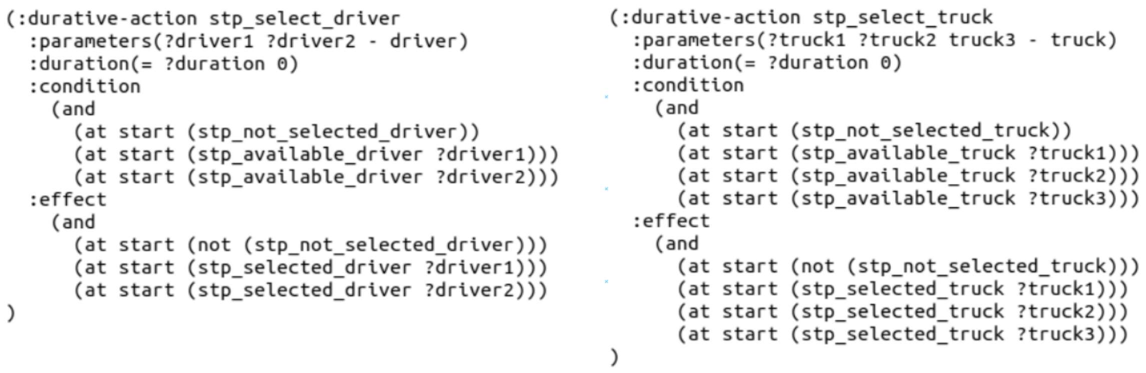
If $\Pi_{t'}$ is unsolvable with the { *driver1*, *driver4*, *truck1*, *truck4*} parent agents, the procedure for the rapid mitigation of the parent agent constraints will then also ignore the $available(driver)$ constraints and allow the relaxed plan to have a unique parent agent group from the { *driver1*, *driver2*, *driver4*, *truck1*, *truck4*} parent agents, as even though $available(driver2)$ is set to false due to having been used in solving Π_t, the $available(driver)$ are ignored and only the $available(truck)$ constraints are taken into account.

If $\Pi_{t'}$ is unsolvable with the { *driver1*, *driver2*, *driver4*, *truck1*, *truck4*} parent agents, the procedure for the rapid mitigation of the parent agent constraints will then also ignore the $available(truck)$ constraints and allow the relaxed plan to have a unique parent agent group from all the parent agents (*driver1*, *driver2*, *driver4*, *truck1*, *truck2*, *truck4*), without taking into account any of the parent agent constraints.

If $\Pi_{t'}$ is unsolvable without taking into account any of the parent agent constraints, the procedure will exit with an error reporting the unsolved problem and the contextual decomposition it was derived from. Otherwise, the procedure will return the unique parent agent group $\mu p\alpha_t$ in the relaxed plan for $\Pi_{t'}$ as the mandatory parent agent group $\omega_{p\alpha_t}$ to be used for relaxing Π_t.

Problems that require Multi-Agents with the Same Parent Type

Problems that are designed with a mandatory requirement of more than one parent agent with the same parent type for achieving the goal state are immediately detected as unsolvable during a reachability analysis due to our one parent agent per parent type constraints. For example, a problem that requires two *fire_truck* parameters for executing a fire extinguishing action will prevent the reachability analysis from arriving at the goal state if only one fire truck can be selected by the planner. In such a situation, we re-create a tactical planning problem and gradually increase the number of parameters of each *stp_select_parent_dynamic_type* action to allow multiple parent agents of the same type to be selectable by a solver during the solution search (Figure 6.16). This is achieved by adding an extra *(stp_selected_parent_dynamic_type ?parent_dynamic_type - parent_dynamic_type)* positive effect to $eff_{stp\_select\_parent\_dynamic\_type}$ for each parameter added to the *stp_select_parent_dynamic_type* action for the parent type the action represents as well as for each of the supertypes of the parent type the action represents provided that the supertypes are also dynamic types. The low cost of performing multiple reachability analyses allows for a rapid identification of the necessary number of parent agents either through a polynomial or an exponential search similar to the ones described in the parent agent constraints mitigation procedures.



```
(:durative-action stp_select_driver
  :parameters(?driver1 ?driver2 - driver)
  :duration(= ?duration 0)
  :condition
    (and
      (at start (stp_not_selected_driver))
      (at start (stp_available_driver ?driver1)))
      (at start (stp_available_driver ?driver2)))
  :effect
    (and
      (at start (not (stp_not_selected_driver)))
      (at start (stp_selected_driver ?driver1)))
      (at start (stp_selected_driver ?driver2)))
)
```

```
(:durative-action stp_select_truck
  :parameters(?truck1 ?truck2 truck3 - truck)
  :duration(= ?duration 0)
  :condition
    (and
      (at start (stp_not_selected_truck))
      (at start (stp_available_truck ?truck1)))
      (at start (stp_available_truck ?truck2)))
      (at start (stp_available_truck ?truck3)))
  :effect
    (and
      (at start (not (stp_not_selected_truck)))
      (at start (stp_selected_truck ?truck1)))
      (at start (stp_selected_truck ?truck2)))
      (at start (stp_selected_truck ?truck3)))
)
```

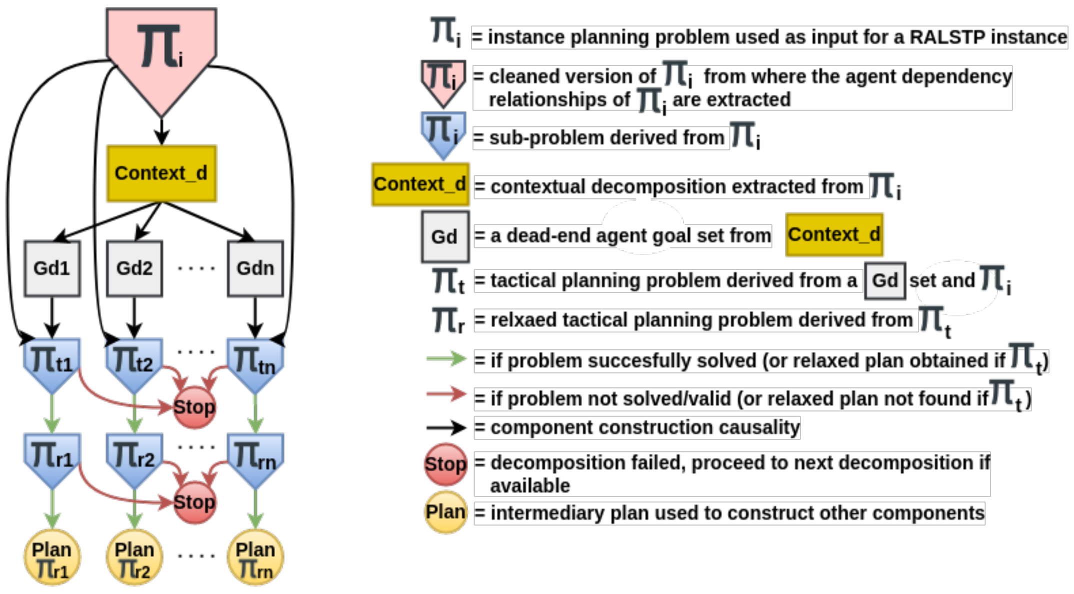
Fig. 6.16 The *stp_select_driver* and *stp_select_truck* actions implemented for supporting multiple agents with the same parent type from the *stp_select_parent_dynamic_type* template and the *{driver, truck}* parent types in the Driverlog Domain

6.4.2.6 Solving the Relaxed Tactical Planning Problems of a Contextual Decomposition

Section Overview In this section, we describe the procedure for solving the relaxed tactical planning problems from a contextual decomposition extracted from a planning problem.

The complete procedure for creating and solving the relaxed tactical planning problems of a contextual decomposition is described in Algorithm 26 (which has a time complexity based on the chosen solver). The procedure starts by setting the $available(p\alpha_t)$ constraints for all parent agents $p\alpha_t$ in Π_i to true and by setting the $in\_goal\_state(p\alpha_t)$ for all parent agents that are part of the $g_{p\alpha_t} \in G$ goals when G in Π_i to true (lines 1 to 3).

The relaxed tactical planning problems are created and solved one after the other to utilise all available computing resources on one problem at a time (as shown in Figure 6.17). Therefore, temporal expressiveness and numeric components are permitted within a relaxed temporal planning problem, as each problem is solved as a stand-alone numeric temporal planning problem using an off-the-shelf temporally expressive numeric planner. Any concurrency violations between two or more relaxed tactical planning problems will be addressed at the strategic level (described in Section 6.4.3).



Πi
Context_d
Gd1
Gd2
Gdn
Πt1
Πt2
Πtn
Stop
Πr1
Πr2
Πrn
Stop
Plan Πr1
Plan Πr2
Plan Πrn
Πi = instance planning problem used as input for a RALSTP instance
Πi = cleaned version of Πi from where the agent dependency relationships of Πi are extracted
Πi = sub-problem derived from Πi
Context_d = contextual decomposition extracted from Πi
Gd = a dead-end agent goal set from Context_d
Πt = tactical planning problem derived from a Gd set and Πi
Πr = relxaed tactical planning problem derived from Πt
= if problem succesfully solved (or relaxed plan obtained if Πt)
= if problem not solved/valid (or relaxed plan not found if Πt)
= component construction causality
Stop = decomposition failed, proceed to next decomposition if available
Plan = intermediary plan used to construct other components

Fig. 6.17 Flowchart for Solving All Relaxed Tactical Planning Problems of a Contextual Decomposition

If even one relaxed tactical planning problem from a contextual decomposition can't be solved or is solved but invalid, then the whole contextual decomposition is regarded as unsolvable and abandoned (lines 22 to 24). Therefore, we create and solve the relaxed tactical planning problems starting from the dead-end agent goal set $G_d \in Context_d$ with the most number of goals to the goal set $G_{d'} \in Context_d$ with the least number of goals (line 5). This is done to attempt to solve the more difficult relaxed tactical planning problems (the ones that have the highest total number of dead-end agents $N(d\alpha)$) before the easier relaxed tactical planning problems (the ones that have the lowest total number of dead-end agents $N(d\alpha)$) derived from $Context_d$ in order to force a fast failure which will potentially save time by not solving the easier problems first.

After a relaxed tactical planning problem Π_r is solved, we validate the obtained plan with VAL [41] and end the procedure for the current contextual decomposition if the plan for Π_r is invalid (line 22). If the plan for Π_r is valid, we set to false the parent agent constraints of all parent agents $p\alpha_t$ from the mandatory parent agent group $\omega_{p\alpha_t}$ used to create Π_r before creating and solving the next relaxed tactical planning problem $\Pi_{t'}$ for the next dead-end agent goal set $G'_d \in Context_d$ (lines 11 to 15). If all $available(p\alpha_t)$ parent agent constraints for all parent agents $p\alpha_t$ that have the same parent type t become false after a constraint update, we reset the $available(p\alpha_t)$ constraint for all parent agents with parent type t to true before creating the next relaxed tactical planning problem (lines 17 to 19).

The procedure for solving the relaxed tactical planning problems of a contextual decomposition is successful only if all relaxed tactical planning problems derived from the contextual decomposition are solved and all resulting plans are valid. If successful, the procedure will output the initial state, final state and plan of each relaxed tactical planning problem derived from the contextual decomposition. The same procedure is applicable for regular (non-relaxed) tactical planning problems (line 27).

Algorithm 26 Algorithm for Solving All Relaxed Tactical Planning Problems derived from a $Context_d$ in Π_i

```
Input: all $p\alpha_t$ and a $Context_d$ from $\Sigma$ in $\Pi_i$
Output: initial states, final states and plans of all $\Pi_r$ derived from $Context_d$ or error
 1: for all parent agents $p\alpha_t$ in $\Pi_i$ do
 2:     set all $available(p\alpha_t)$ parent agent constraints to true
 3:     set $in\_goal\_state(p\alpha_t)$ parent agent constraints to true (as described in Definition 6.38
 4: end for
 5: sort dead-end agent goal sets $G_d \in Context_d$ from largest to smallest
 6: for all $G_d \in Context_d$ do
 7:     $\Pi_t$ = run Algorithm 22 on $G_d$
 8:     $\Pi_r$ = run Algorithm 23 on $\Pi_t$
 9:     solved($\Pi_r$) = attempt to solve $\Pi_r$
10:     if solved($\Pi_r$) == true then
11:         if plan of solved($\Pi_r$) is valid then
12:             store initial state, final state and plan for $\Pi_r$
13:             for all parent agents $p\alpha_t$ in $\omega_{p\alpha_t}$ when $\omega_{p\alpha_t} \in \Pi_r$) do
14:                 set $available(p\alpha_t)$ to false
15:                 set $in\_goal\_state(p\alpha_t)$ to false
16:             end for
17:             for all parent types $t \in T_d$ do
18:                 if all $p\alpha_t \in t$ have $available(p\alpha_t)$ == false then
19:                     set $available(p\alpha_t)$ to true for all $p\alpha_t \in t$
20:                 end if
21:             end for
22:         else
                return error "plan of solved($\Pi_r$) derived from $Context_d$ is invalid" (this error will
                redirect RALSTP to the next decomposition if available)
23:         end if
24:     else
            return error "$\Pi_r$ of $Context_d$ is unsolvable" (this error will redirect RALSTP to the next
            decomposition if available)
25:     end if
26: end for
27: return stored initial states, final states and plans of all $\Pi_r$ derived from $Context_d$
```

6.4.3 Strategic Abstraction for Mitigating Concurrency Violations in Temporal Planning

Section Overview In this section, we present the strategic abstraction technique for mitigating the constraint violations among tactical plans using the duration of durative actions. This is achieved by creating a strategic problem which contains the abstracted tactical plans from a contextual decomposition $Context_d$ in the form of durative macro actions. The solution to the strategic problem provides ordering instructions and potential additional durative actions to efficiently link all durative actions in all tactical plans from $Context_d$ in a way that not only eliminates all constraint violations among conflicting tactical plans but also maintains any temporal expressiveness and numeric computation that may have occurred in the tactical plans. We will first provide a higher-level description of the strategic abstraction technique, which will be followed by a more detailed description of each component used in the technique.

The plans of the relaxed tactical planning problems Π_r (described in Section 6.4.2) in a solved contextual decomposition (described in Section 6.4.1) cumulatively achieve all the dead-end agent goals of the instance planning problem. However, if we merge the plans of the relaxed tactical planning problems into a master plan that achieves all the dead-end agent goals, the master plan might be invalid due to concurrency violations among its actions. This happens because, even though we use parent agent constraints to attempt to spread the parent agents efficiently among all relaxed tactical planning problems, it can still be the case that some parent agents are found in more than one relaxed tactical planning problem plan. For example, concurrency issues be encountered if the number of relaxed tactical planning problems is higher than the maximum quantity of unique parent agent groups. In this case, some or all *available* parent agent constraints will be reset to true throughout the solving of the relaxed tactical planning problems so one or more parent agents will be found in more than one relaxed tactical planning problem plan.

Our procedure for mitigating potential concurrency violations (detailed in Section 6.4.3.2) relies on the start time and duration of durative actions and uses a planner to determine the merging strategy for the tactical plans. This is achieved by encapsulating the initial states and final states of the relaxed tactical planning problems of a solved contextual decomposition into mutually exclusive grounded durative macro actions (detailed in Section 6.4.3.1). The durative macro actions are combined with the rest of the operators in the corresponding instance planning problem to form the strategic domain. The mutual exclusion of all durative actions and durative macro actions in the strategic domain is achieved by adding preconditions and effects that prevent each parent agent $p\alpha_t$ in the strategic problem from being used as a parameter in more than one action at the same time.

The durative macro actions and goal state of a strategic problem are designed to force a planner to execute all durative macro actions in the strategic problem in order to achieve its goal state. If the strategic problem is solved, we use the start time of the mutually exclusive durative macro actions in the plan of the solved strategic problem to adjust the start times of the actions in the tactical plans. Then, we merge the updated tactical plans into a single plan and add all regular (non-macro) durative actions in the plan of the solved strategic planning problem to the merged plan. The end result is a valid plan with no concurrency violations that achieves all the dead-end agent goals in the instance planning problem. The strategic level permits temporal expressiveness and numeric interaction between durative actions and durative macro actions, as the strategic problem is solved as a stand-alone numeric temporal planning problem using an off-the-shelf temporally expressive numeric planner.

6.4.3.1 Creating and Solving a Strategic Planning Problem

Section Overview In this section, we present a detailed description for creating and solving a strategic planning problems corresponding to a contextual decomposition extracted.

Definition 6.43. A *strategic planning problem* is defined as a tuple $\Pi_s := \{P, V, A, I, G\}$ where $\{P, V, A, I\}$ in Π_s are initially inherited from the instance planning problem Π_i from which Π_s is derived. $\{P, A, I\}$ in Π_s have been modified to prevent each parent agent $p\alpha_t$ in Π_s from being used as a parameter in more than one action at the same time. Π_s does not contain any dead-end agents or dead-end agent facts. A in Π_s is supplemented with a durative macro action $a_{(\Pi_r)}$ for each solved relaxed tactical planning problem Π_r derived from a contextual decomposition $Context_d$. G in Π_s contains none of the initial goals from Π_i. Instead, G in Π_s contains one goal g for each durative macro action $a_{(\Pi_r)}$ added in A in Π_s where g represents a positive effect achievable only by the completion of its corresponding durative macro action. All object instances in Π_s are defined as constants. Π_s can be created only from a contextual decomposition $Context_d$ that had all its corresponding relaxed tactical planning problems solved.

Specification 6.44. The *stp_mission* type is created and added to T in Π_s and is used to create and add the *(stp_complete_mission ?stp_mission - stp_mission)* proposition to P in Π_s. A goal $g \in G$ when G in Π_s is constructed from the *(stp_complete_mission ?stp_mission - stp_mission)* proposition for each solved relaxed tactical planning problem Π_r derived from the contextual decomposition from which Π_s is constructed.

Specification 6.45. The *(stp_free_parent_dynamic_type ?dynamic_type)* template is used to create and add a new proposition to P in Π_s for each dynamic type t in Π_i that represents the type or supertype (provided the supertype is a dynamic type) of a parent agent $p\alpha_t$ in Π_i. A *(stp_free_parent_dynamic_type ?dynamic_type)* fact is created and added to I in Π_s for each parent agent $p\alpha_t$ in Π_i. Each proposition and corresponding fact will be constructed according to the parent type they represent.

Definition 6.46. A durative *macro action* $a_{(\Pi_r)} := \{pre_{a(\Pi_r)}, \mathit{eff}_{a(\Pi_r)}\ dur_{a(\Pi_r)}\}$ corresponding to a solved relaxed tactical planning problem Π_r is a grounded durative action where duration $dur_{a(\Pi_r)}$ is the cost of the plan for Π_r. $pre_{\vdash a_{(\Pi_r)}}$ contains as positive conditions each fact $f \in I$ when I in Π_r if f is not a dead-end agent fact and if f had its truth value changed in the plan for Π_r. $pre_{\vdash a_{(\Pi_r)}}$ also contains a *(stp_free_parent_dynamic_type constant)* positive condition for each parent agent (defined as a constant) $p\alpha_t$ in Π_s that is a parameter in any of the f facts previously added as positive conditions to $pre_{\vdash a_{(\Pi_r)}}$. $\mathit{eff}_{\dashv a_{(\Pi_r)}}$ contains as positive effects each positive fact f from the final state of Π_r if f is not a dead-end agent fact and if f had its truth value changed in the plan for Π_r. $\mathit{eff}_{\dashv a_{(\Pi_r)}}$ contains as negative effects each fact $f \in I$ when I in Π_r that was added as a positive condition to $pre_{\vdash a_{(\Pi_r)}}$ if f is negative in the final state of Π_r. $\mathit{eff}_{\vdash a_{(\Pi_r)}}$ contains a *(not (stp_free_parent_dynamic_type constant))* negative effect for each *(stp_free_parent_dynamic_type constant)* positive condition added to $pre_{\vdash a_{(\Pi_r)}}$. $\mathit{eff}_{\dashv a_{(\Pi_r)}}$ contains an *(stp_free_parent_dynamic_type constant)* positive effect for each *(not (stp_free-_parent_dynamic_type constant))* added to $\mathit{eff}_{\vdash a_{(\Pi_r)}}$.

$\mathit{eff}_{\dashv a_{(\Pi_r)}}$ also contains an *(stp_complete_mission stp_mission_constant)* positive effect equivalent to the goal in G in Π_s added for the same solved relaxed tactical planning problem Π_r from which $a_{(\Pi_r)}$ is created.

Specification 6.47. A *(stp_free_parent_dynamic_type ?parent_dynamic_type - parent_dynamic_type)* positive condition is created and added to the precondition $pre_{\vdash a}$ of all regular (non-macro) durative actions $a \in A$ when A in Π_s for every parameter in a that is a parent agent. All regular (non-macro) durative actions $a \in A$ when A in Π_s will also get added a *(not (stp_free_parent_dynamic_type ?parent_dynamic_type - parent_dynamic_type))* negative effect to $\mathit{eff}_{\vdash a}$ for every parameter in a that is a parent agent and a *(stp_free_parent_dynamic_type ?parent_dynamic_type - parent_dynamic_type)* positive effect to $\mathit{eff}_{\dashv a}$ for every parameter in a that is a parent agent.

The procedure for creating and solving a strategic problem (Figure 6.18) is described in Algorithm 27, which has a time complexity based on the chosen solver (the creation of Π_s is computed in polynomial time). A strategic planning problem is derived from a contextual decomposition $Context_d$ (lines 1 to 6) only if all the relaxed tactical planning problems Π_r derived from $Context_d$ have been successfully solved. Π_s derived from $Context_d$ is initially created from $\{P,V,A,I\}$ in Π_i and has all objects defined as constants in order to allow the incorporation of facts from the initial and final states of the relaxed tactical planning problems Π_r derived from $Context_d$ as conditions and effects of the durative macro actions in Π_s. A strategic planning problem is stripped of all dead-end agents and dead-end agent facts to reduce its difficulty, as the dead-end agent goals were processed at the tactical level. Strategic planning problems have no inactive dynamic objects, as each Π_s was derived from the instance planning problem Π_i which had all its inactive dynamic objects removed when it was cleaned.

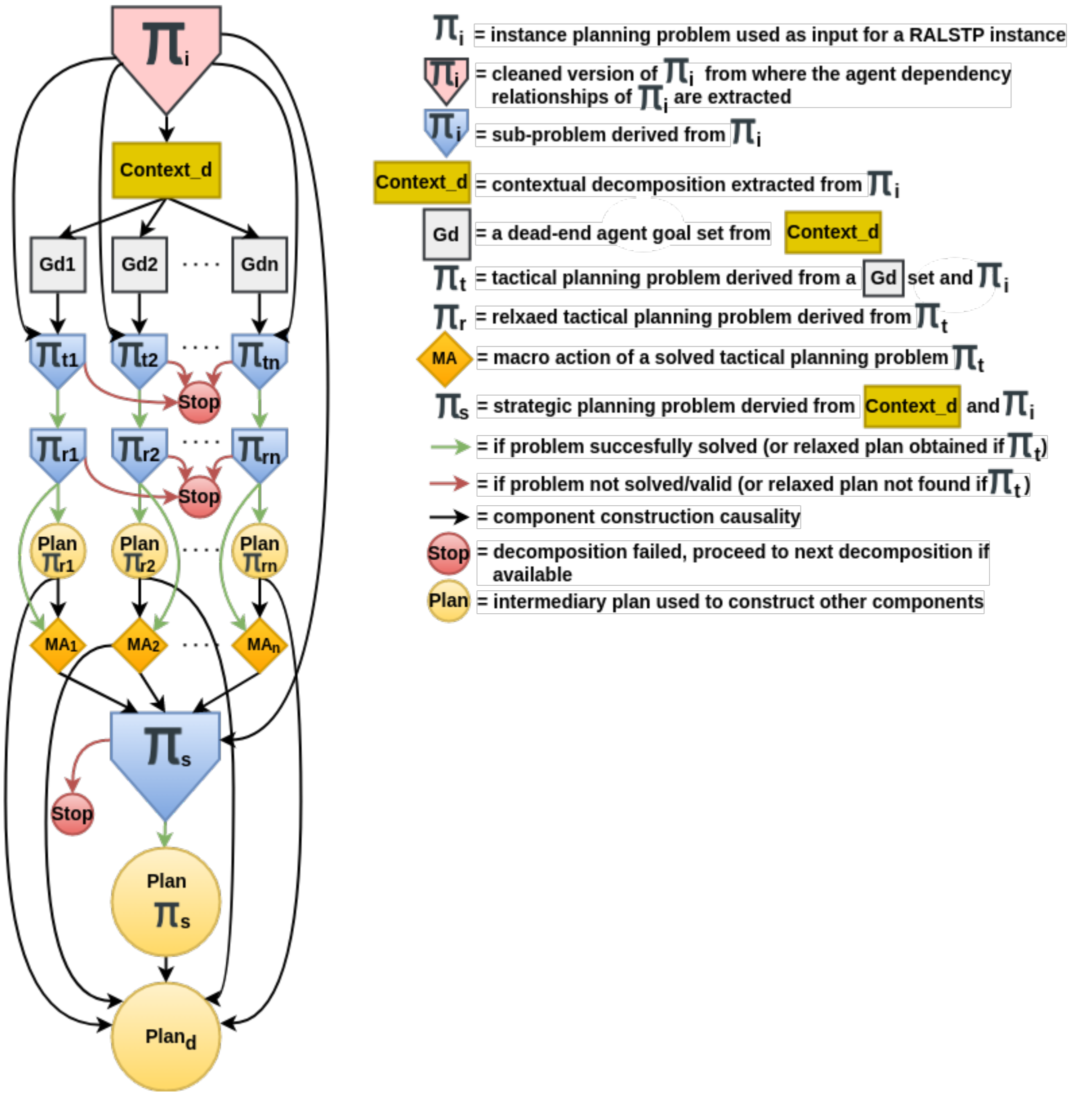

Πi
Context_d
Gd1
Gd2
Gdn
Πt1
Πt2
Πtn
Stop
Πr1
Πr2
Πrn
Plan Πr1
Plan Πr2
Plan Πrn
MA1
MA2
MAn
Πs
Plan Πs
Pland
Πi = instance planning problem used as input for a RALSTP instance
Πi = cleaned version of Πi from where the agent dependency relationships of Πi are extracted
Πi = sub-problem derived from Πi
Context_d = contextual decomposition extracted from Πi
Gd = a dead-end agent goal set from Context_d
Πt = tactical planning problem derived from a Gd set and Πi
Πr = relxaed tactical planning problem derived from Πt
MA = macro action of a solved tactical planning problem Πt
Πs = strategic planning problem dervied from Context_d and Πi
= if problem succesfully solved (or relaxed plan obtained if Πt)
= if problem not solved/valid (or relaxed plan not found if Πt)
= component construction causality
Stop = decomposition failed, proceed to next decomposition if available
Plan = intermediary plan used to construct other components

Fig. 6.18 Flowchart for Creating and Solving a Strategic Planning Problem

A grounded durative macro action $a_{(\Pi_r)}$ (Figure 6.19) is added to the action set A in Π_s for each relaxed tactical planning problem Π_r derived from $Context_d$ (line 11). Each durative macro action $a_{(\Pi_r)}$ will have the facts in the initial state of Π_r that are not dead-end agent facts and had their truth value change in the plan for Π_r as positive conditions in $pre_{\vdash a_{(\Pi_r)}}$, the facts in the final state of Π_r that are not dead-end agent facts and had their truth value changed in the plan for Π_r as positive effects in $\mathit{eff}_{\dashv a}$, the facts in the initial state of Π_r added as positive conditions in $\mathit{pre}_{\vdash a_{(\Pi_r)}}$ which are negative in the final state of Π_r as negative effects in $\mathit{eff}_{\dashv a}$ and the makespan of the plan for Π_r from which $a_{(\Pi_r)}$ is constructed as $\mathit{dur}_{a_{(\Pi_r)}}$. For each durative macro action $a_{(\Pi_r)}$ in Π_s we add a constant $\mathit{stp\_mission\_constant}$ with the $\mathit{stp\_mission}$ type to Π_s (line 9), a *(stp_complete_mission stp_mission_constant)* effect to $\mathit{eff}_{\dashv a(\Pi_r)}$ and a *(stp_complete_mission stp_mission_constant)* goal to the goal state of Π_s in order to force a planner to execute all durative macro actions in order to achieve the goal state of Π_s (line 10).

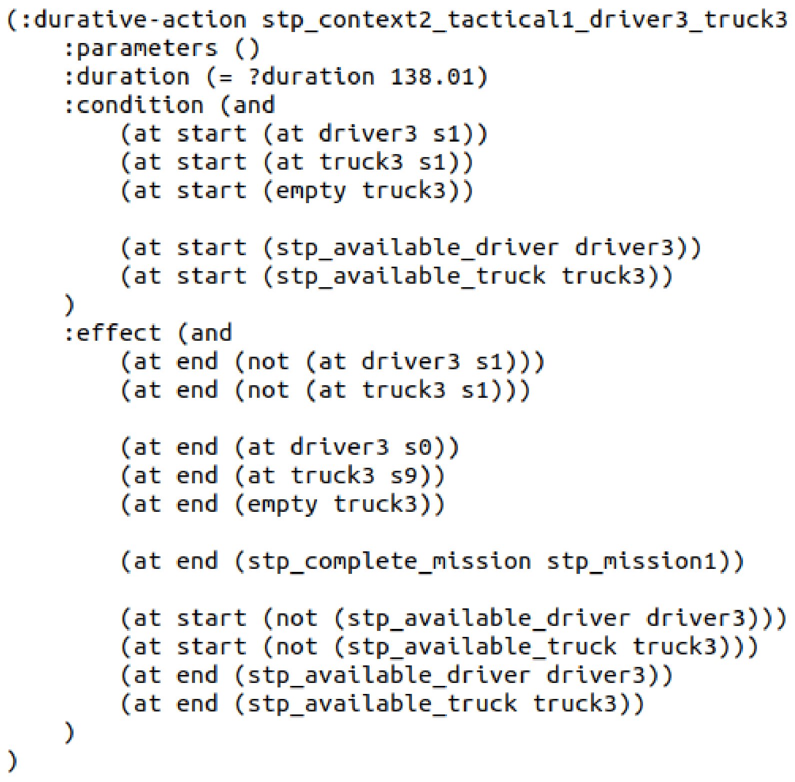
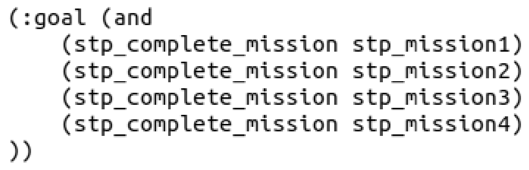

```
(:durative-action stp_context2_tactical1_driver3_truck3
    :parameters ()
    :duration (= ?duration 138.01)
    :condition (and
        (at start (at driver3 s1))
        (at start (at truck3 s1))
        (at start (empty truck3))

        (at start (stp_available_driver driver3))
        (at start (stp_available_truck truck3))
    )
    :effect (and
        (at end (not (at driver3 s1)))
        (at end (not (at truck3 s1)))

        (at end (at driver3 s0))
        (at end (at truck3 s9))
        (at end (empty truck3))

        (at end (stp_complete_mission stp_mission1))

        (at start (not (stp_available_driver driver3)))
        (at start (not (stp_available_truck truck3)))
        (at end (stp_available_driver driver3))
        (at end (stp_available_truck truck3))
    )
)
```

Fig. 6.19 DLOG-5-5-10 Durative Macro Action

Each durative macro action $a_{(\Pi_r)}$ in Π_s will also have a *(stp_free_parent_dynamic_type constant)* positive condition in $\mathit{pre}_{\vdash a_{(\Pi_r)}}$, an *(not (stp_free_parent_dynamic_type constant))* negative effect in $\mathit{eff}_{\vdash a_{(\Pi_r)}}$ and an *(stp_free_parent_dynamic_type constant)* positive effect in $\mathit{eff}_{\dashv a_{(\Pi_r)}}$ for each parent agent (defined as a constant) $p\alpha_t$ in Π_s that is a parameter in any of the parent agent facts $f_{p\alpha_t}$ encapsulated in $a_{(\Pi_r)}$ in order to enforce mutual exclusion among all durative macro actions in Π_s. Each regular (non-macro) durative action $a \in A$ when A in Π_s will have a *(stp_free_parent_dynamic_type ?parent_dynamic_type - parent_dynamic_type)* positive condition in $\mathit{pre}_{\vdash a}$, an *(not (stp_free_parent_dynamic_type*

?parent_dynamic_type - parent_dynamic_type)) negative effect in $\mathit{eff}_{\vdash a}$ and an *(stp_available_parent_dynamic_type ?parent_dynamic_type - parent_dynamic_type)* positive effect in $\mathit{eff}_{\dashv a}$ which combined with the mutual exclusion conditions and effects added to each durative macro action $a_{(\Pi_r)}$ in Π_s enforce mutual exclusion among all durative actions that have parent agent parameters in Π_s.



```
(:goal (and
    (stp_complete_mission stp_mission1)
    (stp_complete_mission stp_mission2)
    (stp_complete_mission stp_mission3)
    (stp_complete_mission stp_mission4)
))
```

Fig. 6.20 Example of a Strategic Planning Problem Goal State

The goal state G in Π_s (Figure 6.20) has no other goals apart from the *(stp_complete_mission stp_mission_constant)* goals (line 10). The only way to achieve the goal state is to execute all durative macro actions, as only they have *(stp_complete_mission stp_mission_constant)* as $\mathit{eff}_{\dashv}$ positive effects. Executing all durative macro actions guarantees that all dead-end agent goals in Π_i are also achieved, as all dead-end agent goals in Π_i are defined as positive effects $\mathit{eff}_{\dashv}$ in the durative macro actions of Π_s. Therefore, the state space of Π_s is greatly reduced in comparison to the state space of its corresponding all dead-end agent goals planning problem Π_d, as Π_s uses only grounded actions for achieving all dead-end agent goals in comparison to Π_d which must use lifted actions for achieving all dead-end agent goals. The regular (non-macro) durative actions in Π_s have a very small impact on the state space of Π_s in comparison to the impact of the equivalent actions in Π_d on the state space of Π_d. This happens because, in Π_s, the regular (non-macro) durative actions can only be used to achieve states where the durative macro actions can be executed without concurrency violations in comparison to Π_d where the regular durative actions are used for achieving the actual goal state of Π_d (as shown in Example 6.48).

After a solution is found to a particular Π_s, we run the constraints mitigation procedure on the solution and tactical plans corresponding to Π_s to obtain the all dead-end agent goals plan $plan_d$ corresponding to Π_s (lines 14 to 15, detailed Section 6.4.3.2). Afterwards, we proceed to the parent agent goals solving procedure (described in section 6.5) to which we will provide $plan_d$ for Π_s as input (lines 16 to 17). Each Π_s is solved as a stand-alone numeric temporal planning problem using an off-the-shelf temporally expressive numeric planner. Therefore, each Π_s can involve temporal expressiveness and numeric components.

Algorithm 27 Algorithm for Creating and Solving a Strategic Planning Problem

```
Input: $\Pi_i$, a $Context_d$ from $\Sigma$ that had all its corresponding $\Pi_r$ solved as well as the initial states, final states and plans of all $\Pi_r$ derived from $Context_d$
Output: $plan_d$ and the final state of solved($\Pi_s$) for each solved $\Pi_s$ or error
 1: create new strategic planning problem $\Pi_s := \{P,V,A,I,G\}$ where $\{P,V,A,I\}$ in $\Pi_i$
 2: define all objects in $\Pi_s$ as constants
 3: add the stp_mission type and the (stp_complete_mission ?stp_mission - stp_mission) proposition to $P$ in $\Pi_s$ (as described in Specification 6.44)
 4: add the (stp_free_parent_dynamic_type ?parent_dynamic_type - parent_dynamic_type) propositions to $P$ in $\Pi_r$ (as described in Specification 6.45)
 5: add the (stp_free_parent_dynamic_type ?parent_dynamic_type - parent_dynamic_type) positive conditions to every regular (non-macro) durative action $a \in A$ when $A$ in $\Pi_s$ (as described in Specification 6.47)
 6: add the (not (stp_free_parent_dynamic_type ?parent_dynamic_type - parent_dynamic_type)) effect to every regular (non-macro) durative action $a \in A$ when $A$ in $\Pi_s$ (as described in Specification 6.47)
 7: for all $G_d \in Context_d$) do
 8:     index($G_d$) = position of set $G_d$ in $Context_d$
 9:     add stp_mission_index($G_d$) - stp_mission constant to $\Pi_s$
10:     add the (stp_complete_mission stp_mission_index($G_d$)) goal to $G$ in $\Pi_s$
11:     add durative macro action $a_{(\Pi_r)}$ to $A$ in $\Pi_s$ using the initial state, final state and plan of $\Pi_r$ corresponding to $G_d$ (as described in Definition 6.46)
12: end for
13: solved($\Pi_s$) = attempt to solve $\Pi_s$
14: if solved($\Pi_s$) then
15:     all dead-end agent goals plan $plan_d$ = run the constraints mitigation procedure (Algorithm 28 described in Section 6.4.3.2) on the plan of solved($\Pi_s$) and on the plans of all $\Pi_r$ derived from $Context_d$
16:     if $plan_d$ is valid then
17:         return $plan_d$)
18:     else
19:         return error "$plan_d$ of $Context_d$ is invalid" (this error will redirect RALSTP to the next decomposition if available)
20:     end if
21: else
        return error "$\Pi_s$ of $Context_d$ is unsolvable (this error will redirect RALSTP to the next decomposition if available)
22: end if
```

6.4.3.2 Mitigating Constraint Violations Using Temporal Data

Section Overview In this section, we describe and provide an example of how solving a strategic problem mitigates the eventual constraint violations among the tactical plans obtained from a contextual decomposition.

In general, decomposing a planning problem presents the risk of obtaining constraint violations among the sub-problems [15]. This is also the case for the tactical planning problems of a contextual decomposition $Context_d$. For example, two or more tactical planning problems can use the same parent agents to solve their goal. In such a scenario, we could have constraint violations among the obtained plans due to the shared (parent agent) resources among them.

Our constraints mitigation technique (Algorithm 28 which computes in polynomial time) follows the description in Buksz et al. (2018) in order to resolve all potential constraints among all tactical planning problems derived from a contextual decomposition $Context_d$. After solving a strategic planning problem Π_s derived from a contextual decomposition $Context_d$, we create the all dead-end agent goals plan $plan_d$ for Π_s by replacing each durative macro action $a_{(\Pi_r)}$ in the plan for Π_s with the durative actions a' in the plans of the relaxed tactical planning problem Π_r encapsulated by $a_{(\Pi_r)}$ and adding them to $plan_d$ (lines 1 to 6). The start time of each action a' in the plan of a relaxed tactical planning problem Π_r added to $plan_d$ will be updated by adding the start time of the durative macro action $a_{(\Pi_r)}$ which encapsulated Π_r in order to eliminate all potential concurrency violations between any of the a' actions added to $plan_d$ (line 5).

The regular (non-macro) durative actions a in the plan for Π_s (if such actions exist) are added without modifying their start time (line 9), as their start time has been specifically set by the solver during the solution search of Π_s to allow actions a to mitigate eventual constraints violations between any of the a' actions added to $plan_d$.

The above modifications allow all concurrencies that take place locally within each Π_r or globally in Π_s to be maintained in $plan_d$. If $plan_d$ is valid, we proceed to solve the parent agent goals by providing as input for Algorithm 29 the all dead-end agent goals plan $plan_d$ for Π_s derived from $Context_d$ (lines 12 and 13, detailed in Section 6.5). If $plan_d$ is found invalid by VAL [41], we stop and return an invalid error for $plan_d$ and the contextual decomposition $Context_d$ from which Π_s was derived and continue the procedure with the next decompositions if available (lines 14 and 15).

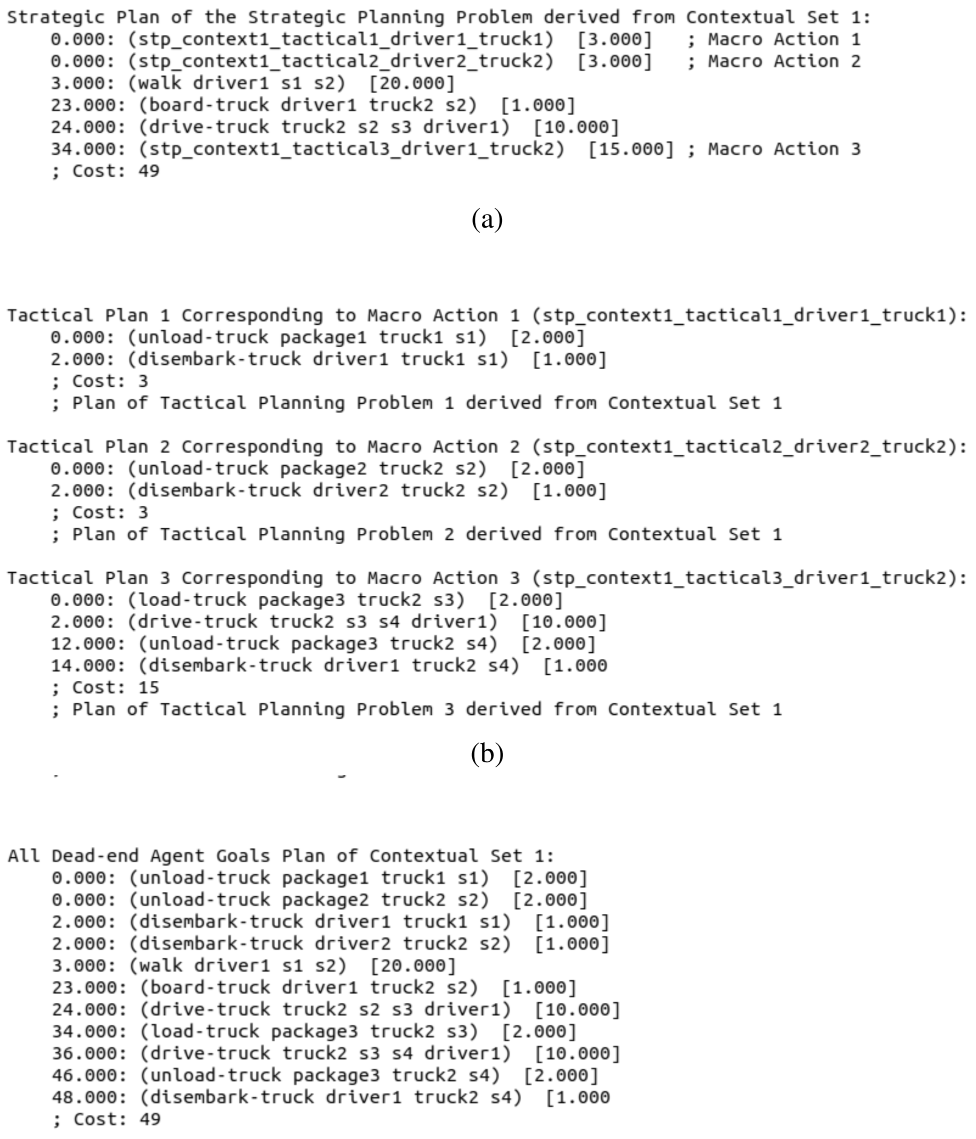
Algorithm 28 Algorithm for Mitigating Constraint Violations

```
Input: the plan of a solved($\Pi_s$) and the plans of all $\Pi_r$ corresponding to $\Pi_s$
Output: all dead-end agent goals plan $plan_d$ or error
 1: all dead-end agent goals plan $plan_d$ = empty
 2: for all actions $a$ in the plan of solved($\Pi_s$) do
 3:     if $a$ == durative macro action then
 4:         for all actions $a'$ in the plan of $\Pi_r$ corresponding to durative macro action $a$ do
 5:             start_time($a'$) = start_time($a'$) + start_time($a$)
 6:             add action $a'$ to all dead-end agent goals plan $plan_d$
 7:         end for
 8:     else
 9:         add action $a$ to all dead-end agent goals plan $plan_d$
10:     end if
11: end for
12: if all dead-end agent goals plan $plan_d$ is valid then
13:     return all dead-end agent goals plan $plan_d$
14: else
15:     return error "all dead-end agent goals plan $plan_d$ of $Context_d$ is invalid" (this error
        will redirect RALSTP to the next decomposition if available)
16: end if
```

Example 6.48. The data in this example do not reflect an actual decomposition of a Driverlog benchmark problem but were created to illustrate how our procedure mitigates concurrency violations and creates the all dead-end agent goals plan from: the durative macro actions in a strategic problem Π_s, the plan of Π_s and the relaxed tactical planning problem plans corresponding to the durative macro actions in the plan of Π_s.

In Figure 6.21a we have an example of a strategic planning problem plan which contains three durative macro actions. Macro actions 1 and 3 both use the *driver1* agent so the mutual exclusions enforced by the *(stp_free_driver driver1)* and *(not (stp_free_driver driver1))* conditions and effects prevent the execution time of the two macro-actions to overlap in the strategic plan. Macro actions 2 and 3 both use the *truck2* agent so the mutual exclusions enforced by the *(stp_free_truck truck2)* and *(not (stp_free_truck truck2))* conditions and effects prevent the execution time of the two macro-actions to overlap in the strategic plan. Macro actions 1 and 2 do not have parent agents in common so their execution times can overlap in the strategic plan.

In Figure 6.21b we have the plans of the relaxed tactical planning problems encapsulated in the three durative macro actions present in the strategic planning problem plan from Figure 6.21a.



```
Strategic Plan of the Strategic Planning Problem derived from Contextual Set 1:
    0.000: (stp_context1_tactical1_driver1_truck1)  [3.000]   ; Macro Action 1
    0.000: (stp_context1_tactical2_driver2_truck2)  [3.000]   ; Macro Action 2
    3.000: (walk driver1 s1 s2)  [20.000]
    23.000: (board-truck driver1 truck2 s2)  [1.000]
    24.000: (drive-truck truck2 s2 s3 driver1)  [10.000]
    34.000: (stp_context1_tactical3_driver1_truck2)  [15.000] ; Macro Action 3
    ; Cost: 49
```

(a)

```
Tactical Plan 1 Corresponding to Macro Action 1 (stp_context1_tactical1_driver1_truck1):
    0.000: (unload-truck package1 truck1 s1)  [2.000]
    2.000: (disembark-truck driver1 truck1 s1)  [1.000]
    ; Cost: 3
    ; Plan of Tactical Planning Problem 1 derived from Contextual Set 1

Tactical Plan 2 Corresponding to Macro Action 2 (stp_context1_tactical2_driver2_truck2):
    0.000: (unload-truck package2 truck2 s2)  [2.000]
    2.000: (disembark-truck driver2 truck2 s2)  [1.000]
    ; Cost: 3
    ; Plan of Tactical Planning Problem 2 derived from Contextual Set 1

Tactical Plan 3 Corresponding to Macro Action 3 (stp_context1_tactical3_driver1_truck2):
    0.000: (load-truck package3 truck2 s3)  [2.000]
    2.000: (drive-truck truck2 s3 s4 driver1)  [10.000]
    12.000: (unload-truck package3 truck2 s4)  [2.000]
    14.000: (disembark-truck driver1 truck2 s4)  [1.000
    ; Cost: 15
    ; Plan of Tactical Planning Problem 3 derived from Contextual Set 1
```

(b)

```
All Dead-end Agent Goals Plan of Contextual Set 1:
    0.000: (unload-truck package1 truck1 s1)  [2.000]
    0.000: (unload-truck package2 truck2 s2)  [2.000]
    2.000: (disembark-truck driver1 truck1 s1)  [1.000]
    2.000: (disembark-truck driver2 truck2 s2)  [1.000]
    3.000: (walk driver1 s1 s2)  [20.000]
    23.000: (board-truck driver1 truck2 s2)  [1.000]
    24.000: (drive-truck truck2 s2 s3 driver1)  [10.000]
    34.000: (load-truck package3 truck2 s3)  [2.000]
    36.000: (drive-truck truck2 s3 s4 driver1)  [10.000]
    46.000: (unload-truck package3 truck2 s4)  [2.000]
    48.000: (disembark-truck driver1 truck2 s4)  [1.000
    ; Cost: 49
```

(c)

Fig. 6.21 Example of mitigating concurrency violations and creating the all dead-end agent goals plan (c) from a strategic planning problem plan (a) and the relaxed tactical planning problem plans (b) corresponding to the durative macro actions in the strategic planning problem.

When running Algorithm 28 on the strategic planning problem plan, we add to the all dead-end agent goals plan (Figure 6.21c) all regular (non-macro) durative actions in the strategic planning problem plan as well as all durative actions from all tactical plans. Each durative action added to the all dead-end agent goals plan from each tactical plan will have its start time updated by adding the start time of the corresponding durative macro-action in the strategic plan in order to avoid concurrency violations. Therefore, the start times of the durative actions added from tactical plan 3 to the all dead-end agent goals plan will be increased by 34 (the start time of durative macro action 3 in the strategic plan) in the all dead-end agent goals plan.

6.5 Achieving the Parent Agent Goals of an Instance Planning Problem

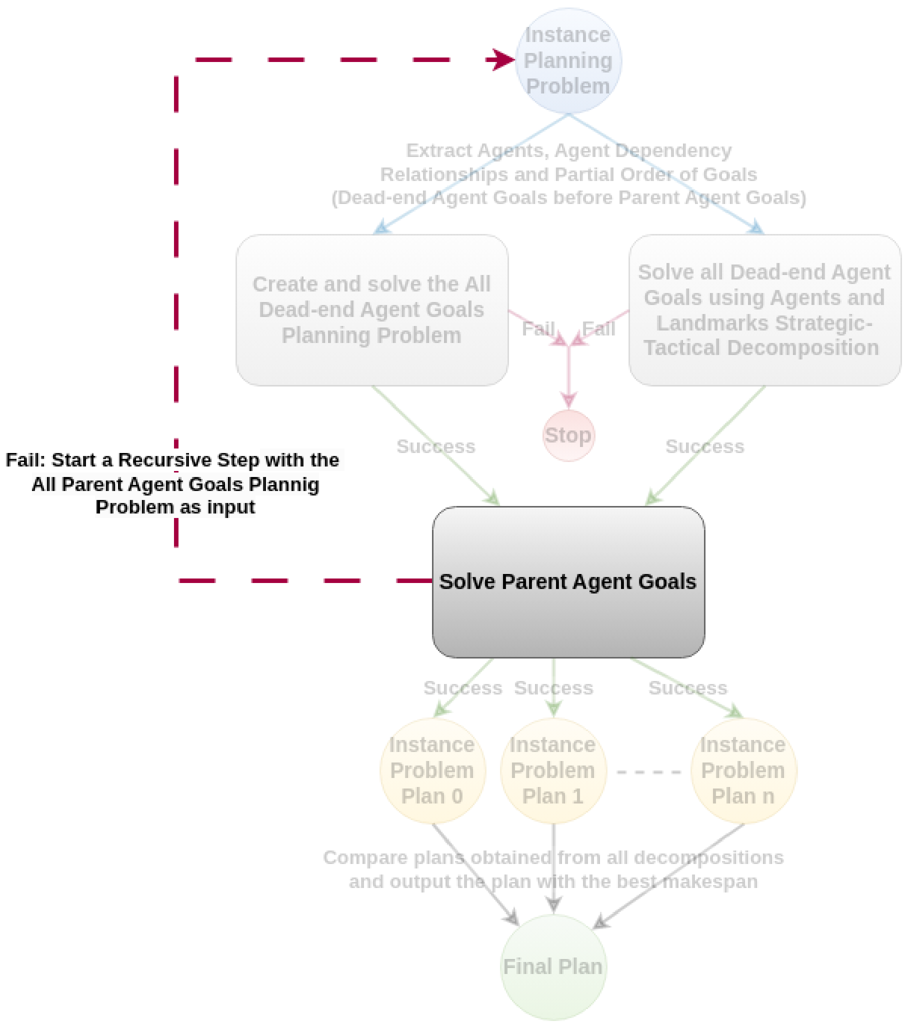

Instance Planning Problem
Extract Agents, Agent Dependency Relationships and Partial Order of Goals (Dead-end Agent Goals before Parent Agent Goals)
Create and solve the All Dead-end Agent Goals Planning Problem
Solve all Dead-end Agent Goals using Agents and Landmarks Strategic-Tactical Decomposition
Fail
Fail
Stop
Success
Success
Fail: Start a Recursive Step with the All Parent Agent Goals Plannig Problem as input
Solve Parent Agent Goals
Success
Success
Success
Instance Problem Plan 0
Instance Problem Plan 1
Instance Problem Plan n
Compare plans obtained from all decompositions and output the plan with the best makespan
Final Plan

Fig. 6.22 Location of Section 6.5 on the High-Level Flow Chart

Section Overview In this section, we present the procedure for creating and solving the parent agent goals of an instance planning problem. We will also present the recursive decomposition for situations where the parent agent goals are too difficult to solve without further decompositions.

6.5.1 Recursive Base Case

Section Overview The base case of the recursive procedure is successful if we are able to find a plan that achieves all parent agent goals in an instance planning problem Π_i.

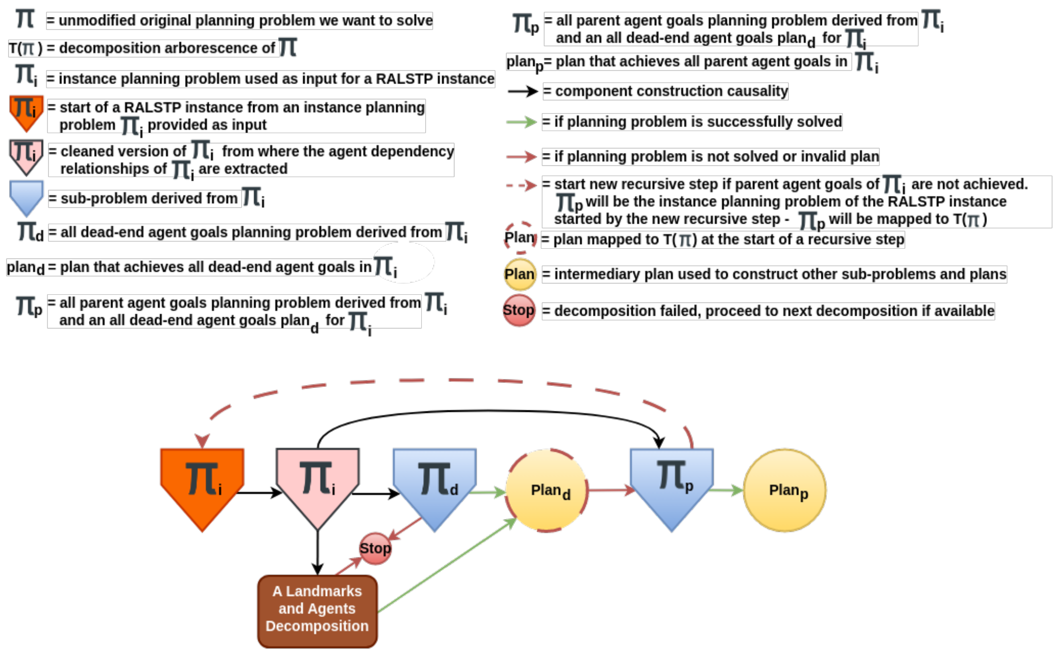
Definition 6.49. An *all parent agent goals planning problem* is defined as a tuple $\Pi_p := \{P,V,A,I,G\}$ where $\{P,V,A\}$ are identical to the instance planning problem Π_i from which Π_p is derived, I in Π_p is the final state of Π_i after executing a valid all dead-end agent goals plan $plan_d$ on Π_i and G is formed of all parent agent goals in Π_i.

Definition 6.50. An *all parent agent goals plan $plan_p$* is a plan that achieves all the parent agent goals of an instance planning problem Π_i when executing the plan from the final state of Π_i after executing a valid all dead-end agent goals plan $plan_d$ for Π_i.

The procedure for achieving the parent agent goals is described in Algorithm 29, which has a time complexity based on the chosen solver (the creation of Π_p is computed in polynomial time). The procedure takes as input a valid all dead-end agent goals plan $plan_d$ for Π_i and is the same regardless if $plan_d$ been obtained by solving the all dead-end agent goals planning problem Π_d (described in Section 6.3) or by using agents and landmarks strategic tactical planning (described in Section 6.4).

The procedure starts by checking the state where all dead-end agent goals of Π_i have been achieved (final state of Π_i after executing $plan_d$) to see if it also contains the achieved parent agent goals (lines 6 and 7). If all parent agent goals have also been achieved, we return an empty plan as the all parent agent goals plan $plan_p$ for Π_i (line 7).

In case there are still unachieved parent agent goals (i.e. $plan_d$ is not a valid plan for Π_i), we create an all parent agent goals planning problem Π_p that has the initial state I in Π_p the final state of Π_i after executing $plan_d$ and has the goal state G in Π_p all parent agent goals in Π_i (lines 8 and 9). The agents in Π_p will be different to that of Π_i, as the goal state of Π_p no longer has any goals with the dead-end agents d_{α_t} in Π_i as parameters. Therefore, our procedure continues by cleaning Π_p (re-executing Algorithm 5 with Π_p as input) in order to identify and remove from Π_p all new inactive dynamic objects $\neg\alpha_t$, all new inactive agent facts $f_{\neg\alpha_t}$ and all new inactive agent goals $g_{\neg\alpha_t}$ specific to Π_p (line 10). The cleaning process is likely to reduce the difficulty of Π_p, as the total number of agents α, the total number of inactive dynamic objects $\neg\alpha$ and the total number of entangled types $N(t_i)$ in Π_p is potentially reduced by no longer having any of the dead-end agents of Π_i in Π_p. We then attempt to solve Π_p (line 11). Π_p is solved as a stand-alone temporal planning problem using an off-the-shelf temporally expressive numeric planner. Therefore, Π_p can involve temporal expressiveness and numeric components.



Π = unmodified original planning problem we want to solve
T(Π) = decomposition arborescence of Π
Πi = instance planning problem used as input for a RALSTP instance
Πi = start of a RALSTP instance from an instance planning problem Πi provided as input
Πi = cleaned version of Πi from where the agent dependency relationships of Πi are extracted
= sub-problem derived from Πi
Πd = all dead-end agent goals planning problem derived from Πi
pland = plan that achieves all dead-end agent goals in Πi
Πp = all parent agent goals planning problem derived from Πi and an all dead-end agent goals pland for Πi
Πp = all parent agent goals planning problem derived from Πi and an all dead-end agent goals pland for Πi
planp = plan that achieves all parent agent goals in Πi
= component construction causality
= if planning problem is successfully solved
= if planning problem is not solved or invalid plan
= start new recursive step if parent agent goals of Πi are not achieved. Πp will be the instance planning problem of the RALSTP instance started by the new recursive step - Πp will be mapped to T(Π)
Plan = plan mapped to T(Π) at the start of a recursive step
Plan = intermediary plan used to construct other sub-problems and plans
Stop = decomposition failed, proceed to next decomposition if available
Πi
Πi
Πd
Pland
Πp
Planp
Stop
A Landmarks and Agents Decomposition

Fig. 6.23 Flowchart for Achieving all Parent Agent Goals of an Instance Planning Problem or Starting a Recursive Step

If we successfully obtain a parent agent goals plan $plan_p$ either by solving Π_p (line 10) or from achieving all parent agent goals in Π with $plan_d$ (case when $plan_p$ is an empty plan, lines 6 and 7), we proceed to create a plan that achieves all goals in Π_i from $plan_d$ and $plan_p$ (described in Section 6.6.1) and a plan that solves all goals in the unmodified original planning problem Π (described in section 6.6.2). Afterwards, we proceed to the remaining contextual decomposition from Σ in Π_i if available in order to potentially extract better plans for Π. If we can not find a plan that achieves all parent agent goals from Π_i, we start a recursive step (described in Section 6.5.2).

Algorithm 29 Algorithm for Achieving all Parent Agent Goals

```
Input: $\Pi_i$, $T(\Pi)$ and an all dead-end agent goals $plan_d$ (could be a from a $\Pi_d$ or a $\Pi_s$)
Output: an all parent agent goals plan $plan_p$ or the start of a new recursive step or the
start of a new dead-end agent goals decomposition
 1: if the all dead-end agent goals $plan_d$ algorithm input is a recursive dead-end agent
    goals $plan_{rd}$ then
 2:     for all actions $a$ in $plan_d$ do
 3:         start_time($a$) += makespan of the all dead-end agent goals plan mapped to the
            incoming edge of the node that maps $\Pi_i$ to arborescence $T(\Pi)$
 4:     end for
 5: end if
 6: if $plan_d$ also achieves all parent agent goals of $\Pi_i$ when executed on $\Pi_i$ then
 7:     return empty plan as the all parent agent goals plan $plan_p$ of $\Pi_i$
 8: else
 9:     create all parent agent agent goals planning problem $\Pi_p$ that has the final state of
        $\Pi_i$ after executing $plan_d$ as $I$ in $\Pi_p$ and all $g_{p\alpha_t}$ in $\Pi_i$ as $G$ in $\Pi_p$
10:     clean $\Pi_p$ by using Algorithm 5 with $\Pi_p$ as input
11:     all parent agent goals plan $plan_p$ = attempt to solve $\Pi_p$
12:     if not found (all parent agent goals plan $plan_p$) then
13:         add a new child node $cn$ to decomposition arborescence $T(\Pi)$ and map $\Pi_p$ to $cn$
14:         add a new edge $e$ to $T(\Pi)$ that has the node that maps $\Pi_i$ as tail and $cn$ as head
            and map $plan_d$ to $e$
15:         pause the current RALSTP instance
16:         initiate and start new RALSTP instance with $\Pi_p$ as input (recursive step)
17:         after the instance started at line 16 has stopped, re-start the paused RALSTP
            instance at line 15 and attempt the next dead-end agent goals decomposition in
            the re-started instance if available
18:     else
19:         return all parent agent goals plan $plan_p$
20:     end if
21: end if
```

6.5.2 Recursive Step

Section Overview A recursive step begins when we are not able to achieve the parent agent goals of the instance planning problem during a decomposition in the running RALSTP instance

Definition 6.51. A *recursive dead-end agent goals plan* $plan_{rd}$ is an all dead-end agent goals plan $plan_d$ that achieves all the dead-end agent goals $g_{d\alpha_t}$ of an instance planning problem mapped to a child node cn in $T(\Pi)$. Every action in a recursive dead-end agent goals plan $plan_{rd}$ has its start time increased with the makespan of the all dead-end agent goals plan $plan_d$ mapped to the incoming edge of cn.

The procedure for initiating a recursive step is described in Algorithm 29. **If we can not find a plan that achieves the parent agent goals of an instance planning problem Π_i during a decomposition dec of Π_i in the running RALSTP instance, we start a new recursive step. Starting a recursive step first requires pausing the running RALSTP instance. Afterwards, a new instance is started with the all parent agent goals problem Π_p from the decomposition dec as input (lines 12 to 16). Π_p becomes the instance planning problem, Π'_i, for the new RALSTP instance. This is done to gradually decompose and attempt to solve Π_p in the same way we decomposed and attempted to solve the instance planning problem Π_i from which Π_p was derived (Figures 6.25 and 6.26).**

When starting a recursive step, we add a new node n to the $T(\Pi)$ decomposition arborescence (line 13) and map to n the instance planning problem of the new RALSTP instance (Π'_i, which is the all parent agent goals problem Π_p derived from Π_i). We also map to the edge in $T(\Pi)$ that has node n as head the $plan_d$ plan executed on Π_i for obtaining the initial state of Π'_i (line 14).

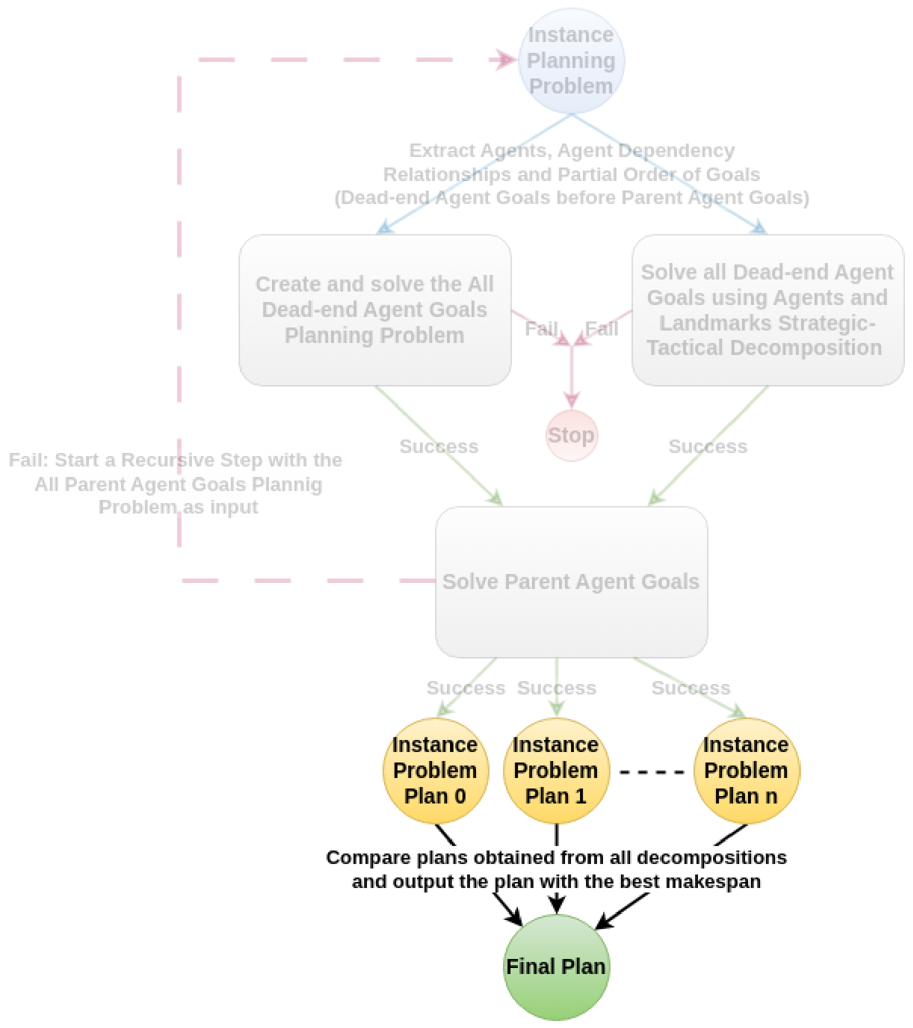
All decomposition elements specific to Π'_i will be extracted when initialising the new instance of RALSTP with Π'_i as input (Section 6.2). It is important to note that the dependency relationships in Π'_i and, consequently, the parent and dead-end classification of agents in Π'_i will be different to that of Π_i, as the agents with dead-end type t that were previously classified as dead-end agents d_{α_t} in Π_i are no longer dead-end agents in Π'_i due to the goal state G of Π'_i no longer having any goals g with the d_{α_t} of Π_i as parameters. This causes the node mapped to dead-end type t in the acyclic dependency relationships graph $DG(\Pi'_i)$ to no longer have an incoming edge. Also, agents previously classified as parent agents in Π_i might get classified as dead-end agents in Π'_i if their type is no longer mapped to a node that has at least one outgoing edge in the acyclic dependency relationships graph $DG(\Pi'_i)$. Therefore, Π'_i will have its own specific dead-end agent and parent agent goals as well as specific maximum quantity of unique parent agent groups M in Π'_i and corresponding contextual decomposition constructed from M in Π'_i. Π'_i will also have its own specific landmarks and relaxed landmarks which in turn will lead to specific similarity sets and contextual decompositions due to having new backchaining starting goals when extracting the landmarks from Π'_i (as the dead-end agent goals of Π_i are no longer backchaining starting goals in Π'_i).

After extracting all decomposition elements specific to Π'_i (Section 6.2), we attempt to solve Π'_i following the same steps and processes utilised for the solving attempt of Π_i. We first try to achieve the dead-end agent goals in Π'_i by creating and solving the all dead-end agent goals planning problem derived from Π'_i (described in Section 6.3) and by applying the agent and landmarks strategic-tactical planning to Π'_i (described in Section 6.4). If none of the dead-end agent goals decompositions of Π'_i yield an all dead-end agent goals plan for the dead-end agent goals in Π'_i, we end the recursive step by stopping the running (child) RALSTP instance and re-starting the last paused (parent) RALSTP instance. The re-started instance will proceed to its next dead-end agent goals decomposition if available (line 17). If we do find a valid all dead-end agent goals plan in a decomposition of Π'_i, we try to solve the parent agent goals in Π'_i by running Algorithm 29 with the found all dead-end agent goals plan for Π'_i as input (same approach as in the base case). The found all dead-end agent goals plan for Π'_i is a recursive dead-end agent goals plan $plan_{rd}$ as it was found during a recursive step. Therefore, the actions in $plan_{rd}$ are updated with the makespan of the all dead-end agent goals plan $plan_d$ mapped to the incoming edge of the node that maps Π'_i to $T(\Pi)$ (lines 1 to 3). This is done to mitigate any concurrency violations among the actions required for achieving the dead-end agent goals in the current recursive step with the actions required for achieving the dead-end agent goals in the base case and in any of the previous recursive steps (Figure 6.26).

If we are unable to achieve the parent agent goals of Π'_i, we start a new recursive step with the all parent agent goals planning problem derived from Π'_i as input and repeat the whole procedure described in this section. Otherwise, we proceed to create a plan that achieves all goals in Π'_i from the all dead-end agents plan and the all parent agents of Π'_i (described in Section 6.6.1) and a plan that solves all goals in the unmodified original planning problem Π (described in section 6.6.2).

As in the base case, the all dead-end agent goals planning problem, each strategic planning problem, each tactical planning problem and the all parent agent goals planning problem in a recursive step are solved as stand-alone numeric temporal planning problems using an off-the-shelf temporally expressive numeric planner. Therefore, any sub-problems created in any RALSTP instance can involve temporal expressiveness and numeric components, even if constructed and solved during a recursive step.

6.6 Processing Decomposition Plans to Obtain the Final Plan



Instance Planning Problem
Extract Agents, Agent Dependency Relationships and Partial Order of Goals (Dead-end Agent Goals before Parent Agent Goals)
Create and solve the All Dead-end Agent Goals Planning Problem
Solve all Dead-end Agent Goals using Agents and Landmarks Strategic-Tactical Decomposition
Fail
Fail
Stop
Success
Success
Fail: Start a Recursive Step with the All Parent Agent Goals Plannig Problem as input
Solve Parent Agent Goals
Success
Success
Success
Instance Problem Plan 0
Instance Problem Plan 1
Instance Problem Plan n
Compare plans obtained from all decompositions and output the plan with the best makespan
Final Plan

Fig. 6.24 Location of Section 6.6 on the High-Level Flow Chart

Section Overview In this section, we describe the procedure for stopping the RALSTP instances and for obtaining the final plan.

6.6.1 Creating a RALSTP Instance Plan

Section Overview In this section, we describe the procedure for obtaining a plan for an instance planning problem Π_i.

Definition 6.52. An *instance planning problem plan* $plan_i$ is a plan that achieves all the goals of an instance planning problem Π_i when executing $plan_i$ from the initial state of Π_i. $plan_i$ for Π_i is achieved by merging $plan_d$ for Π_i with $plan_p$ for Π_i.

If we are able to obtain an all dead-end agent goals plan $plan_d$ and an all parent agent goals plan $plan_p$ for an instance planning problem Π_i, our procedure continues by creating an instance planning problem plan $plan_i$ for Π_i (Algorithm 30 which computes in linear time). $plan_i$ is created by merging the actions in $plam_d$ with the actions in $plan_p$. To respect the partial ordering between all dead-end agent goals and all parent agent goals in Π_i, the start time of the actions in $plan_p$ are increased with the makespan of $plan_d$ when added to $plan_i$ (lines 2 and 3).

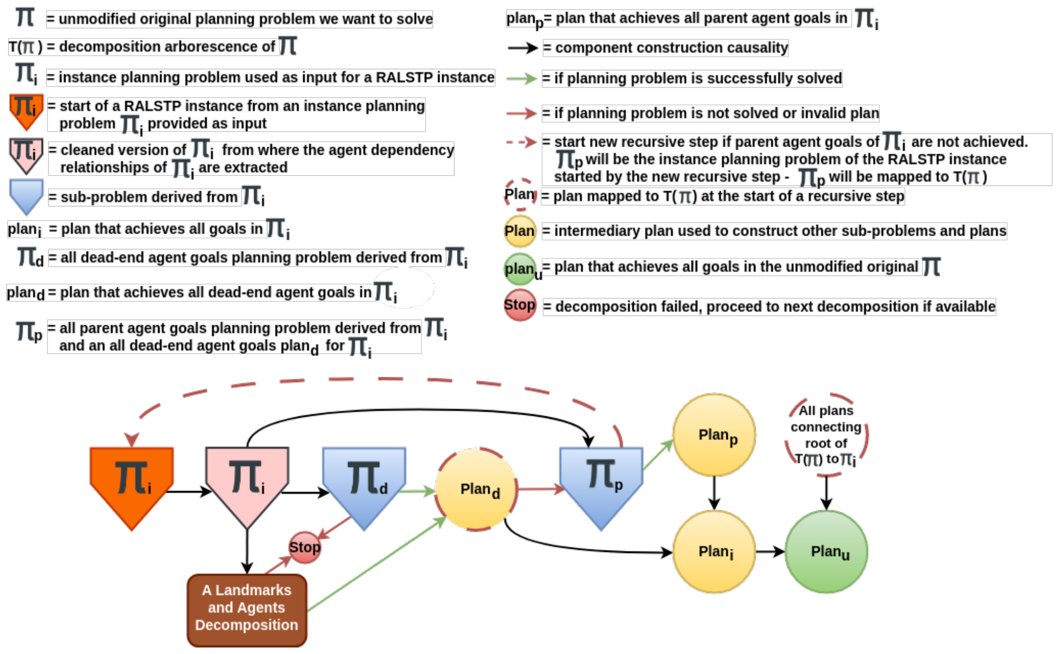
Algorithm 30 Algorithm for Creating an Instance Planning Problem Plan

```
Input: an all dead-end agent goals $plan_d$ (could be a from a $\Pi_d$ or a $\Pi_s$) and an all parent agent goals plan $plan_p$
Output: an instance planning problem plan $plan_i$
1: instance planning problem plan $plan_i$ = all dead-end agent goals plan $plan_d$
2: for all actions $a$ in all parent agent goals plan $plan_p$ do
3:     start_time($a$) += makespan of the all dead-end agent goals plan $plan_d$
4:     add action $a$ to the instance planning problem plan $plan_i$
5: end for
6: return instance planning problem plan $plan_i$
```

6.6.2 Creating a Plan for the Unmodified Original Problem

Section Overview In this section, we describe the procedure for obtaining a plan for the unmodified original planning problem Π.

Definition 6.53. An *unmodified original planning problem plan* $plan_u$ represents a plan that achieves all the goals of the unmodified original planning problem Π (the problem that was the input of the first initialised RASLTP instance).



Π = unmodified original planning problem we want to solve
T(Π) = decomposition arborescence of Π
Π_i = instance planning problem used as input for a RALSTP instance
= start of a RALSTP instance from an instance planning problem Π_i provided as input
= cleaned version of Π_i from where the agent dependency relationships of Π_i are extracted
= sub-problem derived from Π_i
plan_i = plan that achieves all goals in Π_i
Π_d = all dead-end agent goals planning problem derived from Π_i
plan_d = plan that achieves all dead-end agent goals in Π_i
Π_p = all parent agent goals planning problem derived from Π_i and an all dead-end agent goals plan_d for Π_i
plan_p = plan that achieves all parent agent goals in Π_i
= component construction causality
= if planning problem is successfully solved
= if planning problem is not solved or invalid plan
= start new recursive step if parent agent goals of Π_i are not achieved. Π_p will be the instance planning problem of the RALSTP instance started by the new recursive step - Π_p will be mapped to T(Π)
Plan = plan mapped to T(Π) at the start of a recursive step
Plan = intermediary plan used to construct other sub-problems and plans
plan_u = plan that achieves all goals in the unmodified original Π
Stop = decomposition failed, proceed to next decomposition if available
Π_i
Π_i
Π_d
Plan_d
Π_p
Plan_p
All plans connecting root of T(Π) to Π_i
Stop
Plan_i
Plan_u
A Landmarks and Agents Decomposition

Fig. 6.25 Flowchart for Creating an Unmodified Original Planning Problem Plan

If we are able to obtain an instance planning problem plan $plan_i$ for an instance planning problem Π_i, our procedure continues by creating an unmodified original planning problem plan $plan_u$ for Π (Algorithm 31 which computes in polynomial time). In the base case, the instance planning problem plan $plan_i$ is also the unmodified original planning problem plan $plan_u$ (line 1). However, if the RALSTP instance of Π_i was started by a recursive step, $plan_u$ is created by merging the actions in a $plan_i$ for Π_i with the actions in all of the all dead-end agent goals plans $plan_d$ that were obtained by all previous RALSTP instances in the recursive (directed) path between the unmodified original planning problem Π (the root of $T(\Pi)$) and the instance planning problem Π_i (lines 2 to 4). If an unmodified original planning problem plan $plan_u$ is found valid by VAL [41] when executed on the unmodified original planning problem Π, we return $plan_u$ as one of the solutions for Π (line 8).

Algorithm 31 Algorithm for Creating a Plan for the Unmodified Original Problem

```
Input: an instance planning problem plan $plan_i$, $\Pi_i$ and decomposition arborescence $T(\Pi)$
Output: an instance planning problem plan $plan_i$
1: unmodified original planning problem plan $plan_u$ = instance planning problem plan $plan_i$
2: for all edges $e$ in the directed path from the root of decomposition arborescence $T(\Pi)$ to the node that maps $\Pi_i$ in $T(\Pi)$ do
3:     for all actions $a$ in the all dead-end agent goals plan $plan_d$ mapped to $e$ do
4:         add action $a$ to unmodified original planning problem plan $plan_u$
5:     end for
6: end for
7: if unmodified original planning problem plan $plan_u$ is valid when executed on the original unmodified planning problem $\Pi$ we want to solve then
8:     return unmodified original planning problem plan $plan_u$ as a solution to $\Pi$ and continue to the next decomposition or last paused RALSTP instance if available
9: else
10:    return error "unmodified original planning problem plan $plan_u$ is invalid" (this error will redirect RALSTP to the next decomposition or last paused RALSTP instance if available)
11: end if
```

6.6.3 Stopping RALSTP Instances and Creating the Final Plan

Section Overview In this section, we describe the procedure for stopping all initialised RALSTP instances and for obtaining the final plan for the unmodified original planning problem Π.

A RALSTP instance can output multiple valid unmodified original planning problem plans $plan_u$ for Π. This happens due to the multiple potential dead-end agent goals decompositions, such as the decomposition of all the dead-end agent goals into a single problem (Section 6.3) and the multiple potential contextual decompositions obtained during the landmarks and agents decomposition procedure (Section 6.4). The dead-end agent goal decompositions can happen within each RALSTP instance, regardless if the instance was started in the base case or in a recursive step. Therefore, regardless if a valid $plan_u$ is found or not from a particular decomposition *dec* in the running RALSTP instance, the procedure continues with the next available decomposition in the running instance (lines 8 and 10 of Algorithm 31) upon finishing the solving attempt of *dec*. The technique as described in this thesis uses depth-first search for establishing the solving order of all possible decompositions obtained from a RALSTP instance. This design choice has been made to increase the chances of outputting a plan for Π when the planning time allowed for solving is finite. A breadth-first search ordering would increase the chances of solving only dead-end agent goals decompositions without ever advancing to parent agent goals when the planning time allowed for solving is finite. This means that our procedure will never start a new dead-end agent goals decomposition (if available) in the same instance it found a valid all dead-end agent goals plan $plan_d$ for the instance planning problem Π_i until it has found all possible unmodified original planning problem plans which may be found from all possible decompositions obtained from the final state of Π_i after executing $plan_d$. Only the descendant instances (started by recursive steps) of the instance where $plan_d$ was found may create and solve a new dead-end agent goals decomposition (which will be in effect a decomposition of the parent agent goals from a parent instance).

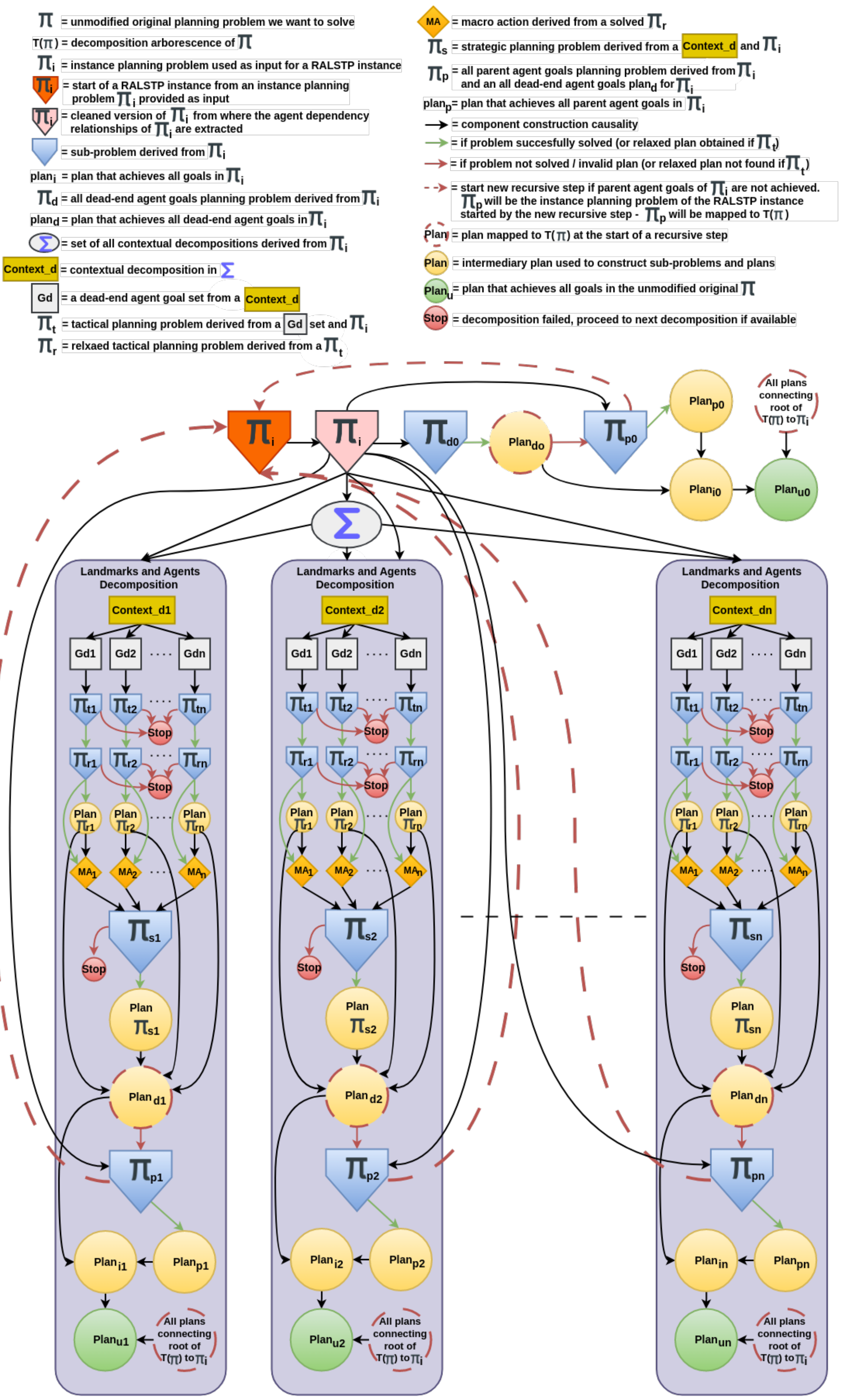

Π = unmodified original planning problem we want to solve
T(Π) = decomposition arborescence of Π
Π_i = instance planning problem used as input for a RALSTP instance
Π_i = start of a RALSTP instance from an instance planning problem Π_i provided as input
Π_i = cleaned version of Π_i from where the agent dependency relationships of Π_i are extracted
= sub-problem derived from Π_i
plan_i = plan that achieves all goals in Π_i
Π_d = all dead-end agent goals planning problem derived from Π_i
plan_d = plan that achieves all dead-end agent goals in Π_i
Σ = set of all contextual decompositions derived from Π_i
Context_d = contextual decomposition in Σ
Gd = a dead-end agent goal set from a Context_d
Π_t = tactical planning problem derived from a Gd set and Π_i
Π_r = relxaed tactical planning problem derived from a Π_t
MA = macro action derived from a solved Π_r
Π_s = strategic planning problem derived from a Context_d and Π_i
Π_p = all parent agent goals planning problem derived from Π_i and an all dead-end agent goals plan_d for Π_i
plan_p = plan that achieves all parent agent goals in Π_i
= component construction causality
= if problem succesfully solved (or relaxed plan obtained if Π_t)
= if problem not solved / invalid plan (or relaxed plan not found if Π_t)
= start new recursive step if parent agent goals of Π_i are not achieved. Π_p will be the instance planning problem of the RALSTP instance started by the new recursive step - Π_p will be mapped to T(Π)
Plan = plan mapped to T(Π) at the start of a recursive step
Plan = intermediary plan used to construct sub-problems and plans
Plan_u = plan that achieves all goals in the unmodified original Π
Stop = decomposition failed, proceed to next decomposition if available
Π_i
Π_i
Π_d0
Plan_do
Π_p0
Plan_p0
All plans connecting root of T(Π) to Π_i
Plan_i0
Plan_u0
Σ
Landmarks and Agents Decomposition
Context_d1
Context_d2
Context_dn
Gd1
Gd2
Gdn
Π_t1
Π_t2
Π_tn
Stop
Π_r1
Π_r2
Π_rn
Plan Π_r1
Plan Π_r2
Plan Π_rn
MA_1
MA_2
MA_n
Π_s1
Π_s2
Π_sn
Plan Π_s1
Plan Π_s2
Plan Π_sn
Plan_d1
Plan_d2
Plan_dn
Π_p1
Π_p2
Π_pn
Plan_i1
Plan_i2
Plan_in
Plan_p1
Plan_p2
Plan_pn
Plan_u1
Plan_u2
Plan_un

Fig. 6.26 Detailed Flowchart for Solving a Planning Problem using RALSTP

After exhausting all possible decompositions from all descendant instances of a RALSTP instance *inst* and all possible decompositions from *inst*, we stop *inst*, re-start the instance that was paused when *inst* was started (if such an instance exists), proceed to the next available decomposition in the resumed instance (Figure 6.26) and continue the search for all unmodified original planning problem plans $plan_u$ in the resumed instance (Algorithm 29 line 17). The procedure ends only when all RALSTP instances have exhausted all their possible decompositions. After all instances have finished (the first initialised instance that had Π as input will be the last to finish) we output as the final plan for Π the $plan_u$ plan with the best makespan among all valid $plan_u$ plans obtained from all decompositions in all RALSTP instances.

6.7 Summary of Chapter Contributions

In this chapter, we presented a collection of automatic decomposition techniques systematically arranged to reduce the difficulty metrics of a temporal numeric planning problem, with as few such decompositions as possible for finding a solution.

Specifically, we introduced an automatic decomposition technique that uses the agent dependency relationships to form a partial order between the dead-end agent goals and parent agent goals in order to decompose a problem into an all dead-end agent goals sub-problem and a parent agent goals sub-problem (Sections 6.3 and 6.5).

We presented an automatic decomposition method from PDDL encoded data which groups dead-end agent goals into goal sets based on the duplicate landmarks found between dead-end agent goals and based on the maximum quantity of unique parent agent groups with no common agents (Section 6.4.1). We also introduced a technique that groups the goal sets obtained from landmarks into multiple distinct contextual decompositions, similar to how humans use the available context and information to generate a variety of alternative solutions to a particular problem (Section 6.4.1).

We used the contextual decompositions and agent dependency classification to extend strategic tactical planning from a manual domain-engineer-dependent and time-consuming procedure into a fast automatic process that requires no human input (Sections 6.4.2 and 6.4.3). We described an automatic procedure for efficiently allocating parent agents in tactical planning problems outside of the solution search (Section 6.4.2.3) in order to not only decrease the solving difficulty but also to allow the solving of parent agent goals at the tactical level (Section 6.4.2.4). We introduced a relaxed version of tactical planning problems that can exponentially decrease the solving difficulty in comparison to regular tactical planning problems (Section 6.4.2.5).

Furthermore, we presented a high-level description (Section 6.1) as well as a detailed description of how the decomposition techniques are utilised in a recursive solving procedure that employs the agents and landmarks strategic tactical planning to obtain quality solutions for numeric temporally expressive problems with as few such decompositions as possible (Sections 6.3, 6.4 6.5 and 6.6). The description also illustrates a technique for using temporal data to mitigate all constraints between the dead-end agent goals and parent agent goals solutions as well as between recursive steps (Section 6.5).

Chapter 7

Evaluation

Chapter Overview In this Chapter, we evaluate an implementation of the recursive agent and landmarks strategic-tactical planning (RALSTP) procedure described in Chapter 6 against a broad range of state-of-the-art temporal planners on benchmark domains from past international planning competitions. We also present an analysis of the potential benefits of using relaxed tactical planning problems.

7.1 Difficulty of Relaxed Tactical Planning Problems

The potential exponential difficulty reductions obtained by using relaxed tactical planning problems can be clearly observed in the test problems in set DT at Section 5.4 from Chapter 5. All test problems in set DT have the same static environment and identical top-level goals: one driver parent agent goal, one truck parent agent goal and one package dead-end agent goal. In each test problem from set DT, only the driver and truck agents are gradually increased, without any corresponding goals. Figure 7.1 shows how the extra added agents exponentially increase the difficulty of the test problems for all tested planners to the point where a solution is no longer found.

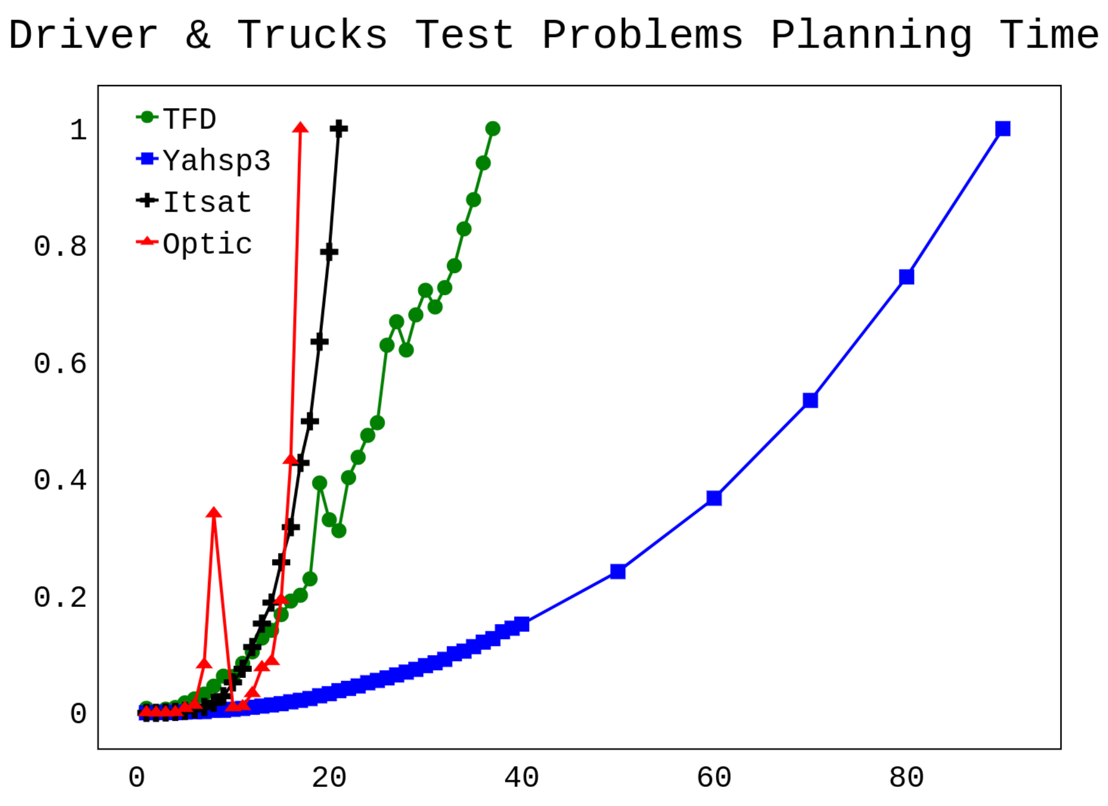


Fig. 7.1 Normalised Planning Time of the Test Problems in Set DT from Section 5.4. X-Axis: Number of Driver and Truck Agents per Problem. Y Axis: Normalised Planning Time

When applying RALSTP to any test problem Π_i in set DT, we obtain an identical relaxed tactical planning problem Π_r for all $\Pi_i \in$ DT. Π_r has as the mandatory parent agent group $\omega_{p\alpha_t}$ the driver and truck parent agents with corresponding $g_{p\alpha_t}$ goals in Π_i regardless of the number of added agents in the test problem Π_i from which Π_r was derived. This happens because the relaxed plan of each test problem Π_i contains the agents that have their $in\_goal\_state(p\alpha_t)$ goal state constraint set to true due to the corresponding $g_{p\alpha_t}$ parent agent goals in Π_i. All solutions for Π_r are also valid solutions for all test problems Π_i. This is the case as Π_r has agents that are present in every Π_i, has the same static environment as every Π_i and contains the same goals as every Π_i. Π_r, however, is identical to the first test problem, the problem that has only one agent per agent type. Therefore, Π_r is solved by RALSTP regardless of the planner selected as the tactical solver (from the planners tested in Section 5.4) irrespective of the number of added agents in the test problem from which Π_r was derived (Figure 7.1). The solution search operation of each relaxed tactical planning problem Π_r derived from a test problem Π_i remains constant regardless of the number of agents added to Π_i. However, the solution search operation of the test problems Π_i when solved directly with a planner increases exponentially the more driver and truck parent agents are added. This happens because the state space of the test problems is expanded by the entanglement of the extra-added agents while no such expansion takes place in Π_r regardless of the number of extra agents added to the problem Π_r was derived from. Therefore, using RALSTP can exponentially increase the size of solvable planning problems.

7.2 RALSTP Evaluation on Benchmark Problems

7.2.1 Evaluation Specifics

Section Overview In this section, we present the evaluation setup.

The evaluation of the technique presented in this thesis has been made using a proof-of-concept implementation built on top of a mix of legacy academic codebases (detailed in Section 8.2 from Chapter 8). The implementation consists of two versions. The first version, named RALSTP, uses regular (non-relaxed) tactical planning problems for the solving process. The second version, named RALSTP-rnd, uses a partial implementation of relaxed tactical planning problems in which the selection of a mandatory parent agent group outside of the solving process of tactical planning problems is done arbitrarily from the parent agents that have their parent agent constraints set to true in each tactical planning problem.

The two implementations have been evaluated against the latest versions of multiple robust and well-known temporal planners that employ different approaches to solving temporal problems and participated in past international planning competitions as well as the most competitive temporal planners that were part of the last international planning competition with a temporal track (IPC 2018). The evaluated planners are Optic [6], Itsat [49], Temporal Fast Downward (TFD) [26], Yahsp3 [64], TBurton [65], TFLAP [53], and TPSHE [32]. The evaluated domains represent all the benchmarks from past international planning competitions with temporal tracks that have the feature exploited by RALSTP explicitly encoded, that do not have irreversible state transitions (detailed in Section 8.3.2 of Chapter 8) and that do not require cooperative agents with the same parent type for solving (within the same map segment in case the problem has a disconnected map) as the feature (described in paragraph 6.4.2.5 from Chapter 6) has not been implemented. We also present an example of a domain where the exploited feature is not explicit (Section 7.2.5) which required modifications to the PDDL encoding for our technique to be beneficial.

Each problem was solved on a Dell XPS 15 9560 laptop with 32GB total and unrestricted RAM and a threshold of 1800 seconds per problem. The implementation specifics (detailed in Section 8.2 from Chapter 8) that are related to the evaluation process are as follows:

- The landmarks-based contextual decompositions $Context_d \in \Sigma$ for each planning problem are sorted in ascending order based on the number of their goal sets. This is done as, usually, fewer decompositions yield a better solution. The contextual decomposition obtained from the maximum quantity of unique parent agent groups is left for last, as the landmarks decompositions if found, are expected to yield better solutions.

- The decompositions in the base case were sufficient to solve the problems without needing the implementation of the recursive step (which would not have affected the results or algorithmic overheads in any way and was assigned to future work).

- Problem 'cleaning' was partially implemented. The cleaning process in the evaluated implementation consists of searching and removing the dead-end agent objects and facts from the all parent agent goals planning problems. The complete implementation of the cleaning process (future work) would have added the cost of an extra reachability analysis for every dynamic object in a planning problem to the algorithmic overhead, but both approaches are polynomial (max 151 reachability analyses, as the highest number of dynamic objects among all evaluated problems is 151, found in problem no 20 from the RTAM domain).

- The agent dependency identification considered all dynamic objects as agents and all static objects as necessary static objects due to the partial cleaning of planning problems ($N(\alpha)$ = all dynamic objects and $N(\Phi)$ = all static objects). However, even without ignoring the inactive dynamic objects and unnecessary static objects (that can decrease the accuracy of the obtained planning problem 'advice'), the obtained dependency relationships were sufficient to solve all problems. There are close to no differences in the algorithmic overhead of the dependency relationships extracting process if the input consists of active dynamic objects and necessary static objects or if the input consists of dynamic objects and static objects, as the algorithmic overhead for identifying the active dynamic objects and necessary static objects is accounted for in the problem 'cleaning' process.

- The partial implementation of the relaxed tactical planning problems in RALSTP-rnd consists of the selection of the mandatory parent agents outside the solving process of a tactical planning problem Π_t in order to decrease the solving difficulty of Π_t. The agents are selected at random from all agents with favourable parent agent constraints in Π_t. The complete implementation of the relaxation for tactical planning problems (future work) would have selected the parent agents in the relaxed plan of Π_t instead of an arbitrary selection and would have added the cost of finding the relaxed plan of Π_t to the algorithmic overhead. Intuitively, the complete implementation is expected to yield better results than the partial implementation due to having heuristic guidance for selecting the mandatory parent agents instead of a random selection.

- Multi-Agents with the same parent type not implemented
- Unlimited landmarks extraction time with a threshold of 450 seconds for the backchaining operations.
- TPSHE as the main planner and Optic as the backup planner in case a problem is not solvable by TPSHE.
- A threshold of max 120 seconds per sub-problem (tactical, strategic, etc.).
- Each sub-problem solution search terminates 5 seconds after the first plan is found and uses the best plan (if multiple plans are found) as the sub-problem plan.

The difficulty of each planning problem was estimated using the $N(\alpha)$ and $N(\Phi)$ difficulty metrics (described in Section 5.6 from Chapter 5) when $N(\alpha)$ = all dynamic objects and $N(\Phi)$ = all static objects (due to the partial implementation of the problem cleaning process). The results in the evaluation against IPC planners are scored using the IPC scoring system. The scoring system uses the ratio C*/C to calculate the score of a planner for each solved problem Π from a specific domain. C is the makespan of the best plan among all planners for Π and C* is the makespan of a reference plan for Π. In case a planner found multiple plans for a specific problem in the allocated time, we considered the plan with the best makespan for scoring purposes. Unsolved problems are scored with 0. The final score of a planner for a specific domain is the sum of all the scores obtained by the planner for each problem in the domain. Results marked with * represent results obtained using the all dead-end agent goals planning problem decomposition while the results not marked with * represent results obtained using the agents and landmarks strategic-tactical decomposition of the dead-end agent goals.

7.2.2 Driverlog

The Driverlog temporal domain consists of trucks that must be boarded by drivers in order to be driven to various locations in order to load, transport and unload packages.

7.2.2.1 RALSTP and RALSTP-rnd Driverlog Results and Algorithmic Overhead

In the Driverlog domain, RALSTP and RALSTP-rnd were not able to find valid contextual decompositions from the similarity goal sets in each problem, so the dead-end agent goals decompositions were constructed using only the maximum quantity of unique parent agent groups M technique (described in Section 6.4.1.2 from Chapter 6). The results of solving the Driverlog problems using RALSTP and RASTP-rnd are found in Tables 7.1 and 7.2. The second and third columns in the two tables represent the difficulty estimation of each Driverlog planning problem using the $N(\alpha)$ and $N(\Phi)$ difficulty metrics. Since the dead-end agent goals are split at random among a number of goal sets equal to the maximum quantity of unique parent agent groups M of each problem, we solved each Driverlog benchmark problem ten times (columns R1 to R10 in Tables 7.1 and 7.2) to reduce the impact of potential outliers on the results. The last two columns in Tables 7.1 and 7.2 represent the worst and median results among the ten solving attempts of a particular problem.

Π	$N(\alpha)$	$N(\Phi)$	**R1**	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**	**R9**	**R10**	**Best**
1	20	27	0.95	0.94*	0.94*	**1.00***	0.94*	0.94*	0.94*	0.94*	**1.00***	0.94*	210.00*
2	25	38	0.99	0.99	0.99	0.91	0.95	**1.00**	0.95	0.91	0.91	0.95	228.00
3	30	51	0.66	0.69	**1.00**	0.74	0.91	0.68	0.64	0.62	0.91	0.71	212.01
4	35	67	0.82	**1.00**	0.89	0.82	0.94	0.91	0.82	0.82	0.85	**1.00**	338.01
5	40	80	0.96	0.91	0.91	**1.00**	0.89	0.97	0.91	0.94	0.99	0.94	318.01
6	45	88	0.92	0.89	**1.00**	0.87	0.97	0.85	**1.00**	0.97	0.97	0.98	330.01
7	50	102	0.99	**1.00**	0.93	**1.00**	0.96	0.91	0.97	0.91	0.93	0.96	340.01
8	55	115	**1.00**	0.86	0.88	0.86	0.73	0.73	0.76	0.93	0.86	0.84	370.01
9	60	131	0.92	0.92	**1.00**	0.94	0.87	0.90	0.80	0.97	0.83	0.92	330.01
10	65	140	0.80	0.93	0.80	0.97	0.85	0.94	0.92	0.89	**1.00**	0.82	334.01
11	24	43	0.83	0.83	0.96	0.84	0.84	0.83	0.93	**1.00**	0.96	0.83	200.00
12	29	56	0.96	0.96	0.96	0.91	0.94	0.94	0.91	0.94	**1.00**	0.96	346.01
13	34	67	0.86	0.84	0.84	0.84	0.86	0.86	**1.00**	0.92	0.84	0.86	268.01
14	39	85	0.96	**1.00**	0.97	0.93	0.79	0.93	0.93	0.93	0.90	**1.00**	260.01
15	44	93	0.96	0.81	0.70	0.96	0.96	0.81	0.66	0.63	**1.00**	0.96	250.01
16	49	105	0.86	**1.00**	0.77	0.81	0.75	0.86	0.91	0.79	**1.00**	0.88	300.01
17	54	118	0.97	0.97	0.83	0.75	0.75	0.76	**1.00**	0.82	0.95	0.80	352.01
18	59	131	0.94	0.97	0.94	0.89	0.86	0.89	0.94	0.89	0.79	**1.00**	310.01
19	64	139	0.83	0.75	0.87	0.89	0.81	0.85	0.85	0.85	**1.00**	0.83	332.01
20	47	70	0.79	0.84	**1.00**	0.83	0.83	0.87	0.84	0.81	0.87	0.83	380.01
-	-	-	17.99	18.10	18.19	17.76	17.38	17.43	17.68	17.49	**18.56**	18.01	Score
-	-	-	**20/20**	**20/20**	**20/20**	**20/20**	**20/20**	**20/20**	**20/20**	**20/20**	**20/20**	**20/20**	Solved

Table 7.1 Results of the Driverlog benchmark problems solved by RALSTP.

Π	$N(\alpha)$	$N(\Phi)$	**R1**	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**	**R9**	**R10**	**Best**
1	20	27	0.94*	0.94*	0.94*	**1.00***	0.94*	0.94*	0.96	0.94*	0.94*	0.94*	210.00
2	25	38	0.82	**1.00**	0.82	0.89	0.82	0.82	0.82	0.82	0.82	0.82	240.01
3	30	51	0.83	0.86	**1.00**	0.83	0.67	0.66	0.80	0.80	0.80	0.84	264.01
4	35	67	0.85	**1.00**	0.80	0.85	0.82	0.99	0.83	0.86	0.96	0.82	318.01
5	40	80	0.65	0.63	0.70	0.71	**1.00**	0.63	0.70	0.74	0.76	0.77	288.01
6	45	88	0.92	**1.00**	0.92	0.92	0.97	0.90	0.86	0.73	0.86	0.93	380.01
7	50	102	0.84	0.80	**1.00**	0.93	0.91	0.96	0.91	0.99	0.95	0.79	404.01
8	55	115	0.82	0.80	0.93	0.98	0.97	**1.00**	0.97	0.89	0.96	0.86	430.01
9	60	131	0.82	0.91	0.87	0.86	**1.00**	0.87	0.82	0.85	0.97	0.97	430.01
10	65	140	0.74	0.83	0.77	**1.00**	0.72	0.91	0.70	0.83	0.83	0.91	394.01
11	24	43	**1.00**	0.72	0.72	0.76	0.75	0.73	0.76	0.76	0.80	0.75	224.00
12	29	56	0.78	0.97	0.79	0.80	0.74	0.97	0.74	0.90	**1.00**	0.97	280.01
13	34	67	0.97	0.82	0.82	0.91	0.86	0.74	**1.00**	0.94	0.84	0.82	310.01
14	39	85	0.85	0.87	0.85	0.67	0.85	**1.00**	0.79	0.82	0.67	**1.00**	280.01
15	44	93	0.97	0.89	0.91	0.89	0.86	**1.00**	0.94	0.82	0.84	0.84	320.01
16	49	105	0.88	0.90	**1.00**	0.84	0.94	0.95	0.88	0.98	0.93	0.91	346.01
17	54	118	0.92	0.70	0.87	0.80	0.76	**1.00**	0.92	0.88	0.80	0.98	462.01
18	59	131	0.87	0.89	0.83	0.83	0.89	0.87	0.89	0.96	0.93	**1.00**	402.01
19	64	139	**1.00**	0.83	0.87	0.81	0.68	0.85	0.90	0.91	0.71	0.82	410.01
20	47	70	0.74	**1.00**	0.92	0.79	0.78	0.85	0.96	0.89	0.94	0.79	350.01
-	-	-	17.22	17.36	17.34	17.06	16.94	**17.64**	17.17	17.32	17.32	17.53	Score
-	-	-	**20/20**	**20/20**	**20/20**	**20/20**	**20/20**	**20/20**	**20/20**	**20/20**	**20/20**	**20/20**	Solved

Table 7.2 Results of the Driverlog benchmark problems solved by RALSTP-rnd.

The averages of the algorithmic overheads in each of the ten solution searches of each Driverlog benchmark problem have been added in Table 7.3. The second and third columns in the table represent the difficulty estimation of each planning problem using the $N(\alpha)$ and $N(\Phi)$ difficulty metrics. The **Agents** columns represent the overheads for extracting the agents, agent dependency relationships and classifications in each problem. The **LM** columns represent the overhead for extracting the landmarks in each problem. The **LM-rlx** columns represent the overhead for relaxing the landmarks and creating the similarity goal sets. The **ADEAG** columns represent the decomposition and solving time when using the all dead-end agent goals planning problem for solving all dead-end agent goals. The **STP** columns represent the decomposition and solving time when using the agents and landmarks strategical tactical planning for solving all dead-end agent goals. The algorithmic overheads in the **Agents** columns clearly show that determining if a Driverlog planning problem is suitable for our solving approach is computationally cheap relative to the problem difficulty.

			RALSTP					RALSTP-rnd				
Π	$N(\alpha)$	$N(\Phi)$	**Agents**	**LM**	**LM-rlx**	**ADEAG**	**STP**	**Agents**	**LM**	**LM-rlx**	**ADEAG**	**STP**
1	20	27	0.0	0.2	0.0	10.6	34.5	0.4	0.3	0.0	10.7	34.0
2	25	38	0.4	2.1	0.0	11.2	41.0	0.9	2.0	0.0	10.5	40.0
3	30	51	1.0	5.1	0.0	10.9	50.8	0.7	5.0	0.0	11.2	48.0
4	35	67	2.0	10.1	0.0	11.8	60.5	1.8	10.0	0.0	11.7	56.8
5	40	80	3.0	22.2	0.0	18.1	80.1	3.2	21.2	0.0	17.4	66.5
6	45	88	5.1	34.3	0.0	36.4	101.0	4.5	31.8	0.0	33.4	77.4
7	50	102	7.8	87.5	1.0	35.0	115.3	7.1	82.2	0.6	34.3	93.6
8	55	115	11.6	130.7	1.0	152.8	163.9	10.9	116.5	1.0	150.0	108.9
9	60	131	16.5	159.8	1.0	93.9	189.0	15.2	146.7	1.0	84.6	128.4
10	65	140	22.8	327.9	1.0	98.4	193.2	21.4	314.5	1.0	103.6	152.2
11	24	43	1.0	2.0	0.0	10.4	44.0	0.7	2.0	0.0	10.6	42.5
12	29	56	1.0	5.3	0.0	11.5	52.6	1.2	5.4	0.0	11.4	49.0
13	34	67	2.1	13.5	0.0	12.1	63.0	2.0	13.3	0.0	11.9	58.0
14	39	85	4.0	17.1	0.0	17.5	82.2	3.7	16.1	0.0	17.4	66.9
15	44	93	6.1	52.3	0.0	24.7	102.6	5.6	48.3	0.0	23.2	80.2
16	49	105	10.0	58.6	0.1	65.3	131.2	8.6	51.0	0.0	54.4	94.0
17	54	118	13.3	139.7	1.0	114.3	184.6	12.3	128.8	1.0	102.6	125.0
18	59	131	18.9	234.0	1.0	177.2	197.0	17.4	203.8	1.0	172.3	134.5
19	64	139	24.7	443.3	2.0	87.9	207.9	22.8	392.7	1.7	84.2	159.2
20	47	70	4.2	28.8	0.0	57.9	95.4	4.0	26.9	0.0	53.4	76.0

Table 7.3 Average Algorithmic Overheads of RALSTP and RALSTP-rnd while solving the Driverlog benchmark problems when the results in Tables 7.1 and 7.2 were obtained.

7.2.2.2 Driverlog Evaluation Against IPC Planners

The results [1] of executing all Driverlog problems on all evaluated planners can be seen in Table 7.4. The second and third columns in the table represent the difficulty estimation of each planning problem using the $N(\alpha)$ and $N(\Phi)$ difficulty metrics. The rest of the columns represent the results of each evaluated planner using the IPC scoring system. For each Driverlog problem, we have used the worst and median among all ten results obtained by RALSTP and RALSTP-rnd (Tables 7.1 and 7.2) in the evaluation against the other planners.

Π	$N(\alpha)$	$N(\Phi)$	**RALSTP**		**RALSTP-rnd**		**Optic**	**Itsat**	**TFD**	**Yah-sp3**	**TF-LAP**	**TPSHE**	Best
			Me-dian	Worst	Me-dian	Worst							
1	20	27	0.71*	0.71*	0.71*	0.71*	-	**1.00**	-	0.61	-	0.89	159.15
2	25	38	0.57	0.54	0.47	0.47	-	**1.00**	-	0.34	0.88	0.65	136.14
3	30	51	**1.00**	0.89	0.93	0.75	-	-	-	-	0.87	0.55	302.01
4	35	67	0.96	0.91	**1.00**	0.93	-	-	-	0.37	-	0.75	372.01
5	40	80	**1.00**	0.95	0.84	0.75	-	-	-	0.35	-	0.85	340.01
6	45	88	**1.00**	0.87	0.82	0.65	-	-	-	-	-	0.61	340.01
7	50	102	0.95	0.91	0.77	0.66	-	-	-	-	-	**1.00**	338.01
8	55	115	**1.00**	0.84	0.95	0.80	-	-	-	-	-	0.56	430.01
9	60	131	**1.00**	0.88	0.73	0.69	-	-	-	-	-	0.24	360.01
10	65	140	**1.00**	0.88	0.78	0.65	-	-	-	-	-	0.31	369.01
11	24	43	0.55	0.55	0.44	0.42	-	**1.00**	-	0.44	0.55	0.87	131.07
12	29	56	0.55	0.53	0.61	0.53	-	**1.00**	-	-	0.80	0.70	200.09
13	34	67	**1.00**	0.97	0.85	0.74	-	-	-	0.53	-	0.89	310.01
14	39	85	**1.00**	0.85	0.85	0.67	-	-	-	-	-	0.75	280.01
15	44	93	**1.00**	0.71	0.79	0.73	-	-	-	0.27	-	0.60	285.01
16	49	105	**1.00**	0.87	0.93	0.85	-	-	-	-	-	0.63	350.01
17	54	118	**1.00**	0.90	0.81	0.64	-	-	-	-	-	0.37	426.01
18	59	131	**1.00**	0.87	0.75	0.70	-	-	-	-	-	-	340.01
19	64	139	**1.00**	0.88	0.80	0.65	-	-	-	-	-	0.46	392.01
20	47	70	0.89	0.84	**1.00**	0.86	-	-	-	0.61	-	0.41	402.01
-	-	-	18.18	16.36	15.82	13.84	0.00	4.00	0.00	3.51	3.10	12.09	Score
-	-	-	**20/20**	**20/20**	**20/20**	**20/20**	0/20	4/20	0/20	8/20	4/20	19/20	Solved

Table 7.4 RALSTP and RALSTP-rnd Evaluation Against IPC Planners on the Driverlog benchmark problems using IPC scoring.

The results in Table 7.4 show that RALSTP and RALSTP-rand were able to solve all Driverlog problems and obtained a far better IPC score than all other evaluated planners even when considering the worst result among the ten solutions obtained by each of the two implementation versions for each problem (Tables 7.1 and 7.2).

The all dead-end agent goals decomposition proved better than the agents and landmarks strategic tactical decomposition only in the easiest Driverlog problem according to

[1]The TBurton planner did not start due to a licensing issue of one of its components and outputted the following error upon execution: *'Lisp has expired. Please contact sales@franz.com for a new license file.'*

our difficulty metrics (result marked with * in Table 7.3). In the larger, more difficult problems, the agents and landmarks strategic tactical decomposition obtained overwhelmingly better results than the all dead-end agent decompositions.

Most Driverlog problems were too difficult to solve in the allocated threshold by most IPC planners except TPSHE. Optic and TFD were not able to solve any of the problems. Itsat was able to solve only four of the easier problems according to the $N(\alpha)$ and $N(\Phi)$ difficulty metrics but with the best makespans among all planners in all four solved problems. TFLAP was also able to solve only four of the easier problems according to $N(\alpha)$ and $N(\Phi)$ and obtained better results than RALSTP and RALSTP-rnd in two of them. Yahsp3 was able to solve problems with a slightly increased difficulty in comparison to the other IPC planners but with inferior results than RALSTP and RALSTP-rnd.

7.2.3 RTAM

The RTAM temporal-numeric domain simulates multiple simultaneous car accidents that happen at distinct locations. The accidents must be efficiently mitigated by a response team of ambulances, police cars, fire brigades and tow trucks.

7.2.3.1 RALSTP and RALSTP-rnd RTAM Results and Algorithmic Overhead

In the RTAM problems, RALSTP and RALSTP-rnd were able to find multiple valid landmarks-based contextual decompositions from the similarity goal sets in each problem and attempted to solve as many contextual decompositions as possible within the allocated threshold of 1800 seconds. The results as well as the algorithmic overheads of the solutions searches for each RTAM problem have been added to Table 7.5. The second and third columns in the table represent the difficulty estimation of each planning problem using the $N(\alpha)$ and $N(\Phi)$ difficulty metrics. The **Agents** columns represent the overheads for extracting the agents, agent dependency relationships and classifications in each problem. The **LM** columns represent the overhead for extracting the landmarks in each problem. The **LM-rlx** columns represent the overhead for relaxing the landmarks and creating the similarity goal sets. The **ADEAG** columns represent the decomposition and solving time when using the all dead-end agent goals planning problem for solving all dead-end agent goals. The **STP** columns represent the decomposition and solving time when using the agents and landmarks strategical tactical planning for solving all dead-end agent goals. The algorithmic overheads in the **Agents** columns clearly show that determining if an RTAM planning problem is suitable for our solving approach is computationally cheap relative to the problem difficulty.

Π	$N(\alpha)$	$N(\Phi)$	RALSTP						RALSTP-rnd					
			Make-span	Age-nts	LM	LM-rlx	AD-EAG	STP	Make-span	Age-nts	LM	LM-rlx	AD-EAG	STP
1	77	41	411	2	6	0	42	1098	507	2	9	0	47	873
2	90	41	345	3	7	0	46	1226	335	4	9	0	46	992
3	74	43	1256	1	8	0	24	1767	1256	1	10	0	27	1762
4	87	43	511	3	9	0	29	1753	577	3	11	0	30	1756
5	100	43	343	3	9	0	46	1742	396	5	11	0	45	1739
6	79	45	1290	1	10	0	20	1318	1290	1	14	0	23	1363
7	92	45	630	2	11	0	45	1742	535	3	14	0	45	1565
8	105	45	456	5	11	0	45	1739	395	5	14	0	46	1486
9	76	47	1607	0	10	0	21	1769	1666	1	13	0	22	1764
10	89	47	520	2	11	0	50	1347	588	3	13	0	56	1587
11	110	47	390	5	12	0	46	1559	351	6	16	0	46	1575
12	95	49	1632	2	16	0	42	1499	1632	2	20	0	34	1543
13	108	49	594	3	16	0	46	1623	608	4	21	0	46	1586
14	121	49	397	10	16	0	46	1664	549	8	22	0	46	1716
15	90	51	1518	1	15	0	30	859	1526	1	19	0	32	939
16	103	51	600	3	15	0	45	675	565	3	21	0	46	1006
17	116	51	374	6	16	0	46	732	410	7	20	0	46	601
18	125	53	2002	3	27	0	57	1713	1933	3	34	0	62	1701
19	138	53	894	6	28	0	46	1720	751	7	35	0	45	1713
20	151	53	767	9	28	0	46	1717	762	12	36	0	45	1707

Table 7.5 Results and Algorithmic Overheads of RALSTP and RALSTP-rnd when solving the RTAM benchmark problems.

7.2.3.2 Landmarks Contextual Decompositions vs Random Decompositions

The landmarks contextual decompositions were also evaluated against same-sized random decompositions (R1 to R10 in Table 7.6). A random decomposition $random_d$ is constructed from each landmarks-based contextual set $Context_d$ obtained from a RTAM test problem Π. Each $random_d$ has the same number of goal-sets G_d as the $context_d$ from which it was constructed. Each $G_d \in random_d$ has the same size as its corresponding $G_d \in context_d$. The dead-end agent goals in Π are randomly split among all $G_d \in random_d$ for each $random_d$ in Π (while in each $context_d$ in Π they are arranged according to the landmarks decomposition procedure described in Algorithm 18 when applied to Π). The results from the all dead-end agent goals problems have not been added to highlight the differences between the landmarks contextual decompositions and random decompositions. The result of the landmarks contextual decompositions vs ten instances of random decompositions can be seen in Table 7.6.

Π	**RAL STP**	**RAL STP -rnd**	**R1**	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**	**R9**	**R10**	**Best**
1	**1.00**	0.81	0.83	0.79	0.63	0.76	0.73	0.75	0.68	0.70	0.74	0.69	411.39
2	0.86	0.89	0.86	0.71	0.72	0.82	0.79	0.78	0.81	0.94	0.67	**1.00**	297.06
3	**1.00**	**1.00**	0.87	0.96	0.78	0.79	0.87	0.88	0.95	0.85	0.86	0.88	1,256.01
4	**1.00**	0.89	0.79	0.83	0.80	0.99	0.82	0.84	0.83	0.91	0.77	0.73	510.90
5	**1.00**	0.87	0.77	0.77	0.83	0.94	0.87	0.77	0.67	0.88	0.75	0.89	343.39
6	**1.00**	**1.00**	0.99	0.90	0.94	0.87	0.89	0.89	0.88	0.99	0.86	0.99	1,289.58
7	0.83	0.98	0.72	0.79	0.73	0.81	0.82	**1.00**	0.80	0.97	0.80	0.75	525.75
8	0.82	0.95	0.75	0.80	0.89	0.86	0.80	**1.00**	0.86	0.98	0.82	0.80	376.40
9	0.85	0.82	0.93	**1.00**	0.94	0.84	0.92	0.95	0.93	0.94	0.92	0.89	1,372.21
10	0.91	0.81	**1.00**	0.97	0.95	0.92	0.83	0.86	0.94	0.79	0.92	0.85	473.90
11	0.90	**1.00**	0.74	0.86	0.63	0.59	0.77	0.74	0.75	0.83	0.66	0.71	351.38
12	**1.00**	**1.00**	0.83	0.95	0.79	0.80	0.93	0.85	0.82	0.90	0.67*	0.80	1,632.13
13	**1.00**	0.98	0.71	0.64	0.67	0.56	-	-	0.66	0.71	-	0.74	593.72
14	**1.00**	0.72	0.64	-	-	-	-	-	-	-	-	-	397.24
15	0.95	0.94	0.93	0.98	0.98	0.99	0.91	**1.00**	0.94	0.99	0.97	0.90	1,434.92
16	0.94	**1.00**	0.95	0.92	0.88	0.93	0.84	0.89	0.89	0.94	0.95	0.96	564.73
17	**1.00**	0.91	0.93	0.88	0.88	0.63	0.81	0.84	0.84	0.78	0.79	0.77	373.55
18	0.97	**1.00**	0.87	0.85	0.79	0.87	0.82	0.91	0.85	0.78	0.81	0.88	1,932.57
19	0.84	**1.00**	-	-	0.59	-	-	-	-	0.73	-	-	750.72
20	0.92	0.93	-	-	-	**1.00**	-	-	-	-	-	-	705.43
-	**18.80**	18.50	15.10	14.60	14.44	14.96	13.43	13.94	14.08	15.61	12.95	14.23	Score
-	**20/20**	**20/20**	18/20	17/20	18/20	18/20	16/20	16/20	17/20	18/20	16/20	17/20	Solved

Table 7.6 Results of Landmarks Contextual Decompositions vs Random Decompositions of the RTAM benchmark problems using IPC scoring.

7.2.3.3 RTAM Evaluation Against IPC Planners

The results [2] of executing all RTAM problems on all evaluated planners can be seen in Table 7.7. The second and third columns in the table represent the difficulty estimation of each planning problem using the $N(\alpha)$ and $N(\Phi)$ difficulty metrics. The rest of the columns represent the results of each evaluated planner using the IPC scoring system.

Π	$N(\alpha)$	$N(\Phi)$	**RALSTP**	**RALSTP -rnd**	**Optic**	**Itsat**	**TFD**	**Yahsp3**	**TFLAP**	**TPSHE**	Best
1	77	41	0.96	0.78	-	**1.00**	-	0.62	-	0.61	395.22
2	90	41	0.70	0.72	-	**1.00**	-	0.48	-	0.39	240.16
3	74	43	**1.00**	**1.00**	-	-	-	0.70	-	0.70	1,256.01
4	87	43	0.76	0.68	-	**1.00**	-	0.52	-	-	390.24
5	100	43	0.83	0.72	-	**1.00**	-	0.45	-	0.41	285.19
6	79	45	**1.00**	**1.00**	-	-	-	0.35	-	0.60	1,289.58
7	92	45	0.85	**1.00**	-	-	-	0.43	-	-	535.42
8	105	45	0.66	0.76	-	**1.00**	-	0.35	-	0.25	300.16
9	76	47	**1.00**	0.96	-	-	-	0.61	-	0.94	1,606.84
10	89	47	**1.00**	0.89	-	-	-	0.51	-	-	520.12
11	110	47	0.90	**1.00**	-	-	-	0.30	-	0.49	351.38
12	95	49	**1.00**	**1.00**	-	-	-	0.41	-	0.63	1,632.13
13	108	49	**1.00**	0.98	-	-	-	0.25	-	0.43	593.72
14	121	49	**1.00**	0.72	-	-	-	0.38	-	0.30	397.24
15	90	51	**1.00**	**1.00**	-	-	-	0.43	-	0.63	1,518.25
16	103	51	0.94	**1.00**	-	-	-	0.41	-	0.43	564.73
17	116	51	**1.00**	0.91	-	-	-	0.29	-	0.37	373.55
18	125	53	0.97	**1.00**	-	-	-	0.46	-	0.65	1,932.57
19	138	53	0.84	**1.00**	-	-	-	0.32	-	-	750.72
20	151	53	0.99	**1.00**	-	-	-	0.37	-	0.80	761.73
	-	-	**18.40**	18.11	0.00	5.00	0.00	8.64	0.00	8.62	Score
	-	-	**20/20**	**20/20**	0/20	5/20	0/20	**20/20**	0/20	16/20	Solved

Table 7.7 RALSTP and RALSTP-rnd Evaluation Against IPC Planners on the RTAM benchmark problems using IPC scoring.

The results in Table 7.7 show that RALSTP and RALSTP-rand were able to solve all RTAM problems and obtained a far better IPC score than all other evaluated planners. Most RTAM problems were too difficult to solve in the allocated threshold by most IPC planners except TPSHE and Yahsp3. However, Yahsp3 and TPSHE obtained results with inferior quality to RALSTP and RALSTP-rnd in all problems regardless of difficulty. Optic, TFD and TFLAP were not able to solve any of the problems. Itsat was able to solve only five of the problems that have a lower $N(\Phi)$ difficulty metric but with the best makespans among all planners in all five solved problems.

[2]The TBurton planner did not start due to a licensing issue of one of its components and outputted the following error upon execution: *'Lisp has expired. Please contact sales@franz.com for a new license file.'*

7.2.4 ZenoTravel

In the ZenoTravel temporal domain, planes must transport passengers between cities while refuelling when needed.

7.2.4.1 RALSTP and RALSTP-rnd ZenoTravel Results and Algorithmic Overhead

In the ZenoTravel domain, RALSTP and RALSTP-rnd were not able to find valid contextual decompositions from the similarity goal sets in each problem, so the dead-end agent goals decompositions were constructed using only the maximum quantity of unique parent agent groups M technique (described in Section 6.4.1.2 from Chapter 6). The results of solving the ZenoTravel problems using RALSTP and RASTP-rnd are found in Tables 7.8 and 7.9. The second and third columns in the two tables represent the difficulty estimation of each ZenoTravel planning problem using the $N(\alpha)$ and $N(\Phi)$ difficulty metrics. Since the dead-end agent goals are split at random among a number of goal sets equal to the maximum quantity of unique parent agent groups M of each problem, we solved each ZenoTravel benchmark problem ten times (columns R1 to R10 in Tables 7.8 and 7.9) to reduce the impact of potential outliers on the results. The last two columns in Tables 7.8 and 7.9 represent the worst and median results among the ten solving attempts of a particular problem.

Π	$N(\alpha)$	$N(\Phi)$	**R1**	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**	**R9**	**R10**	**Best**
1	20	27	0.84	0.94	0.86	0.88	**1.00**	**1.00**	0.75	0.86	**1.00**	0.86	1,518.00
2	25	38	0.95	0.95	0.95	**1.00**	0.95	0.95	0.95	0.95	0.95	0.95	2,144.00
3	30	51	0.89	0.92	0.92	**1.00**	0.84	0.92	0.91	0.84	0.89	0.91	2,377.00
4	35	67	0.99	0.99	0.93	0.85	0.91	0.79	0.91	**1.00**	0.98	0.99	2,893.01
5	40	80	**1.00***	**1.00***	**1.00***	**1.00***	**1.00***	**1.00***	**1.00***	**1.00***	**1.00***	**1.00***	3,041.01
6	45	88	0.81	0.92	0.81	0.75	0.86	0.81	0.97	0.75	0.86	**1.00**	3,056.01
7	50	102	**1.00**	0.94	0.94	0.93	0.91	0.85	0.94	0.89	**1.00**	0.91	3,359.01
8	55	115	**1.00**	0.89	0.92	0.89	0.81	0.88	0.88	0.86	0.87	0.94	3,409.01
9	60	131	**1.00**	0.95*	0.95*	0.95*	0.95*	0.95*	0.95*	0.95*	0.95*	0.97	3,965.01
10	65	140	0.84	0.94	0.45	**1.00**	0.83	0.84	0.94	0.99	0.88	0.86	3,602.01
11	24	43	**1.00**	0.88	0.87	0.84	0.87	0.99	0.84	0.83	0.77	0.83	3,815.01
12	29	56	0.91	0.91	0.89	0.99	0.91	0.94	0.95	0.85	0.95	**1.00**	4,381.01
13	34	67	0.95	0.94	0.99	**1.00**	0.90	0.94	0.91	0.88	0.94	0.86	4,554.01
14	39	85	0.91	0.83	**1.00**	0.91	0.90	0.86	0.77	0.91	0.86	0.87	4,634.01
15	44	93	0.89	0.86	0.93	0.96	0.89	0.96	0.94	**1.00**	0.86	0.89	3,166.01
16	49	105	0.89	0.98	0.95	0.83	0.97	0.98	0.89	**1.00**	0.98	0.89	2,357.00
17	54	118	0.99	0.95	0.99	0.95	0.97	0.95	0.90	0.90	0.95	**1.00**	2,044.00
18	59	131	0.98	0.98	0.98	0.98	**1.00**	**1.00**	0.98	0.97	**1.00**	0.98	1,771.00
19	64	139	0.92	0.85	**1.00**	0.85	0.85	0.85	0.85	0.90	0.92	0.85	1,392.00
20	47	70	0.98	**1.00**	**1.00**	0.98	0.98	**1.00**	0.98	**1.00**	0.98	**1.00**	1,265.00
-	-	-	**18.74**	18.63	18.33	18.54	18.30	18.46	18.20	18.32	18.60	18.57	Score
-	-	-	**20/20**	**20/20**	**20/20**	**20/20**	**20/20**	**20/20**	**20/20**	**20/20**	**20/20**	**20/20**	Solved

Table 7.8 Results of the ZenoTravel benchmark problems solved by RALSTP.

Π	$N(\alpha)$	$N(\Phi)$	**R1**	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**	**R9**	**R10**	**Best**
1	20	27	**1.00**	0.86	0.96	0.86	0.87	0.75	0.86	0.86	0.86	0.86	1,518.00
2	25	38	0.91	0.85	0.85	0.88	0.85	0.85	0.85	**1.00**	0.85	0.85	1,911.00
3	30	51	0.81	0.90	0.99	0.99	0.89	**1.00**	0.81	**1.00**	0.96	0.89	2,308.00
4	35	67	0.84	0.83	0.78	0.78	0.78	0.91	0.84	0.99	**1.00**	0.72	2,660.01
5	40	80	**1.00***	**1.00***	**1.00***	**1.00***	**1.00***	**1.00***	**1.00***	**1.00***	**1.00***	**1.00***	3,041.01*
6	45	88	0.84	0.97	**1.00**	0.84	0.86	0.80	0.85	0.90	0.75	0.90	3,056.01
7	50	102	0.88	0.97	0.98	**1.00**	0.98	0.91	0.91	0.99	**1.00**	0.99	3,582.01
8	55	115	0.99	0.96	0.88	0.80	0.93	**1.00**	0.83	0.93	0.94	0.99	3,642.01
9	60	131	0.94	0.96	**1.00**	0.94	**1.00**	0.93	0.93	0.96	0.94	0.88*	3,692.01
10	65	140	0.88	0.88	0.94	0.93	0.83	**1.00**	0.88	0.88	0.89	0.88	3,583.01
11	24	43	0.94	0.99	0.85	0.94	0.99	**1.00**	**1.00**	0.99	**1.00**	0.89	4,301.01
12	29	56	0.95	0.95	0.95	0.95	0.95	0.90	0.95	**1.00**	0.95	0.90	4,401.01
13	34	67	0.85	0.82	0.81	0.94	0.94	0.94	0.89	0.86	**1.00**	0.81	4,301.01
14	39	85	0.96	0.91	0.97	0.93	0.96	0.96	**1.00**	0.95	0.96	0.92	5,130.01
15	44	93	0.91	0.96	**1.00**	0.96	0.96	0.85	0.99	0.97	0.90	0.93	3,289.01
16	49	105	0.86	0.90	0.91	0.90	0.92	0.95	**1.00**	0.96	0.90	0.95	2,404.00
17	54	118	0.82	0.82	0.79	0.89	0.88	0.81	0.82	**1.00**	0.80	0.80	1,871.00
18	59	131	0.97	**1.00**	0.97	0.98	0.97	0.98	0.97	0.97	0.97	0.97	1,841.00
19	64	139	0.87	0.97	**1.00**	0.87	0.94	0.87	0.94	0.97	0.91	0.91	1,549.00
20	47	70	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	1,365.00
-	-	-	18.24	18.51	18.61	18.39	18.49	18.42	18.32	**19.20**	18.57	18.06	Score
-	-	-	**20/20**	**20/20**	**20/20**	**20/20**	**20/20**	**20/20**	**20/20**	**20/20**	**20/20**	**20/20**	Solved

Table 7.9 Results of the ZenoTravel benchmark problems solved by RALSTP-rnd.

The averages of the algorithmic overheads in each of the ten solution searches of each ZenoTravel benchmark problem have been added in Table 7.10. The second and third columns in the table represent the difficulty estimation of each planning problem using the $N(\alpha)$ and $N(\Phi)$ difficulty metrics. The **Agents** columns represent the overheads for extracting the agents, agent dependency relationships and classifications in each problem. The **LM** columns represent the overhead for extracting the landmarks in each problem. The **LM-rlx** columns represent the overhead for relaxing the landmarks and creating the similarity goal sets. The **ADEAG** columns represent the decomposition and solving time when using the all dead-end agent goals planning problem for solving all dead-end agent goals. The **STP** columns represent the decomposition and solving time when using the agents and landmarks strategical tactical planning for solving all dead-end agent goals. The algorithmic overheads in the **Agents** columns clearly show that determining if a ZenoTravel planning problem is suitable for our solving approach is computationally cheap relative to the problem difficulty.

			RALSTP					**RALSTP-rnd**				
Π	$N(\alpha)$	$N(\Phi)$	**Agents**	**LM**	**LM-rlx**	**ADEAG**	**STP**	**Agents**	**LM**	**LM-rlx**	**ADEAG**	**STP**
1	25	17	0	2	0	11.3	36.5	0.2	2.0	0.0	11.0	36.0
2	35	17	0	3	0	11.2	39.8	0.4	3.5	0.0	10.9	39.0
3	45	17	1	4.6	0	10.8	111.5	0.5	4.9	0.0	11.1	105.9
4	55	17	1	6	0	5.8	43.3	0.9	6.0	0.0	5.9	40.5
5	65	17	1.0	8.2	0.0	11.8	44.2	0.8	8.2	0.0	11.8	43.1
6	75	17	1	10.1	0	41.4	126.1	1.2	10.2	0.0	41.3	109.0
7	85	17	1	13	0	17	121.5	1.4	13.1	0.0	16.8	113.8
8	95	17	2	15.1	0	16.3	129.6	1.8	15.7	0.0	17.2	118.1
9	105	17	2	18	0	11.6	136.1	1.9	18.5	0.0	11.4	120.6
10	65	22	2	17.4	0	41.5	56.6	1.8	17.8	0.0	45.6	45.7
11	65	27	2.9	30.7	0	46.9	62.3	2.6	31.6	0.0	46.1	52.0
12	65	32	4	46.8	0	46.2	68.3	4.0	47.1	0.0	46.7	59.5
13	65	37	5.1	492.9	1	47.6	78.1	5.9	484.7	1.0	47.1	66.0
14	65	42	7.1	108.6	0	47.9	89.4	7.2	109.8	0.0	47.7	78.2
15	80	32	8.3	83.1	0	48.3	151.4	8.4	83.7	0.0	48.3	119.7
16	85	32	12.2	117.4	1	49.9	252.4	11.4	115.4	1.0	49.7	172.0
17	90	32	15.9	812.2	0	51.6	340.7	12.7	632.3	0.0	50.0	194.3
18	95	37	27.3	330.1	1	57	905.9	21.8	260.9	1.0	53.8	439.6
19	100	37	31.4	379.0	1.1	58.2	841.1	27.0	321.8	1.0	56.1	402.7
20	105	37	36.7	419.5	1.4	59.4	1030.0	32.5	371.2	1.0	57.8	479.9

Table 7.10 Average Algorithmic Overheads of RALSTP and RALSTP-rnd while solving the ZenoTravel benchmark problems when the results in Tables 7.8 and 7.9 were obtained.

7.2.4.2 ZenoTravel Evaluation Against IPC Planners

The results [3] of executing all ZenoTravel problems on all evaluated planners can be seen in Table 7.11. The second and third columns in the table represent the difficulty estimation of each planning problem using the $N(\alpha)$ and $N(\Phi)$ difficulty metrics. The rest of the columns represent the results of each evaluated planner using the IPC scoring system. For each ZenoTravel problem, we have used the worst and median among all ten results obtained by RALSTP and RALSTP-rnd (Tables 7.8 and 7.9) in the evaluation against the other planners.

Π	$N(\alpha)$	$N(\Phi)$	**RALSTP**		**RALSTP-rnd**		**Op-tic**	**Itsat**	**TFD**	**Yah-sp3**	**TF-LAP**	**TPSHE**	Best
			Me-dian	Worst	Me-dian	Worst							
1	25	17	0.88	0.76	0.86	0.76	-	**1.00**	-	0.47	0.73	0.49	1,529.17
2	35	17	0.68	0.68	0.68	0.68	-	**1.00**	-	0.80	0.47	0.80	1,533.18
3	45	17	0.68	0.63	0.72	0.63	-	**1.00**	-	0.66	0.49	0.69	1,779.21
4	55	17	0.51	0.42	0.49	0.42	-	**1.00**	-	0.58	0.55	0.38	1,543.18
5	65	17	0.54*	0.54*	0.54*	0.54*	-	**1.00**	-	0.37	0.58	0.37	1,640.18
6	75	17	0.55	0.50	0.57	0.50	-	**1.00**	-	0.73	0.35	0.32	2,039.22
7	85	17	0.57	0.52	0.56	0.51	-	**1.00**	-	0.71	0.37	0.33	2,051.24
8	95	17	0.46	0.42	0.46	0.39	-	**1.00**	-	0.55	0.40	0.28	1,786.20
9	105	17	0.43*	0.43*	0.46	0.43*	-	**1.00**	-	0.69	-	0.43	1,813.20
10	65	22	0.70	0.36	0.71	0.66	-	**1.00**	-	-	0.46	0.46	2,869.33
11	65	27	0.97	0.88	**1.00**	0.86	-	-	-	-	0.79	0.65	4,346.01
12	65	32	0.98	0.90	**1.00**	0.95	-	-	-	-	0.70	0.45	4,644.01
13	65	37	**1.00**	0.91	0.99	0.91	-	-	-	-	-	0.71	4,837.01
14	65	42	**1.00**	0.87	0.98	0.93	-	-	-	-	-	0.42	5,237.01
15	80	32	0.98	0.92	**1.00**	0.88	-	-	-	-	-	0.41	3,414.01
16	85	32	**1.00**	0.86	0.93	0.87	-	-	-	-	-	0.67	2,451.00
17	90	32	**1.00**	0.94	0.94	0.90	-	-	-	-	-	0.64	2,151.00
18	95	37	**1.00**	0.98	0.95	0.95	-	-	-	-	-	0.37	1,801.00
19	100	37	**1.00**	**1.00**	0.98	0.93	-	-	-	-	-	0.27	1,645.00
20	105	37	**1.00**	0.99	0.94	0.94	-	-	-	-	-	0.31	1,280.00
-	-	-	**15.95**	14.53	15.75	14.64	0.00	10.00	0.00	5.55	5.88	9.46	Score
-	-	-	**20/20**	**20/20**	**20/20**	**20/20**	0/20	10/20	0/20	9/20	11/20	**20/20**	Solved

Table 7.11 RALSTP and RALSTP-rnd Evaluation Against IPC Planners on the ZenoTravel benchmark problems using IPC scoring.

The results in Table 7.11 show that RALSTP and RALSTP-rand were able to solve all ZenoTravel problems and obtained a far better IPC score than all other evaluated planners even when considering the worst result among the ten solutions obtained by each of the two implementation versions for each problem (Tables 7.8 and 7.9). Optic and TFD were not able to solve any of the problems.Itsat, Yahsp3 and TFLAP were able to solve only the problems that were mostly in the lower difficulty half according to the $N(\Phi)$ difficulty metric. Yahsp3 obtained comparable, yet slightly better results than RALSTP and

[3]The TBurton planner did not start due to a licensing issue of one of its components and outputted the following error upon execution: *'Lisp has expired. Please contact sales@franz.com for a new license file.'*

RALSTP-rnd in the problem where it could find solutions. TFLAP obtained inferior results to RALSTP and RALSTP-rnd in the problem where it could find solutions. Itsat obtained the best makespans among all planners in the problem where it could find solutions.

7.2.5 Rovers

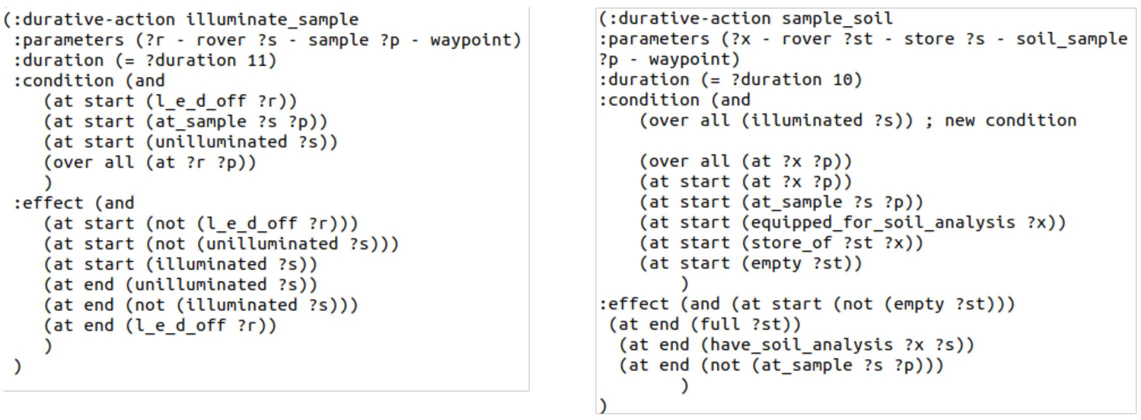
The Rovers temporal domain models problems confronted during the NASA Mars Exploration Rovers [62] missions. Rovers must travel between waypoints in over to collect rock and soil samples and to take images of objectives. The data from the samples and images must be communicated with the help of a lander. The rovers have various equipment configurations ranging from rock and soil extraction equipment to a camera that can support one or several image-capture modes.

7.2.5.1 Explicit vs Implicit Features

The Rovers domain is an example of a planning problem encoding where the feature we are exploiting is not explicit. Specifically, even though the rock and soil samples represent key elements of the planning problem and are part of the goal, they are not explicitly defined as objects. Instead, the rock and soil samples appear implicitly in the domain by being defined within predicates, such as in the *(have_rock_analysis ?r - rover ?w - waypoint)* and *(communicated_soil_data ?w - waypoint)* predicates. When executing RALSTP on the Rovers domain, the rock and soil samples are not identified as agents (as they are not defined as objects in the problem) and our procedure can not find solutions as the implicit agent dependency relationships can not be identified. However, as shown in Table 7.14, we gain huge benefits by explicitly defining the samples as objects. This encoding modification allows RALSTP to correctly identify the samples as dead-end agents, extract the agent dependency relationships and find competitive solutions.

7.2.5.2 Temporal Expressiveness

To showcase the capacity of RALSTP to solve temporally expressive problems, we added temporal expressiveness to the Rover domain by extending it with a new action, *illuminate_sample* (Figure 7.2), that allows a specific sample to be illuminated. An explicitly defined rock or soil sample must be illuminated by a rover during the sample collection procedure (similar to how a match must be lit during the fuse repair in the Matchcellar domain). Also, to showcase the scalability of our method, we extended the problems (IPC large problems version) with additional landers and added all possible sample communication goals for each sample in the problem. We also added all image communication goals supported by the existing cameras in a problem for all objectives present in the problem.



```
(:durative-action illuminate_sample
 :parameters (?r - rover ?s - sample ?p - waypoint)
 :duration (= ?duration 11)
 :condition (and
    (at start (l_e_d_off ?r))
    (at start (at_sample ?s ?p))
    (at start (unilluminated ?s))
    (over all (at ?r ?p))
    )
 :effect (and
    (at start (not (l_e_d_off ?r)))
    (at start (not (unilluminated ?s)))
    (at start (illuminated ?s))
    (at end (unilluminated ?s))
    (at end (not (illuminated ?s)))
    (at end (l_e_d_off ?r))
    )
 )
```

```
(:durative-action sample_soil
:parameters (?x - rover ?st - store ?s - soil_sample
?p - waypoint)
:duration (= ?duration 10)
:condition (and
    (over all (illuminated ?s)) ; new condition

    (over all (at ?x ?p))
    (at start (at ?x ?p))
    (at start (at_sample ?s ?p))
    (at start (equipped_for_soil_analysis ?x))
    (at start (store_of ?st ?x))
    (at start (empty ?st))
        )
:effect (and (at start (not (empty ?st)))
 (at end (full ?st))
  (at end (have_soil_analysis ?x ?s))
  (at end (not (at_sample ?s ?p)))
        )
)
```

Fig. 7.2 The *illuminate_sample* and *sample_soil* actions that use explicitly defined samples and make the Rover domain temporally expressive.

7.2.5.3 Rovers Results and Algorithmic Overhead

In the Rovers problems, RALSTP and RALSTP-rnd were able to find multiple valid landmarks-based contextual decompositions from the similarity goal sets in each problem and attempted to solve as many contextual decompositions as possible within the allocated threshold of 1800 seconds. The results as well as the algorithmic overheads of the solutions searches for each Rovers problem have been added to Table 7.12. The second and third columns in the table represent the difficulty estimation of each planning problem using the $N(\alpha)$ and $N(\Phi)$ difficulty metrics. The **Agents** columns represent the overheads for extracting the agents, agent dependency relationships and classifications in each problem. The **LM** columns represent the overhead for extracting the landmarks in each problem. The **LM-rlx** columns represent the overhead for relaxing the landmarks and creating the similarity goal sets. The **ADEAG** columns represent the decomposition and solving time when using the all dead-end agent goals planning problem for solving all dead-end agent goals. The **STP** columns represent the decomposition and solving time when using the agents and landmarks strategical tactical planning for solving all dead-end agent goals. The algorithmic overheads in the **Agents** columns clearly show that determining if a Rovers planning problem is suitable for our solving approach is computationally cheap relative to the problem difficulty.

Π	$N(\alpha)$	$N(\Phi)$	**RALSTP**						**RALSTP-rnd**					
			Make-span	**Age-nts**	**LM**	**LM-rlx**	**AD-EAG**	**STP**	**Make-span**	**Age-nts**	**LM**	**LM-rlx**	**AD-EAG**	**STP**
1	18	43	351	2	13	0	10	178	673	2	14	0	10	26
2	21	52	403	6	33	0	15	106	914	5	31	0	17	17
3	20	55	361	6	3	0	15	136	898	4	3	0	16	95
4	19	54	332	3	36	0	16	102	979	3	33	0	16	27
5	19	54	509	8	42	0	25	117	929	8	39	0	23	45
6	18	60	294	8	41	0	22	274	849	7	37	0	22	83
7	23	62	402	13	4	0	32	401	1127	12	3	0	31	72
8	25	75	1155	25	166	0	53	540	1155	24	162	0	53	521
9	33	79	577	34	277	0	60	249	-	34	272	0	59	378
10	27	78	495	28	203	0	64	404	1310	28	199	0	60	505
11	35	73	619	32	221	0	58	325	803	32	225	0	59	434
12	33	79	648	34	249	0	60	500	-	36	274	0	61	535
13	43	88	853	76	929	0	75	726	-	78	950	0	77	695
14	39	84	802	56	517	0	70	287	-	58	502	0	69	430
15	32	74	506	41	373	0	65	246	-	38	335	0	62	380
16	26	76	533	38	277	0	64	858	-	37	274	0	64	907
17	36	89	638	57	529	0	70	331	-	55	460	0	69	383
18	35	95	659	77	830	0	79	378	-	69	713	0	77	386
19	46	101	824	85	916	0	79	380	-	92	1127	0	85	396
20	48	105	-	-	-	-	-	-	-	-	-	-	-	-

Table 7.12 Results and Algorithmic Overheads of RALSTP and RALSTP-rnd when solving the Rovers planning problems.

7.2.5.4 Landmarks Contextual Decompositions vs Random Decompositions

The landmarks contextual decompositions were also evaluated against random decompositions with the same size specifications (as in the case of the RTAM problems). The results from the all dead-end agent goals problems have not been added to highlight the differences between the landmarks contextual decompositions and random decompositions. The result of the landmarks contextual decompositions vs ten instances of random decompositions can be seen in Table 7.13.

The vast majority of random decompositions were unsolvable as the goals required specific equipment and camera configurations that no single rover possessed in most problems or the decompositions contained disconnected locations. The few problems that had fully equipped rovers obtained poor results as the fully equipped rovers were sequentially used for all tactical problems. On the other hand, the landmarks decomposition (Table 7.12) was able to successfully group the goals into solvable similarity goal sets. This was achieved by the identification of similarity goal sets with characteristics specific to each problem such as: all rock samples, all soil samples, all colour images, all low-res images and all high-res images in the problem.

Π	**RAL STP**	**RAL STP -rnd**	**R1**	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**	**R9**	**R10**	**Best**
1	**1.00**	0.52*	0.52*	0.52*	0.52*	0.52*	0.52*	0.52*	0.52*	0.52*	0.52*	0.52*	351.02
2	**1.00**	0.44*	0.44*	0.44*	0.44*	0.44*	0.44*	0.44*	0.44*	0.44*	0.44*	0.44*	403.02
3	**1.00**	0.40*	0.40*	0.40*	0.40*	0.40*	0.40*	0.40*	0.40*	0.40*	0.40*	0.40*	361.02
4	**1.00**	0.34*	0.34*	0.34*	0.34*	0.34*	0.34*	0.34*	0.34*	0.34*	0.34*	0.34*	332.01
5	**1.00**	0.55*	0.55*	0.55*	0.55*	0.55*	0.55*	0.56*	0.56*	0.55*	0.55*	0.55*	509.02
6	**1.00**	0.35*	0.35*	0.35*	0.35*	0.35*	0.35*	0.35*	0.35*	0.35*	0.35*	0.35*	294.01
7	**1.00**	0.36*	0.35*	0.36*	0.35*	0.35*	0.35*	0.35*	0.35*	0.35*	0.35*	0.35*	402.02
8	**1.00***	**1.00***	**1.00***	**1.00***	**1.00***	**1.00***	**1.00***	**1.00***	**1.00***	**1.00***	**1.00***	**1.00***	1,155.04*
9	**1.00**	-	-	-	-	-	-	-	-	-	-	-	577.03
10	**1.00**	0.38	0.38	0.38	0.38	0.38	0.38	0.38	0.38	0.38	0.38	0.38	495.03
11	**1.00**	0.77	-	-	0.41	0.41	-	-	0.41	0.41	0.41	0.41	619.03
12	**1.00**	-	-	-	-	-	-	-	-	-	-	-	648.03
13	**1.00**	-	-	-	-	-	-	-	-	-	-	-	853.04
14	**1.00**	-	-	-	-	-	-	-	-	-	-	-	802.04
15	**1.00**	-	-	-	-	-	-	-	-	-	-	-	506.03
16	**1.00**	-	-	-	-	-	-	-	-	-	-	-	533.03
17	**1.00**	-	-	-	-	-	-	-	-	-	-	-	638.03
18	**1.00**	-	-	-	-	-	-	-	-	-	-	-	659.03
19	**1.00**	-	-	-	-	-	-	-	-	-	-	-	824.04
20	-	-	-	-	-	-	-	-	-	-	-	-	-
-	**19.00**	5.10	4.32	4.33	4.73	4.73	4.32	4.34	4.75	4.73	4.73	4.73	Score
-	**19/20**	10/20	9/20	9/20	10/20	10/20	9/20	9/20	10/20	10/20	10/20	10/20	Solved

Table 7.13 Results of Landmarks Contextual Decompositions vs Random Decompositions of the Rovers planning problems using IPC scoring. The second and third columns represent the difficulty estimation of each planning problem using the $N(\alpha)$ and $N(\Phi)$ difficulty metrics.

7.2.5.5 Rovers Evaluation Against IPC Planners

The results [4] of executing all Rovers problems on all evaluated planners can be seen in Table 7.14. The second and third columns in the table represent the difficulty estimation of each planning problem using the $N(\alpha)$ and $N(\Phi)$ difficulty metrics. The rest of the columns represent the results of each evaluated planner using the IPC scoring system.

Π	$N(\alpha)$	$N(\Phi)$	**RALSTP**	**RALSTP -rnd**	**Optic**	**Itsat**	**TFD**	**Yahsp3**	**TFLAP**	**TPSHE**	Best
1	18	43	0.70	0.37*	**1.00**	0.78	-	-	-	0.37	247.02
2	21	52	**1.00**	0.44*	-	-	-	-	-	0.45	403.02
3	20	55	**1.00**	0.40*	-	0.87	-	-	-	0.40	361.02
4	19	54	**1.00**	0.34*	-	0.84	-	-	-	0.35	332.01
5	19	54	**1.00**	0.55*	-	-	-	-	-	0.56	509.02
6	18	60	**1.00**	0.35*	-	-	-	-	-	0.35	294.01
7	23	62	**1.00**	0.36*	-	-	-	-	-	0.36	402.02
8	25	75	**1.00***	**1.00***	-	-	-	-	-	**1.00**	1,155.04*
9	33	79	**1.00**	-	-	-	-	-	-	0.37	577.03
10	27	78	**1.00**	0.38*	-	-	-	-	-	0.38	495.03
11	35	73	**1.00**	0.77	-	-	-	-	-	0.41	619.03
12	33	79	**1.00**	-	-	-	-	-	-	0.51	648.03
13	43	88	**1.00**	-	-	-	-	-	-	0.50	853.04
14	39	84	**1.00**	-	-	-	-	-	-	0.49	802.04
15	32	74	**1.00**	-	-	-	-	-	-	0.44	506.03
16	26	76	**1.00**	-	-	-	-	-	-	0.48	533.03
17	36	89	**1.00**	-	-	-	-	-	-	0.50	638.03
18	35	95	**1.00**	-	-	-	-	-	-	0.48	659.03
19	46	101	**1.00**	-	-	-	-	-	-	0.50	824.04
20	48	105	-	-	-	-	-	-	-	-	-
	-	-	**18.70**	4.95	1.00	2.49	0.00	0.00	0.00	8.92	Score
	-	-	**1.0**	10/20	1/20	3/20	0/20	0/20	0/20	**1.0**	Solved

Table 7.14 RALSTP and RALSTP-rnd Evaluation Against IPC Planners on the Rovers planning problems using IPC scoring.

The results in Table 7.14 show that RALSTP was able to solve all but one of the Rovers problems and obtained a far better IPC score than all other evaluated planners.

RALSTP was not able to preprocess problem 20 due to an unoptimised PDDL parsing procedure inherited from legacy code. As a solution, we could either optimise the current parser or implement other existing designs. An efficient PDDL parser is particularly required for large problems. However, optimising the current parsing procedure or implementing a different design is outside the scope of our thesis.

All RALSTP-rnd results except for the results of problem 11 have been obtained from the all dead-end agent goals decomposition (results marked with *) as RALSTP-rnd was not able to use the agents and landmarks strategic-tactical planning in most cases. This

[4]The TBurton planner did not start due to a licensing issue of one of its components and outputted the following error upon execution: *'Lisp has expired. Please contact sales@franz.com for a new license file.'*

happened because the goals in Rovers require specific equipment and camera configuration, and the random selection of parent agents did not select the appropriate rover for the goals in the tactical planning problems in all but problem 11. However, if the complete implementation of the relaxed tactical planning problem procedure had been made, the rovers would have been selected based on their appearance in the relaxed plan of each tactical planning problem instead of a random selection. This would have guaranteed that the selected rover would have been capable of solving the tactical planning problem it was obtained from via reachability analysis.

Most Rovers problems were too difficult to solve in the allocated threshold by most IPC planners except TPSHE. However, TPSHE obtained results with inferior quality to RALSTP and RALSTP-rnd in all problems regardless of difficulty. The temporal expressivity of the problems prevented Yahsp3 from finding valid solutions. TFD and TFLAP were not able to solve any of the problems. Optic was able to solve only the easiest problem according to the $N(\alpha)$ and $N(\Phi)$ difficulty metrics but with the best makespan among all planners. Itsat was able to solve only three of the easier problems according to the $N(\alpha)$ and $N(\Phi)$ difficulty metrics. Only in the easiest solved problem did Itsat obtain a comparable, yet slightly better result than RALSTP and RALSTP-rnd.

7.3 Summary of Results

The overall results in all tested domains show that RALSTP and RALSTP-rand not only solved the larger, more difficult, problems that the evaluated IPC planners were unable to solve but also obtained better quality solutions than most IPC planners in the commonly solved problems. The inferior solutions obtained even in the commonly solved problems show that planners used without external decomposition techniques make costly sacrifices in solution quality to be able to solve difficult problems in comparison to the cost of our abstracted decomposition technique. The few exceptions where the evaluated IPC planners obtained better results than our technique were in the less difficult problems according to the $N(\alpha)$ and $N(\Phi)$ difficulty metrics.

The data in the **Agents** columns of the algorithmic overheads tables clearly show that determining if a planning problem is suitable for our solving approach is computationally cheap relative to the problem difficulty.

A strong correlation can be observed between the $N(\alpha)$ and $N(\Phi)$ difficulty metrics and the capacity of the evaluated IPC planners to find a solution. The metrics are particularly relevant when comparing problems from the same domain.

The analysis performed in Section 7.1 showed the potential exponential reductions in the difficulty of tactical planning problems if applying the relaxation technique described in Section 6.4.2.5 from Chapter 6.

Chapter 8

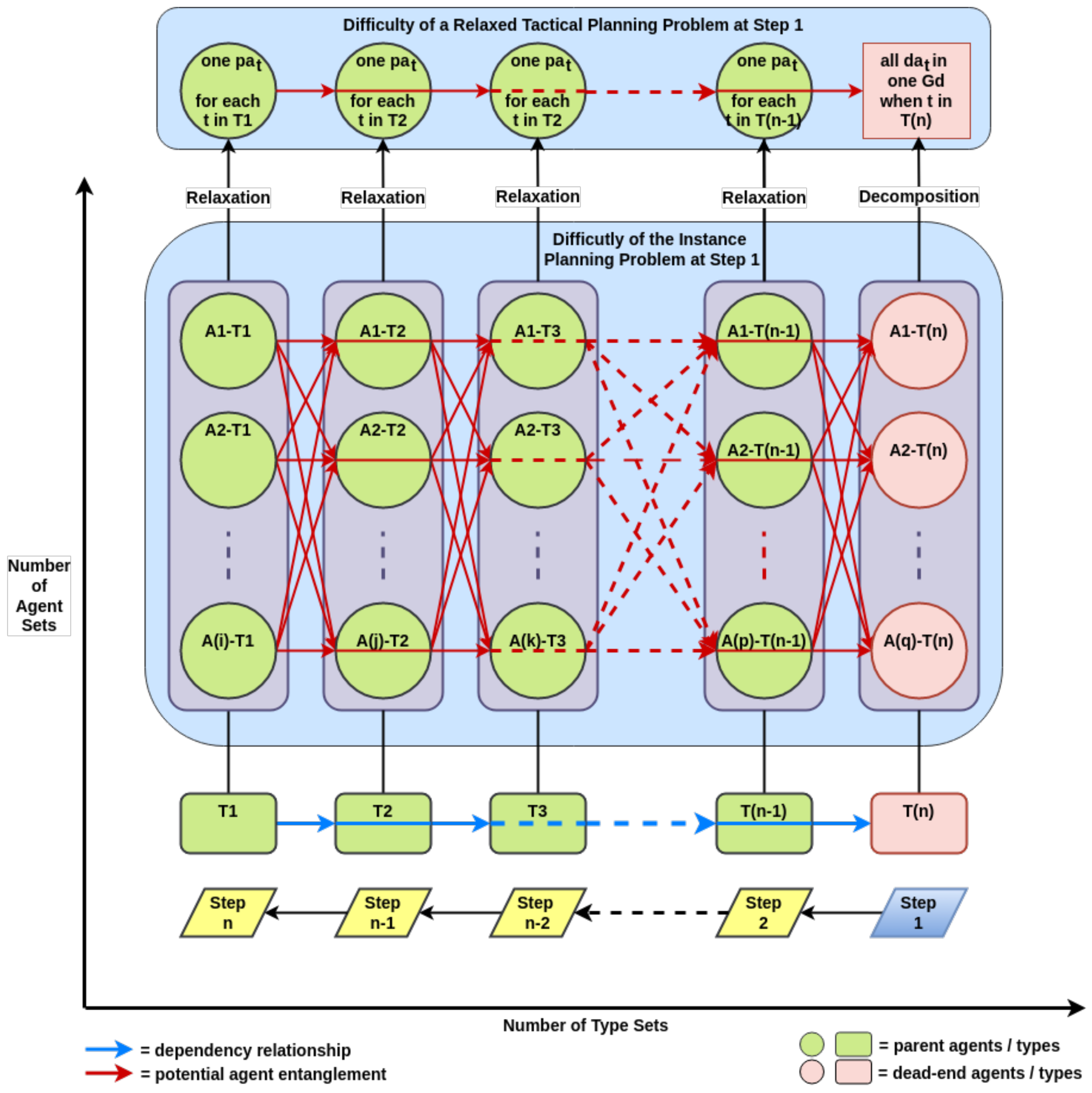
Discussion and Future Work

Chapter Overview In this chapter, we analyse the applicability and limitations of RALSTP and present potential avenues for future research.

8.1 Technique Analysis

8.1.1 Type and Size of Planning Problems

RALSTP can operate on temporally expressive numeric problems as all sub-problems are solved as stand-alone numeric temporal planning problems using an off-the-shelf temporally expressive numeric planner. Our technique makes use of temporal data to potentially modify the action start times in all sub-plans in a way that eliminates all eventual constraint violations among all actions from all sub-plans (detailed in Sections 6.4.3.2 and 6.5 from Chapter 6. RALSTP works best on large planning problems from which we can extract an acyclic, directed and weakly connected dependency graph that contains all the agent types present in a specific planning problem (Figure 8.1). In such problems, the partial ordering according to the dependency relationships, combined with the agents and landmarks strategic-tactical decomposition during each individual recursive step (Figure 8.3), can output sub-problems with a significantly reduced difficulty compared with the initial problem. In addition, the external selection of the mandatory parent agent group $\omega_{p\alpha_t}$ from the relaxed plan of a tactical planning problem Π_t can further substantially reduce the difficulty of a relaxed tactical planning problem Π_r as it eliminates the parent agent entanglement for selecting a unique parent agent group during the solution search. This can lead to an exponential reduction of the state space of a tactical planning problem that has multiple parent types and multiple parent agents with a distinct parent type (as shown in Section 7.1 of the evaluation).



Difficulty of a Relaxed Tactical Planning Problem at Step 1
one pa$_t$ for each t in T1
one pa$_t$ for each t in T2
one pa$_t$ for each t in T2
one pa$_t$ for each t in T(n-1)
all da$_t$ in one Gd when t in T(n)
Relaxation
Relaxation
Relaxation
Relaxation
Decomposition
Difficutly of the Instance Planning Problem at Step 1
A1-T1
A2-T1
A(i)-T1
A1-T2
A2-T2
A(j)-T2
A1-T3
A2-T3
A(k)-T3
A1-T(n-1)
A2-T(n-1)
A(p)-T(n-1)
A1-T(n)
A2-T(n)
A(q)-T(n)
T1
T2
T3
T(n-1)
T(n)
Step n
Step n-1
Step n-2
Step 2
Step 1
Number of Agent Sets
Number of Type Sets
= dependency relationship
= potential agent entanglement
= parent agents / types
= dead-end agents / types

Fig. 8.1 Difficulty of the Instance Planning Problem and a Relaxed Tactical Planning Problem at Step 1 (base case of the recursion). Ti = set of unique instantiated dynamic types (no types in common between any two sets Ti,Tj). Ai-Tj = set of unique agents α_t with type $t \in$ Tj, of all f_{α_t} agent facts and of all g_{α_t} agent goals (no α_t agents in common between any two sets Ap-Tj,Aq-Tj). For all types t in Tj there can be zero or max one α_t agent in Ai-Tj.

Therefore, RALSTP can operate on very large problems, as it can prune the state space of the initial problem and allow a planner to only focus on the state space of a single relaxed tactical planning problem found in the base case or in an individual recursive step (Figures 8.1 and 8.2) or on a single strategic abstraction found in the base case or in a single recursive step (Figure 8.3) of the decomposition.

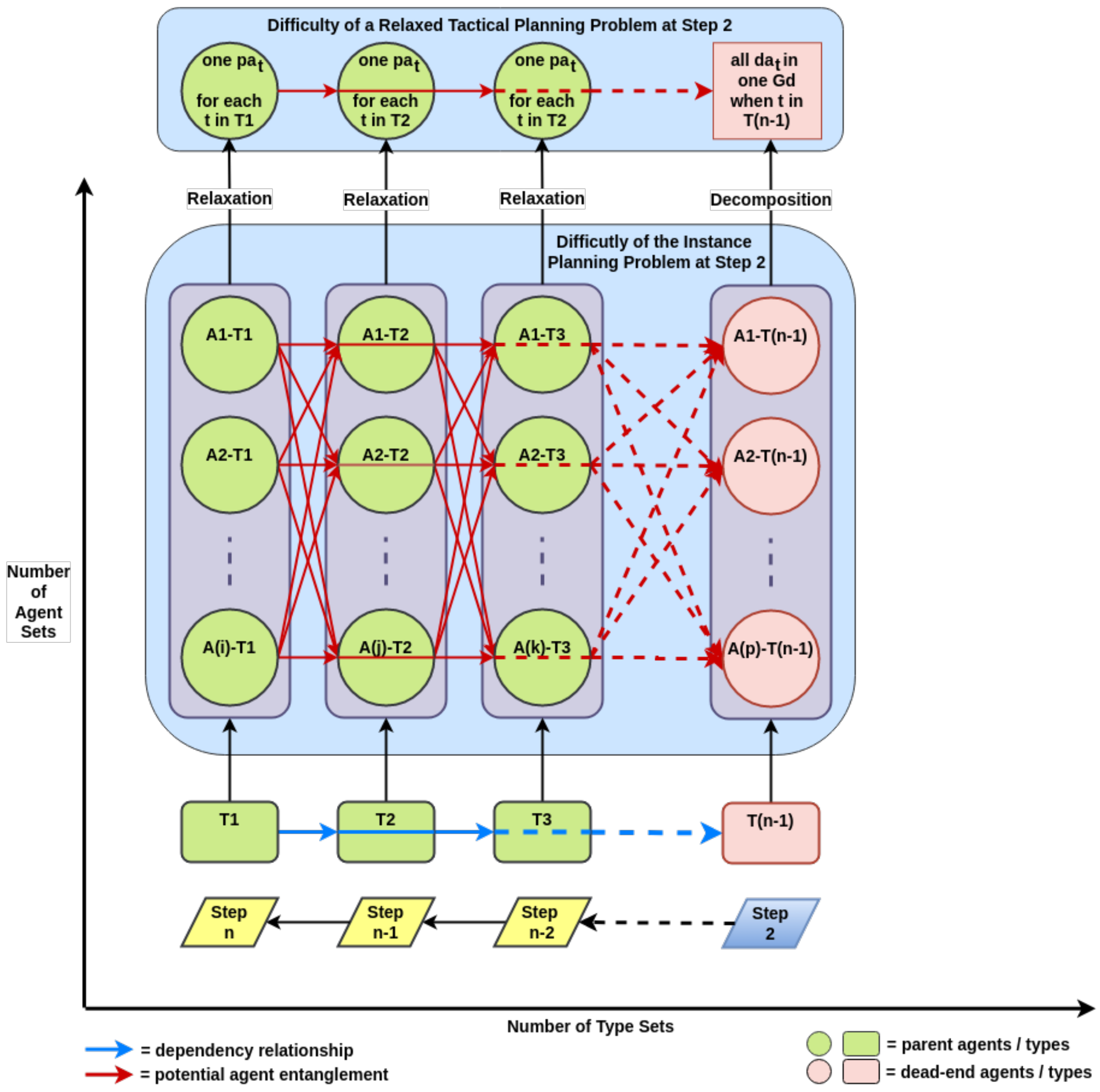

Difficulty of a Relaxed Tactical Planning Problem at Step 2
one pa_t for each t in T1
one pa_t for each t in T2
one pa_t for each t in T2
all da_t in one Gd when t in T(n-1)
Relaxation
Relaxation
Relaxation
Decomposition
Difficutly of the Instance Planning Problem at Step 2
A1-T1
A2-T1
A(i)-T1
A1-T2
A2-T2
A(j)-T2
A1-T3
A2-T3
A(k)-T3
A1-T(n-1)
A2-T(n-1)
A(p)-T(n-1)
T1
T2
T3
T(n-1)
Step n
Step n-1
Step n-2
Step 2
Number of Agent Sets
Number of Type Sets
= dependency relationship
= potential agent entanglement
= parent agents / types
= dead-end agents / types

Fig. 8.2 Difficulty of the Instance Planning Problem and a Relaxed Tactical Planning Problem at Step 2 (first recursive step). Ti = set of unique instantiated dynamic types (no types in common between any two sets Ti,Tj). Ai-Tj = set of unique agents α_t with type $t \in$ Tj, of all f_{α_t} agent facts and of all g_{α_t} agent goals (no α_t agents in common between any two sets Ap-Tj,Aq-Tj). For all types t in Tj there can be zero or max one α_t agent in Ai-Tj.

If the problem structure is compatible, the size of a problem solvable by RALSTP is limited only by the capacity to preprocess the initial problem, by the difficulty of each individual relaxed tactical planning problem found in the base case or in an individual recursive step (which can be exponentially lower than the difficulty of the initial problem if the initial problem has multiple parent types and multiple parent agents with a distinct parent type) by the difficulty of each individual strategic abstraction found in the base case or in an individual recursive step of the decomposition and by the difficulty of the all dead-end agent problems in the last recursive step (which could also be decomposed into a set of trivial sub-problems, as many as one sub-problem for each dead-end agent with corresponding goals).

8.1.2 Abstraction Properties

Our technique respects the ordered monotonicity property [43] as the refinement of the strategic plan with the information from the tactical plan never affects the literals in the strategic plan. Our technique also respects the downward refinement property [5], as once we find a strategic plan we immediately refine it with the information obtained from the plans at the tactical level and are never required to backtrack to the strategic level.

8.1.3 Soundness, Completeness and Complexity

All algorithms in RALSTP have been designed considering the main aim of our technique: to efficiently solve large-scale problems. The Algorithms in RALSTP can be classified into three major categories: element extraction, decomposition and solving.

The element extraction part (described in Chapter 4) is composed of domain-independent algorithms designed with a heuristic approach that aims to identify problem-specific 'advice' in order to potentially reduce the difficulty of planning problems and facilitate efficient decompositions. The heuristic nature of the algorithms makes them lack completeness, but our aim is to find quality solutions to large-scale (hard) problems, so guaranteeing completeness is not a practical concern, as using other existing techniques either yields bad solutions or no solutions at all (as shown in the empirical evaluation from Chapter 7). The algorithmic overhead in the first category is minimal for identifying if a problem is suitable for our procedure (detailed in the algorithmic overhead analyses from Chapter 7). The more expensive algorithms (such as the landmarks extraction) only execute if a planning problem has been identified as compatible (as described in Section 6.2 from Chapter 6), so using RALSTP as the get-go technique for solving large-scale temporally expressive numeric planning problems will either fail fast or have a high chance of yielding competitive solutions (as shown in the evaluation from Chapter 7. On the other hand, it is possible to engineer domains in a way that affects the accuracy of the element identification algorithms. However, we have not encountered (so far) a situation where a plan found with RALSTP was invalid. Additionally, VAL [41] is integrated into RALSTP to rule out with minimal cost any invalid solutions that might occur.

The decomposition part (described in Chapter 4) is mostly formed of sorting and merging algorithms used for goal decomposition and sub-problem creation. The algorithms in the decomposition part are sound with respect to their specific semantics and have a negligible algorithmic overhead in comparison to the solution search. Their completeness is reliant on the output of the element extraction algorithms, so completeness is not guaranteed but not a practical concern considering the aim of RALSTP and the subpar outcomes obtained by using alternative solutions.

The recombination algorithms from the solving part (described in Chapter 4) have a negligible algorithmic overhead and are sound and complete in reference to their specific semantics and properties. However, the solving part employs off-the-shelf planners in the algorithms responsible for finding solutions to all sub-problems derived from the main problem. Therefore, these algorithms inherit the soundness, completeness and algorithmic overheads of the chosen solvers.

Overall, our empirical data (presented in Chapter 7) shows that using RALSTP as the get-go technique for solving large-scale temporally expressive numeric planning problems either has a low cost when the problems are incompatible or the cost of using RALSTP for solving such problems is justifiable considering the scale of addressable problems and the quality of the solutions. RALSTP does not guarantee completeness, but considering the aim of RALSTP is to solve large-scale problems and the subpar outcomes of using alternative solutions for these types of problems, completeness is not a practical concern. There are, of course, ways to mitigate this (the integration of a breadth-first solution search after the main technique finishes or fails, often employed by other techniques to claim completeness), but, in reality, no existing planner can be considered practically capable of finding all solutions to all large scale problems. So far we have not encountered a situation where a solution found by RALSTP was invalid, but all solutions found by RALSTP are validated with VAL [41]. Therefore, RALSTP is sound.

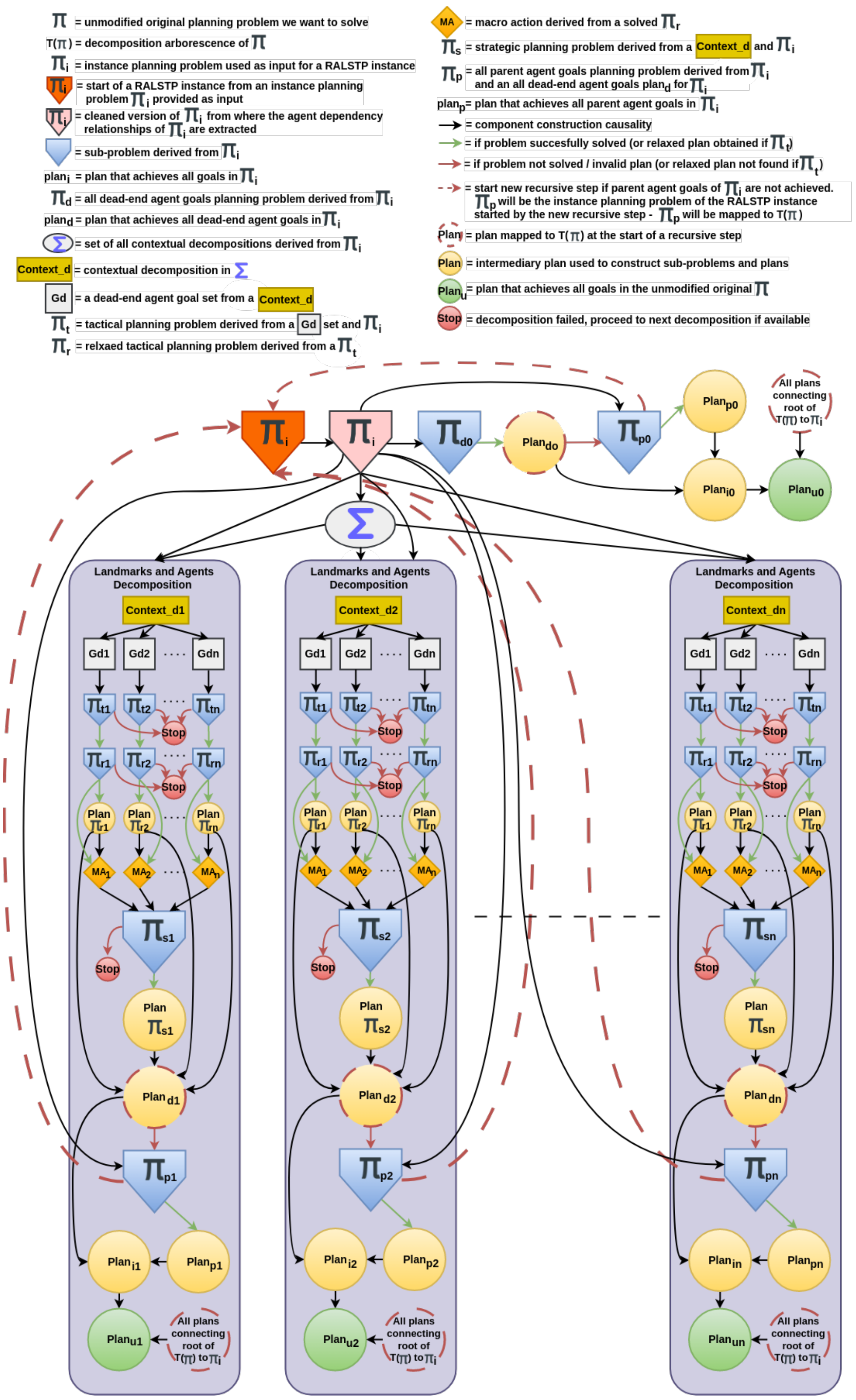
8.1.4 Optimality and Solution Quality

The solutions obtained by RALSTP are not guaranteed to be optimal, as using decompositions leads to the isolation of the components of a problem and to a disregard of the global constraints of the problem when solving the sub-problems. However, the few inferior solutions obtained by RALSTP in the evaluation in comparison to the solutions of state-of-the-art planners are mostly in instances where the problems have low difficulty (the small problems). On the other hand, the solutions found by state-of-the-art planners in the larger, more difficult problems (if they were able to find solutions), are overall inferior to the solutions of RALSTP. The results of the evaluation correlate with the results in the RBS problem [15], the results of the examples in the motivation and the results in the difficulty evaluation from Chapter 5. Overall, all results tend to show that the solution quality in difficult problems compatible with RALSTP is worse when solved with regular planners without decompositions in comparison to using efficient decompositions such as the ones in RALSTP. This happens because the addition of entangled types and objects exponentially expands the state space of a problem to the point where the solving approaches of regular planners have to make costly trade-offs (such as using only one agent or one unique parent agent group for solving the whole problem) in order to find a solution. **However, RALSTP prunes such exponentially expanded state spaces both polynomially using the partial ordering obtained from the dependency relationships and the agents and landmarks strategic-tactical decomposition as well as exponentially by relaxing the tactical planning problems (as shown in Figures 8.1, 8.2 and Section 7.1). Also, the decomposition acts as an efficient agent management control centre that divides the responsibility of solving the goal among the maximum number of unique parent agent groups at every step of the decomposition. Therefore, if we use the solutions obtained by state-of-the-art planners as a qualitative benchmark, RALSTP obtains very good solutions to large, difficult, problems even if the solutions are not optimal.**

8.2 RALSTP Software

Our thesis has been created by performing both theoretical and practical experimentation. The practical experimentation has been conducted by developing new software specific to this thesis using a rapid prototyping approach. The experimentation concluded with a proof-of-concept implementation of the work presented in this thesis. The largest part of the proof-of-concept was built on top of the C++ temporal landmarks extraction codebase used in the work by Karpas et al. (2015). The landmarks codebase is itself built on top of an old version of Optic (clp version) [6] that uses an old version of VAL [41]. The codebase was used for every aspect of RALSTP except for solving the sub-problems and for the creation of the macro-actions. The 'cplex' version of Optic and the latest version of TPSHE [32] were integrated with the landmarks extraction codebase for parsing and solving the sub-problems. The tactical planning problems and plans were also parsed with the latest version of VAL (PlanToValStep and ValStep in particular) which we modified to create the macro-actions and which was also integrated with the landmarks codebase.

Using a mix of legacy academic codebases was good for rapidly prototyping several design choices until we settled for the proof-of-concept akin to the description in our thesis. However, the mix of codebases was not ideal for an optimal implementation of our algorithms due to different optimisation focuses and constraints that came with the legacy code. While no problems were encountered with the latest version of VAL, we did encounter some issues with the PDDL parser from the landmarks extraction code, which made the implementation more complicated and less optimised than it could have been. For example, we encountered situations where the types of objects and constants were



π = unmodified original planning problem we want to solve
T(π) = decomposition arborescence of π
π_i = instance planning problem used as input for a RALSTP instance
= start of a RALSTP instance from an instance planning problem π_i provided as input
= cleaned version of π_i from where the agent dependency relationships of π_i are extracted
= sub-problem derived from π_i
plan_i = plan that achieves all goals in π_i
π_d = all dead-end agent goals planning problem derived from π_i
plan_d = plan that achieves all dead-end agent goals in π_i
= set of all contextual decompositions derived from π_i
Context_d = contextual decomposition in Σ
Gd = a dead-end agent goal set from a Context_d
π_t = tactical planning problem derived from a Gd set and π_i
π_r = relxaed tactical planning problem derived from a π_t
MA = macro action derived from a solved π_r
π_s = strategic planning problem derived from a Context_d and π_i
π_p = all parent agent goals planning problem derived from π_i and an all dead-end agent goals plan_d for π_i
plan_p = plan that achieves all parent agent goals in π_i
= component construction causality
= if problem succesfully solved (or relaxed plan obtained if π_t)
= if problem not solved / invalid plan (or relaxed plan not found if π_t)
= start new recursive step if parent agent goals of π_i are not achieved. π_p will be the instance planning problem of the RALSTP instance started by the new recursive step - π_p will be mapped to T(π)
Plan = plan mapped to T(π) at the start of a recursive step
Plan = intermediary plan used to construct sub-problems and plans
Plan_u = plan that achieves all goals in the unmodified original π
Stop = decomposition failed, proceed to next decomposition if available
π_i
π_d0
Plan_d0
π_p0
Plan_p0
Plan_i0
Plan_u0
All plans connecting root of T(π) to π_i
Σ
Landmarks and Agents Decomposition
Context_d1
Context_d2
Context_dn
Gd1
Gd2
Gdn
π_t1
π_t2
π_tn
π_r1
π_r2
π_rn
Stop
Plan π_r1
Plan π_r2
Plan π_rn
MA_1
MA_2
MA_n
π_s1
π_s2
π_sn
Plan π_s1
Plan π_s2
Plan π_sn
Plan_d1
Plan_d2
Plan_dn
π_p1
π_p2
π_pn
Plan_i1
Plan_i2
Plan_in
Plan_p1
Plan_p2
Plan_pn
Plan_u1
Plan_u2
Plan_un

Fig. 8.3 RALSTP Flowchart.

not correctly imported and situations where importing additional PDDL files following the initial import done while starting the code would make the code crash (segmentation fault). We also encountered issues with the legacy RPG analysis (particularly the removal of pointless effects) which added considerable algorithmic overhead to the landmarks extraction process in some cases. TPSHE also caused some issues while parsing PDDL files with numeric components. For every encountered issue, we had to implement workarounds with various degrees of efficiency that made the complete implementation of some of the algorithms described in this thesis best left for a new RALSTP implementation optimised and built from the ground up.

The RALSTP proof-of-concept is open-source and is available at the following GitHub repository: https://github.com/b-dorian/RALSTP

8.3 Limitations

In this section, we discuss the limitations of RALSTP as well as potential approaches for mitigating the limitations.

8.3.1 Problem Encodings Not Compatible with RALSTP

Our technique is suitable for planning problems that can be encoded in such a way that we can obtain an acyclic, directed and preferably weekly-connected dependency graph that explicitly contains all the agent types present in a problem that has dead-end agent goals. RALSTP does not bring any benefits to problems that do not have the above feature. Such problems have been immediately identified by RALSTP as incompatible during our evaluation.

Our empirical data shows that the cost of finding out if a planning problem was suitable for our solving technique was minimal (the cost of the problem compatibility check from the evaluated problems can be seen in the 'Agents' columns from the algorithmic overheads tables in the evaluations from Chapter 7), as we only needed to run the procedures for identifying and classifying the agents to determine if a problem had the feature we were looking to exploit (as described in Section 6.2 from Chapter 6). Upon failure to identify the desired structure (agent classification failed or there were no dead-end agent goals in Π_i or there were goals in Π_i that contain a dead-end agent as well as a parent agent or G in Π_i was not equal to the reunion of dead-end agent goals and parent agent goals derived from Π_i) the rest of the components in RALSTP were not invoked, so there is presumably no extra cost beyond the explicit feature probing to finding out if a solution search with RALSTP should be attempted for any specific problem. However, our thesis does not present a formal proof that guarantees a successful solution by using our decomposition technique for all planning problems that do contain the feature we are exploiting. Without such proof, we can not exclude the possibility of a domain engineered in a way that is compatible with our current feature identification protocol yet not solvable with our decomposition technique (such as a problem with unmitigated irreversible state transitions as described in Section 8.3.2). In such a case, our technique would spend fruitless time searching for a non-existent solution. On the other hand, if additional outlier domain structures are found in the future, then the current feature identification protocol could be potentially updated to exclude them. Additionally, even if such proof exists and would have been provided, guaranteeing that a temporal planning problem is solvable if it contains the exploited feature would still be infeasible as a temporal planning problem can in principle be undecidable [35].

Our work presents the evaluation of the IPC domains that were identified as compatible with RALSTP and excludes the rest, as our technique brings no benefit to incompatible problems. The compatible domains where the exploited feature is explicitly encoded are the Driverlog, ZenoTravel and RTAM IPC domains. Using RALSTP on incompatible IPC domains (or any other domains) brings no benefit, as the lack of an appropriate encoding prevents us from conducting an automatic exploitation. However, the encodings of the incompatible IPC domains could have well been made with the mentioned feature explicit and compatible with RALSTP in order to make the domains more practically interesting. For example, in the Rover domain, we had to explicitly define the *sample* agents for our procedure to work, even though the samples are implicit agents in the unmodified Rover domain. Our method brings no benefit to the Rover domain without the added manual encoding. However, with an encoding carefully constructed to expose the implicit agents, we have immediate access to huge benefits over other planners. The explicit encoding was also the reason we obtained superior results in the rest of the evaluated domains. There is nothing particularly special about the Driverlog, RTAM and Zenotravel domains other than they are encoded in a way that exposes the exploited feature in a compatible format with RALSTP in its current form, while the rest of the competition domains fail to achieve that.

On the other hand, we could probably create a more sophisticated analysis that identifies some of the implicit features automatically (TIM [29] would be the place to start for constructing such an analysis). This would give us access to the same exploitation as the explicit manually encoded versions of the feature in the Rovers domain. However, this was not the focus of the work presented in this thesis. What we were able to demonstrate with the manual encoding in the Rover domain is the potential savings that are out there to be made by either doing a comprehensive automatic analysis or by encouraging good practices in domain modelling to ensure these features are made more explicit.

We can not estimate the exact compatibility frequency of our technique in typical planning problems due to neither having a way of accurately determining what constitutes a typical planning domain nor a record of all real-world planning problems. However, the domains evaluated in Chapter 7 were not constructed artificially to fit our technique. They are, instead, pre-existing benchmark domains used in past international planning competitions. IPC benchmark domains are rather specific by design, as they are built for testing distinct aspects of AI planning instead of representations of typical planning problems. The fact that our technique is compatible with multiple distinct benchmark domains is a confirmation of its relevance in AI planning.

8.3.2 Irreversible State Transitions

Our procedure might not work if the initial state of the initial problem from the starting step or a specific recursive step cannot be reached from the final state of the tactical planning problems at the specific step before the final solution to the original problem is found. This happens because each tactical planning problem starts from the same initial state in the starting step and eventual recursive steps. For example, let's consider a version of Driverlog where you can only pass through each location once and in which we have only one driver, only one truck and two packages with corresponding goals reachable from the initial state. Let's also assume an individual tactical planning problem for each of the two packages. In this scenario, the strategic planning problem would be unsolvable, as the driver and truck parent agents will be prevented from returning to their starting locations once they depart the starting locations. This will cause one of the two macro actions corresponding to the two tactical planning problems to not be allowed to execute due to the parent agents' starting location preconditions not being met. An interesting observation is that, even though returning the agents to their initial state adds an extra cost to the solution, our procedure still achieves very good results. Potential fixes to the issue described above are to create an automatic procedure that re-encodes the problem in a way that allows the required state transitions or to encourage best practices in the domain design to avoid such issues for the problems where this is possible or to create a procedure for determining if a problem has irreversible state transitions as to not start a solving procedure for such problems.

8.4 Future Work

We have made several proposals for ways of improving the opportunities to exploit the decompositions based on deeper automated analysis as well as for using the newly introduced technical concepts in other areas of AI Planning.

8.4.1 Extend the Generality of the Approach

RALSTP was designed, implemented and tested to be compatible with temporal planning problems encoded with the PDDL 2.1 formalism. The technique as presented in this thesis is not applicable to problems encoded with non-temporal (classical) planning representations but could be modified to also support such problems (detailed in Section 8.4.1.1).

Our technique could also be potentially extended to additional versions of PDDL and planning representations. The use of PDDL+ processes and events [30] should be compatible with our technique provided that the problems which employ them do so in a way that can be contained within the decompositions employed by our techniques (similar to the applicability of temporal expressivity).

Planning representations that use negative preconditions should also be compatible with our technique as long as the problems we want to solve do not contain irreversible state transitions (detailed in Section 8.3.2).

Disjunctive preconditions should also not cause issues with our technique, as each action with such preconditions is effectively equivalent to multiple actions (one for each disjunct) and our technique does not have a bottleneck related to the number of actions in a problem. Relaxed landmarks (described in Section 4.3.2 from Chapter 4) are particularly useful in the case of disjunctive preconditions, as the landmarks relaxation can effectively reduce the disjunctive landmarks which might be extracted to relaxed landmarks that consist of a single proposition or event [42] (as shown in Example 4.65 from Chapter 4).

Conditional effects can be effectively compiled away with negative preconditions, so the former share the same level of compatibility with our technique as the latter.

Planning representations that use quantified effects might interfere with our technique if the quantified effects are in direct conflict with the decompositions. Even though such effects can be effectively compiled away, we could potentially encounter planning problems that are designed to not be able to achieve their solution without a quantified effect over all the agents in a way that doesn't allow splitting the agents among multiple sub-problems. However, problems with quantified effects that have less restrictive constraints which are compatible with our decomposition approach should be solvable with our technique, so support for quantified effects is also a potential area for future research.

Our decomposition techniques might also be applicable to hierarchical task networks, as our techniques effectively capture some of the natural structuring that domain engineers typically use in building HTN encodings. With our methods, such encoding can be created automatically or semi-automatically and with flexibility - as we can identify with minimal cost if a problem has a structure compatible with our decompositions and not push the technique if not suitable (described in Section 8.3.1).

8.4.1.1 Classical Planning Compatibility

The decompositions in RALSTP can be considered from a two-dimensional perspective. The horizontal decompositions stretch across the final solution from start to finish. These decompositions are sequential and represent the partial ordering among the dead-end agent goals and parent agent goals in the base case and in any eventual recursive step. Along the horizontal dimension, we have one or more orthogonal decompositions. These decompositions represent the agents and landmarks strategic tactical decompositions. Each orthogonal decomposition is designed to permit concurrency among its sub-problems.

Applying our technique to classical planning problems is straightforward provided we are no longer interested in concurrency among actions. For the horizontal decompositions, we simply consider the order of the obtained plans and merge them sequentially. For the orthogonal decompositions, we consider all macro-actions mutually exclusive and follow the order of the actions in the strategic plan when merging the regular (non-macro) actions from the strategic plan with the actions in the encapsulated tactical plans (corresponding to each macro-action in the strategic plan). On the other hand, potential conflicts among tactical plans could also be resolved using constraint satisfaction techniques (as described in Yang (2012)). However, problems that can only be solved in a temporal framework (such as the temporally expressive problems that can not be solved if we ignore the information about duration) would not be guaranteed a valid solution.

8.4.2 Using Global Constraints to Improve Solution Quality

The current heuristic for selecting the mandatory parent agents of the relaxed tactical planning problems disregards potential global constraints. A more comprehensive procedure for the mandatory parent agent selection prior to search that also takes into account some of the global constraints of the problem could be constructed. For example, we could construct a problem that contains all parent agents and abstractions of all dead-end agent goal sets which is designed to output in the relaxed plan specific distinct unique parent agent groups $\mu_{p\alpha_t}$ for solving each dead-end agent goal set. This way, the parent agent selection will potentially take into account some of the global constraints among all dead-end agent goals when assigning the mandatory parent agent groups to tactical planning problems.

8.4.3 Merging Local Constraints to Improve Solution Quality

Our technique could also be improved with a procedure that analyses the tactical planning problems and respective tactical plans to find potential cost reductions that do not violate any of the local constraints of each tactical planning problem. For example, two tactical plans that use the same parent agents for solving distinct dead-end agent goals are encapsulated in mutually exclusive macro-actions by the current technique. The actions in these plans are sequentially arranged in the final plan according to the order of the corresponding macro-actions in the strategic plan. However, executing these plans individually might yield similar states in which the parent agents interact with the environment and dead-end agents. In such a situation, the actions in the two plans could be merged into a single plan that might be more cost-effective than the sequential arrangement of the two tactical plans according to the order of the corresponding macro-actions in the strategic plan.

8.4.4 Agents and Dependencies Identification

We believe that the construction of more sophisticated automatic analyses for identifying the agents and agent dependencies of a problem is a promising area of future research. Our work has shown the benefits that can be obtained if we are able to determine agent-based partial orderings prior to the solution search. However, there are still problems that are incompatible with our agent identification and dependency extraction techniques due to the way they are encoded. On the other hand, each planning problem is, at its core, a collection of agents that interact with each other and with the environment, so the agent identification and a partial order among the agents could theoretically be obtained in most (if not all) planning problems (particularly in large-scale problems).

8.4.5 Encoding Validator and Best Practice Guide for Designing Domains

The Rovers domain is an example of how a minor change in the PDDL encoding makes the domain compatible with our technique and brings huge benefits in the quality of the solutions (as shown in the evaluation from Chapter 7). Considering the above, another potential area of future research is the creation of a best practice guide for domain design. The guide could be used by domain engineers to encode problems in a way that prevents conflicting dependency relationships from appearing in problems or that allows the mitigation of eventual conflicting dependency relationships. The guide should focus on design choices that create problems from which we can extract an acyclic, directed and preferably weekly-connected dependency graph that explicitly contains all the agent types present in the problem. The guide could be combined with our cost-effective procedure for identifying and classifying the agents (described in Section 6.2 from Chapter 6) which would act as an *Encoding Validator* that determines if a problem was encoded in a way that makes the feature we are exploiting explicit.

8.4.6 Agents and Relaxed Landmarks Heuristics, Abstractions and Decompositions

Our empirical data shows that the heuristics, decompositions and abstractions presented in this thesis are effective in finding quality solutions to large-scale problems. However, additional heuristics, decompositions and abstractions can be potentially created using the agents, relaxed landmarks as well as relaxed propositions and relaxed events. New constructs (such as the abstraction of all dead-end agent goal sets described in Section 8.4.2) could potentially further decrease the solving difficulty of large-scale problems and further improve the quality of the obtained solutions.

Agents as defined in our work could potentially be used along existing heuristics (or potentially open a new class of heuristics) to improve the search operation of planners. For example, our cost-effective agents identification and classification procedure (described in Section 6.2 from Chapter 6) could be incorporated into the pre-processing of heuristic planners in order to use agent, facts, and goals dead-end vs parent classifications to fine-tune, back-track or act as a tie-break for the planner decision process when states problematic for the employed heuristic are reached.

The relaxed landmarks could also potentially be used along existing heuristics (or potentially open a new class of heuristics) to improve the search operation of planners. For example, the relaxed landmarks could be incorporated into existing techniques that use heuristics constructed from regular landmarks as a guide for the solution search (such as the LAMA planner [50]) to potentially improve the efficiency of the landmarks-based heuristics.

8.4.7 Using Difficultly Metrics for Determining the Appropriate Solving Approach

Our empirical analysis reconfirms that the use of AI planning beyond demonstration examples is challenging for expressive problems with numerous components. Heuristics planners are good for small-scale precision but are not ideal for efficiently navigating large state spaces. However, using decompositions is also not without potential risks, as the more you isolate the components of a problem and disregard global constraints the more you risk degrading the quality of the obtained solution. Therefore, appreciating when a problem is difficult enough that employing decompositions would be beneficial is a potential area of future research. The evaluation in Chapter 7 shows a strong correlation between the $N(\alpha)$ and $N(\Phi)$ difficulty metrics (described in Chapter 5) and the effectiveness of decompositions of problems from the same domain. Our metrics are cheap to compute and could be used at the pre-processing stage to determine the appropriate solving technique for a specific planning problem.

8.5 Conclusions

Using decompositions in AI planning carries the risk of constraints becoming stronger due to the choice restrictions of certain variables. However, our thesis shows that, in large-scale problems, using current planners without a decomposition strategy is not effective. After a certain difficulty threshold, the solving approaches employed by existing planners either fail or start making costly trade-offs (such as using only one agent or one unique parent agent group for solving the whole problem) that result in poor-quality solutions in comparison to the solutions obtained by our abstracted decompositions technique. The success of heuristic planners in the past years has drawn focus away from using decompositions in AI planning. However, our thesis shows that abstracted decompositions might be the lesser of evils in large-scale planning if correctly applied. Our results point out that abstracted decomposition techniques should be refocused on if we aim to increase the use of AI planning beyond demonstration examples.

The emphasis of this thesis has been on the role of agents and landmarks in a planning problem and how they can be applied to recognise and extract key planning problem components, properties, and metrics that enable efficient data-driven algorithmic decompositions and abstractions. The resulting technique improves the scale and solution quality of solvable planning problems using decomposition methods that attempt to simulate human intuition. An interesting remark is that writing our thesis consisted of decomposing and partially ordering vast quantities of information obtained from our experiments, which more often than not resembled the actual large task decomposition approaches described by our thesis.

Our work introduced new AI planning technical concepts along with automated extraction procedures that output data utilised as "advice" for efficiently decomposing and abstracting planning problems. These concepts consist of formal definitions for the agents, the agent dependency relationships and classifications, the necessary and unnecessary static environment as well as for the relaxed landmarks, propositions and events.

We described a new framework for evaluating the difficulty of a planning problem according to object-based difficulty metrics such as the number of agents and inactive dynamic objects, the number of instantiated dynamic types and the number of necessary and unnecessary static objects entangled in a planning problem.

Our work presented a detailed description of a new fully automated data-driven recursive agents and landmarks strategic-tactical planning decomposition and abstraction procedure (RALSTP) that significantly increases the solution quality of solvable planning problems. The procedure uses a novel goal clustering method based on the regular and relaxed landmarks found in common between the individual backchaining of each top-level goal. We have also shown that the abstractions and relaxations in our technique can exponentially increase the scale of solvable planning problems that have a large state space due to numerous entangled parent types with multiple parent agent instances for each parent type.

A proof-of-concept implementation of our thesis was built across multiple C++ codebases and was made publicly available as open-source software. The implementation was used to perform an evaluation of the decomposition and abstraction technique described in our thesis on IPC benchmark problems. The evaluation showed the benefits in scale and solution quality of our method in comparison to the solutions obtained by a broad range of state-of-the-art temporal planners. Furthermore, we presented promising areas of future research enabled by the new concepts and techniques introduced by our thesis.

If the project had to be done from the beginning, the implementation for the practical experimentation would start with the latest version of VAL as the base layer, to which the landmarks extraction part from the Karpas et al. (2015) codebase would be added. A new PDDL/RPG parser optimised for RALSTP might also be considered if an industry-grade implementation of RALSTP is required.

References

[1] Asai, M. and Fukunaga, A. (2014). Fully automated cyclic planning for large-scale manufacturing domains. In *Twenty-Fourth International Conference on Automated Planning and Scheduling*.

[2] Asai, M. and Fukunaga, A. (2015). Solving large-scale planning problems by decomposition and macro generation. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 25, pages 16–24.

[3] Bacchus, F. (2001). AIPS 2000 planning competition: The fifth international conference on artificial intelligence planning and scheduling systems. *AI magazine*, 22(3):47–47.

[4] Bacchus, F. and Yang, Q. (1991). The downward refinement property. In *IJCAI*, pages 286–293.

[5] Bacchus, F. and Yang, Q. (1994). Downward refinement and the efficiency of hierarchical problem solving. *Artificial Intelligence*, 71(1):43–100.

[6] Benton, J., Coles, A., and Coles, A. (2012). Temporal planning with preferences and time-dependent continuous costs. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 22, pages 2–10.

[7] Blum, A. L. and Furst, M. L. (1997). Fast planning through planning graph analysis. *Artificial intelligence*, 90(1-2):281–300.

[8] Bonet, B. and Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33.

[9] Bonnet, B. and Geffner, H. (1998). HSP: Heuristic search planner. *AIPS-98 Planning Competition*.

[10] Botea, A. (2006). Improving AI planning and search with automatic abstraction. *Thesis (Ph.D.), University of Alberta*.

[11] Botea, A., Enzenberger, M., Müller, M., and Schaeffer, J. (2005). Macro-FF: Improving AI planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research*, 24:581–621.

[12] Brailsford, S. C., Potts, C. N., and Smith, B. M. (1999). Constraint satisfaction problems: Algorithms and applications. *European journal of operational research*, 119(3):557–581.

[13] Brassard, G. and Bratley, P. (1996). *Fundamentals of algorithmics*. Prentice-Hall, Inc.

[14] Brooks, R. A. (1991). Intelligence without representation. *Artificial intelligence*, 47(1-3):139–159.

[15] Buksz, D., Cashmore, M., Krarup, B., Magazzeni, D., and Ridder, B. (2018). Strategic-tactical planning for autonomous underwater vehicles over long horizons. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3565–3572. IEEE.

[16] Buksz, D., Mujumdar, A., Orlić, M., Mohalik, S., Daoutis, M., Badrinath, R., Magazzeni, D., Cashmore, M., and Feljan, A. V. (2020). Intent-driven strategic tactical planning for autonomous site inspection using cooperative drones. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6733–6740. IEEE.

[17] Carreno, Y., Pairet, È., Petillot, Y., and Petrick, R. P. (2020). A decentralised strategy for heterogeneous auv missions via goal distribution and temporal planning. In *Proceedings of the international conference on automated planning and scheduling*, volume 30, pages 431–439.

[18] Cashmore, M., Fox, M., Larkworthy, T., Long, D., and Magazzeni, D. (2014). AUV mission control via temporal planning. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 6535–6541. IEEE.

[19] Cashmore, M., Fox, M., Long, D., Magazzeni, D., Ridder, B., Carrera, A., Palomeras, N., Hurtos, N., and Carreras, M. (2015). Rosplan: Planning in the robot operating system. In *Proceedings of the international conference on automated planning and scheduling*, volume 25, pages 333–341.

[20] Chrpa, L. (2010). Generation of macro-operators via investigation of action dependencies in plans. *The Knowledge Engineering Review*, 25(3):281–297.

[21] Coles, A., Coles, A., Fox, M., and Long, D. (2010). Forward-chaining partial-order planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 20, pages 42–49.

[22] Coles, A. I. and Smith, A. J. (2007). Marvin: A heuristic search planner with online macro-action learning. *Journal of Artificial Intelligence Research*, 28:119–156.

[23] Coles, A. J., Coles, A. I., Fox, M., and Long, D. (2012). COLIN: Planning with continuous linear numeric change. *Journal of Artificial Intelligence Research*, 44:1–96.

[24] Crosby, M., Rovatsos, M., and Petrick, R. (2013). Automated agent decomposition for classical planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 23, pages 46–54.

[25] Erol, K., Hendler, J., and Nau, D. S. (1994). HTN planning: Complexity and expressivity. In *AAAI*, volume 94, pages 1123–1128.

[26] Eyerich, P., Mattmüller, R., and Röger, G. (2012). Using the context-enhanced additive heuristic for temporal and numeric planning. In *Towards Service Robots for Everyday Environments: Recent Advances in Designing Service Robots for Complex Tasks in Everyday Environments*, pages 49–64. Springer.

[27] Fawcett, C., Helmert, M., Hoos, H., Karpas, E., Röger, G., and Seipp, J. (2011). Fd-autotune: Domain-specific configuration using fast downward. In *ICAPS 2011 Workshop on Planning and Learning*, pages 13–17.

[28] Fikes, R. E. and Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208.

[29] Fox, M. and Long, D. (1998). The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research*, 9:367–421.

[30] Fox, M. and Long, D. (2002). PDDL+: Modeling continuous time dependent effects. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, volume 4, page 34.

[31] Fox, M. and Long, D. (2003). PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of artificial intelligence research*, 20:61–124.

[32] Furelos Blanco, D., Jonsson, A., Palacios Verdes, H. L., and Jiménez, S. (2018). Forward-search temporal planning with simultaneous events. In *Salido MA, Barták R, editors. COPLAS 2018. 13th Workshop on Constraint Satisfaction Techniques for Planning and Scheduling; 2018 Jun 24-29; Delft, the Netherlands. Palo Alto (CA): AAAI; 2018. p. 11-20.* Association for the Advancement of Artificial Intelligence (AAAI)-Congress.

[33] Geffner, T. and Geffner, H. (2015). Width-based planning for general video-game playing. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 11, pages 23–29.

[34] Gerevini, A., Saetti, A., and Vallati, M. (2009). An automatically configurable portfolio-based planner with macro-actions: Pbp. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 19, pages 350–353.

[35] Gigante, N., Micheli, A., Montanari, A., and Scala, E. (2020). Decidability and complexity of action-based temporal planning over dense time. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 9859–9866.

[36] Helmert, M. (2002). Decidability and undecidability results for planning with numerical state variables. In *AIPS*, pages 44–53.

[37] Hoffmann, J. (2001). FF: The fast-forward planning system. *AI magazine*, 22(3):57–57.

[38] Hoffmann, J., Porteous, J., and Sebastia, L. (2004). Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, 22:215–278.

[39] Hofmann, T., Niemueller, T., and Lakemeyer, G. (2017). Initial results on generating macro actions from a plan database for planning on autonomous mobile robots. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 27, pages 498–503.

[40] Howe, A., Knoblock, C., McDermott, I. D., Ram, A., Veloso, M., Weld, D., SRI, D. W., Barrett, A., Christianson, D., et al. (1998). PDDL | the planning domain definition language. *Technical Report, Tech. Rep.*

[41] Howey, R., Long, D., and Fox, M. (2004). VAL: Automatic plan validation, continuous effects and mixed initiative planning using PDDL. In *16th IEEE International Conference on Tools with Artificial Intelligence*, pages 294–301. IEEE.

[42] Karpas, E., Wang, D., Williams, B., and Haslum, P. (2015). Temporal landmarks: What must happen, and when. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 25.

[43] Knoblock, C. A. (1994). Automatically generating abstractions for planning. *Artificial intelligence*, 68(2):243–302.

[44] Lipovetzky, N., Ramirez, M., and Geffner, H. (2015). Classical planning with simulators: Results on the atari video games. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

[45] Long, D., Kautz, H., Selman, B., Bonet, B., Geffner, H., Koehler, J., Brenner, M., Hoffmann, J., Rittinger, F., Anderson, C. R., et al. (2000). The AIPS-98 planning competition. *AI magazine*, 21(2):13–13.

[46] Marzal, E., Sebastia, L., and Onaindia, E. (2014). On the use of temporal landmarks for planning with deadlines. In *Twenty-Fourth International Conference on Automated Planning and Scheduling*.

[47] Nau, D., Munoz-Avila, H., Cao, Y., Lotem, A., and Mitchell, S. (2001). Total-order planning with partially ordered subtasks. In *IJCAI*, volume 1, pages 425–430.

[48] Nilsson, N. J. (1990). Triangle-table trees. *Proposal to the Army Research Office, the Air Force Office of Scientific Research, and the National Aeronautics and Space Administration*.

[49] Rankooh, M. F. and Ghassem-Sani, G. (2015). ITSAT: an efficient sat-based temporal planner. *Journal of Artificial Intelligence Research*, 53:541–632.

[50] Richter, S. and Westphal, M. (2010). The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39:127–177.

[51] Ridder, B. and Fox, M. (2014). Heuristic evaluation based on lifted relaxed planning graphs. In *Twenty-Fourth International Conference on Automated Planning and Scheduling*.

[52] Sacerdoti, E. D. (1974). Planning in a hierarchy of abstraction spaces. *Artificial intelligence*, 5(2):115–135.

[53] Sapena, O., Marzal, E., and Onaindia, E. (2018). TFLAP: a temporal forward partial-order planner. *URL https://ipc2018-temporal. bitbucket. io/planner-abstracts/team2. pdf*.

[54] Sebastia, L., Onaindia, E., and Marzal, E. (2006). Decomposition of planning problems. *AI Communications*, 19(1):49–81.

[55] Shah, M. M. S., Chrpa, L., Kitchin, D. E., McCluskey, T. L., and Vallati, M. (2013). Exploring knowledge engineering strategies in designing and modelling a road traffic accident management domain. AAAI Press/International Joint Conferences on Artificial Intelligence.

[56] Shleyfman, A., Katz, M., Helmert, M., Sievers, S., and Wehrle, M. (2015). Heuristics and symmetries in classical planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29.

[57] Simpson, R., Long, D., McCluskey, T., and Fox, M. (2002). Generic types as design patterns for planning domain specifications.

[58] Simpson, R. M., McCluskey, T. L., Liu, D., and Kitchin, D. E. (2000). Knowledge representation in planning: a PDDL to OCL h translation. In *International Symposium on Methodologies for Intelligent Systems*, pages 610–618. Springer.

[59] Srivastava, B. (2000). Realplan: Decoupling causal and resource reasoning in planning. In *AAAI/IAAI*, pages 812–818.

[60] Štolba, M., Komenda, A., and Kovacs, D. (2016). Competition of distributed and multiagent planners (codmap). In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

[61] Tenenberg, J. (1988). Abstraction in planning.

[62] Tunstel, E., Huntsberger, T., Aghazarian, H., Backes, P., Baumgartner, E., Cheng, Y., Garrett, M., Kennedy, B., Leger, C., Magnone, L., et al. (2002). FIDO rover field trials as rehearsal for the NASA 2003 mars exploration rovers mission. In *Proceedings of the 5th Biannual World Automation Congress*, volume 14, pages 320–327. IEEE.

[63] Vallati, M., Chrpa, L., Grześ, M., McCluskey, T. L., Roberts, M., Sanner, S., et al. (2015). The 2014 international planning competition: Progress and trends. *AI Magazine*, 36(3):90–98.

[64] Vidal, V. (2014). YAHSP3 and YAHSP3-MT in the 8th international planning competition. *Proceedings of the 8th International Planning Competition (IPC-2014)*, pages 64–65.

[65] Wang, D. and Williams, B. (2015). tburton: A divide and conquer temporal planner. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29.

[66] Winner, E. and Veloso, M. M. (2003). Distill: Learning domain-specific planners by example. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 800–807.

[67] Yang, Q. (2012). *Intelligent planning: a decomposition and abstraction based approach.* Springer Science & Business Media.

Appendix A

Appendix

A.1 PDDL Files

A.1.1 Driverlog

A.1.1.1 Driverlog Domain

```
(define (domain driverlog)
(:requirements :typing :durative-actions)
(:types location locatable - object
driver truck obj - locatable)

(:predicates
(at ?obj - locatable ?loc - location)
(in ?obj1 - obj ?obj - truck)
(driving ?d - driver ?v - truck)
(link ?x ?y - location) (path ?x ?y - location)
(empty ?v - truck)
)

(:durative-action LOAD-TRUCK
:parameters
(?obj - obj
?truck - truck
?loc - location)
:duration (= ?duration 2)
:condition
(and
(over all (at ?truck ?loc)) (at start (at ?obj ?loc)))
:effect
(and (at start (not (at ?obj ?loc))) (at end (in ?obj ?truck))))

(:durative-action UNLOAD-TRUCK
:parameters
(?obj - obj
?truck - truck
?loc - location)
:duration (= ?duration 2)
:condition
(and
(over all (at ?truck ?loc)) (at start (in ?obj ?truck)))
:effect
(and (at start (not (in ?obj ?truck))) (at end (at ?obj ?loc))))

(:durative-action BOARD-TRUCK
:parameters
(?driver - driver
?truck - truck
?loc - location)
:duration (= ?duration 1)
:condition
(and
(over all (at ?truck ?loc)) (at start (at ?driver ?loc))
(at start (empty ?truck)))
:effect
(and (at start (not (at ?driver ?loc)))
(at end (driving ?driver ?truck)) (at start (not (empty ?truck)))))

(:durative-action DISEMBARK-TRUCK
:parameters
(?driver - driver
?truck - truck
?loc - location)
:duration (= ?duration 1)
:condition
(and (over all (at ?truck ?loc)) (at start (driving ?driver ?truck)))
```

```
:effect
(and (at start (not (driving ?driver ?truck)))
(at end (at ?driver ?loc)) (at end (empty ?truck))))

(:durative-action DRIVE-TRUCK
:parameters
(?truck - truck
?loc-from - location
?loc-to - location
?driver - driver)
:duration (= ?duration 10)
:condition
(and (at start (at ?truck ?loc-from))
(over all (driving ?driver ?truck)) (at start (link ?loc-from ?loc-to)))
:effect
(and (at start (not (at ?truck ?loc-from)))
(at end (at ?truck ?loc-to))))

(:durative-action WALK
:parameters
(?driver - driver
?loc-from - location
?loc-to - location)
:duration (= ?duration 20)
:condition
(and (at start (at ?driver ?loc-from))
(at start (path ?loc-from ?loc-to)))
:effect
(and (at start (not (at ?driver ?loc-from)))
(at end (at ?driver ?loc-to))))

)
```

A.1.1.2 DLOG-5-5-10 Problem

```
(define (problem DLOG-5-5-10)
(:domain driverlog)
(:objects
driver1 - driver
driver2 - driver
driver3 - driver
driver4 - driver
driver5 - driver
truck1 - truck
truck2 - truck
truck3 - truck
truck4 - truck
truck5 - truck
package1 - obj
package2 - obj
package3 - obj
package4 - obj
package5 - obj
package6 - obj
package7 - obj
package8 - obj
package9 - obj
package10 - obj
s0 - location
s1 - location
s2 - location
s3 - location
s4 - location
s5 - location
s6 - location
s7 - location
s8 - location
s9 - location
p0-4 - location
p0-5 - location
p1-6 - location
```

```
p2-0 - location
p2-4 - location
p3-2 - location
p3-8 - location
p4-1 - location
p4-3 - location
p5-6 - location
p5-7 - location
p5-9 - location
p6-4 - location
p7-0 - location
p8-4 - location
p9-5 - location
p9-7 - location
)
(:init
(at driver1 s9)
(at driver2 s1)
(at driver3 s5)
(at driver4 s7)
(at driver5 s3)
(at truck1 s0)
(empty truck1)
(at truck2 s1)
(empty truck2)
(at truck3 s9)
(empty truck3)
(at truck4 s7)
(empty truck4)
(at truck5 s8)
(empty truck5)
(at package1 s1)
(at package2 s3)
(at package3 s9)
(at package4 s0)
(at package5 s5)
(at package6 s2)
(at package7 s2)
```

```
(at package8 s6)
(at package9 s4)
(at package10 s1)
(path s0 p0-4)
(path p0-4 s0)
(path s4 p0-4)
(path p0-4 s4)
(path s0 p0-5)
(path p0-5 s0)
(path s5 p0-5)
(path p0-5 s5)
(path s1 p1-6)
(path p1-6 s1)
(path s6 p1-6)
(path p1-6 s6)
(path s2 p2-0)
(path p2-0 s2)
(path s0 p2-0)
(path p2-0 s0)
(path s2 p2-4)
(path p2-4 s2)
(path s4 p2-4)
(path p2-4 s4)
(path s3 p3-2)
(path p3-2 s3)
(path s2 p3-2)
(path p3-2 s2)
(path s3 p3-8)
(path p3-8 s3)
(path s8 p3-8)
(path p3-8 s8)
(path s4 p4-1)
(path p4-1 s4)
(path s1 p4-1)
(path p4-1 s1)
(path s4 p4-3)
(path p4-3 s4)
(path s3 p4-3)
```

```
(path p4-3 s3)
(path s5 p5-6)
(path p5-6 s5)
(path s6 p5-6)
(path p5-6 s6)
(path s5 p5-7)
(path p5-7 s5)
(path s7 p5-7)
(path p5-7 s7)
(path s5 p5-9)
(path p5-9 s5)
(path s9 p5-9)
(path p5-9 s9)
(path s6 p6-4)
(path p6-4 s6)
(path s4 p6-4)
(path p6-4 s4)
(path s7 p7-0)
(path p7-0 s7)
(path s0 p7-0)
(path p7-0 s0)
(path s8 p8-4)
(path p8-4 s8)
(path s4 p8-4)
(path p8-4 s4)
(path s9 p9-7)
(path p9-7 s9)
(path s7 p9-7)
(path p9-7 s7)
(link s0 s7)
(link s7 s0)
(link s1 s2)
(link s2 s1)
(link s1 s3)
(link s3 s1)
(link s1 s6)
(link s6 s1)
(link s1 s8)
```

```
(link s8 s1)
(link s1 s9)
(link s9 s1)
(link s2 s0)
(link s0 s2)
(link s2 s5)
(link s5 s2)
(link s2 s6)
(link s6 s2)
(link s2 s7)
(link s7 s2)
(link s2 s9)
(link s9 s2)
(link s3 s0)
(link s0 s3)
(link s3 s8)
(link s8 s3)
(link s4 s0)
(link s0 s4)
(link s4 s1)
(link s1 s4)
(link s4 s8)
(link s8 s4)
(link s4 s9)
(link s9 s4)
(link s5 s4)
(link s4 s5)
(link s5 s7)
(link s7 s5)
(link s5 s9)
(link s9 s5)
(link s6 s3)
(link s3 s6)
(link s6 s7)
(link s7 s6)
(link s9 s6)
(link s6 s9)
(link s9 s8)
```

```
(link s8 s9)
)
(:goal (and
(at driver2 s0)
(at driver4 s1)
(at driver5 s9)
(at truck2 s9)
(at truck3 s5)
(at truck4 s7)
(at truck5 s3)
(at package2 s4)
(at package4 s7)
(at package5 s1)
(at package6 s1)
(at package7 s2)
(at package8 s1)
(at package9 s9)
(at package10 s7)
))

(:metric minimize (total-time))

)
```

A.1.1.3 DLOG-7-7-16 Problem

```
(define (problem DLOG-7-7-16)
(:domain driverlog)
(:objects
driver1 - driver
driver2 - driver
driver3 - driver
driver4 - driver
driver5 - driver
driver6 - driver
driver7 - driver
```

```
truck1 - truck
truck2 - truck
truck3 - truck
truck4 - truck
truck5 - truck
truck6 - truck
truck7 - truck
package1 - obj
package2 - obj
package3 - obj
package4 - obj
package5 - obj
package6 - obj
package7 - obj
package8 - obj
package9 - obj
package10 - obj
package11 - obj
package12 - obj
package13 - obj
package14 - obj
package15 - obj
package16 - obj
s0 - location
s1 - location
s2 - location
s3 - location
s4 - location
s5 - location
s6 - location
s7 - location
s8 - location
s9 - location
s10 - location
s11 - location
s12 - location
s13 - location
s14 - location
```

```
s15 - location
s16 - location
s17 - location
p0-8 - location
p1-8 - location
p1-15 - location
p2-12 - location
p3-4 - location
p3-11 - location
p4-6 - location
p4-8 - location
p5-8 - location
p6-9 - location
p6-15 - location
p7-1 - location
p7-2 - location
p7-5 - location
p7-17 - location
p9-10 - location
p9-12 - location
p10-11 - location
p10-13 - location
p10-14 - location
p11-7 - location
p13-4 - location
p13-16 - location
p14-0 - location
p14-3 - location
p14-7 - location
p14-17 - location
p15-8 - location
p16-0 - location
p16-7 - location
p16-10 - location
p16-12 - location
p17-1 - location
)
(:init
(at driver1 s13)
(at driver2 s14)
(at driver3 s9)
(at driver4 s7)
(at driver5 s8)
(at driver6 s13)
(at driver7 s3)
(at truck1 s9)
(empty truck1)
(at truck2 s11)
(empty truck2)
(at truck3 s11)
(empty truck3)
(at truck4 s16)
(empty truck4)
(at truck5 s6)
(empty truck5)
(at truck6 s1)
(empty truck6)
(at truck7 s10)
(empty truck7)
(at package1 s2)
(at package2 s16)
(at package3 s11)
(at package4 s14)
(at package5 s8)
(at package6 s15)
(at package7 s0)
(at package8 s4)
(at package9 s13)
(at package10 s17)
(at package11 s7)
(at package12 s5)
(at package13 s7)
(at package14 s12)
(at package15 s0)
(at package16 s10)
(path s0 p0-8)
```

```
(path p0-8 s0)
(path s8 p0-8)
(path p0-8 s8)
(path s1 p1-8)
(path p1-8 s1)
(path s8 p1-8)
(path p1-8 s8)
(path s1 p1-15)
(path p1-15 s1)
(path s15 p1-15)
(path p1-15 s15)
(path s2 p2-12)
(path p2-12 s2)
(path s12 p2-12)
(path p2-12 s12)
(path s3 p3-4)
(path p3-4 s3)
(path s4 p3-4)
(path p3-4 s4)
(path s3 p3-11)
(path p3-11 s3)
(path s11 p3-11)
(path p3-11 s11)
(path s4 p4-6)
(path p4-6 s4)
(path s6 p4-6)
(path p4-6 s6)
(path s4 p4-8)
(path p4-8 s4)
(path s8 p4-8)
(path p4-8 s8)
(path s5 p5-8)
(path p5-8 s5)
(path s8 p5-8)
(path p5-8 s8)
(path s6 p6-9)
(path p6-9 s6)
(path s9 p6-9)
```

```
(path p6-9 s9)
(path s6 p6-15)
(path p6-15 s6)
(path s15 p6-15)
(path p6-15 s15)
(path s7 p7-1)
(path p7-1 s7)
(path s1 p7-1)
(path p7-1 s1)
(path s7 p7-2)
(path p7-2 s7)
(path s2 p7-2)
(path p7-2 s2)
(path s7 p7-5)
(path p7-5 s7)
(path s5 p7-5)
(path p7-5 s5)
(path s7 p7-17)
(path p7-17 s7)
(path s17 p7-17)
(path p7-17 s17)
(path s9 p9-10)
(path p9-10 s9)
(path s10 p9-10)
(path p9-10 s10)
(path s9 p9-12)
(path p9-12 s9)
(path s12 p9-12)
(path p9-12 s12)
(path s10 p10-11)
(path p10-11 s10)
(path s11 p10-11)
(path p10-11 s11)
(path s10 p10-13)
(path p10-13 s10)
(path s13 p10-13)
(path p10-13 s13)
(path s10 p10-14)
```

```
(path p10-14 s10)
(path s14 p10-14)
(path p10-14 s14)
(path s11 p11-7)
(path p11-7 s11)
(path s7 p11-7)
(path p11-7 s7)
(path s13 p13-4)
(path p13-4 s13)
(path s4 p13-4)
(path p13-4 s4)
(path s13 p13-16)
(path p13-16 s13)
(path s16 p13-16)
(path p13-16 s16)
(path s14 p14-0)
(path p14-0 s14)
(path s0 p14-0)
(path p14-0 s0)
(path s14 p14-3)
(path p14-3 s14)
(path s3 p14-3)
(path p14-3 s3)
(path s14 p14-7)
(path p14-7 s14)
(path s7 p14-7)
(path p14-7 s7)
(path s14 p14-17)
(path p14-17 s14)
(path s17 p14-17)
(path p14-17 s17)
(path s15 p15-8)
(path p15-8 s15)
(path s8 p15-8)
(path p15-8 s8)
(path s16 p16-0)
(path p16-0 s16)
(path s0 p16-0)
```

```
(path p16-0 s0)
(path s16 p16-7)
(path p16-7 s16)
(path s7 p16-7)
(path p16-7 s7)
(path s16 p16-10)
(path p16-10 s16)
(path s10 p16-10)
(path p16-10 s10)
(path s16 p16-12)
(path p16-12 s16)
(path s12 p16-12)
(path p16-12 s12)
(path s17 p17-1)
(path p17-1 s17)
(path s1 p17-1)
(path p17-1 s1)
(link s0 s1)
(link s1 s0)
(link s0 s9)
(link s9 s0)
(link s0 s13)
(link s13 s0)
(link s1 s3)
(link s3 s1)
(link s1 s6)
(link s6 s1)
(link s1 s13)
(link s13 s1)
(link s1 s15)
(link s15 s1)
(link s2 s3)
(link s3 s2)
(link s2 s5)
(link s5 s2)
(link s2 s8)
(link s8 s2)
(link s2 s11)
```

```
(link s11 s2)
(link s2 s13)
(link s13 s2)
(link s2 s16)
(link s16 s2)
(link s2 s17)
(link s17 s2)
(link s3 s9)
(link s9 s3)
(link s4 s0)
(link s0 s4)
(link s4 s2)
(link s2 s4)
(link s4 s3)
(link s3 s4)
(link s4 s10)
(link s10 s4)
(link s4 s11)
(link s11 s4)
(link s4 s13)
(link s13 s4)
(link s5 s0)
(link s0 s5)
(link s5 s12)
(link s12 s5)
(link s5 s14)
(link s14 s5)
(link s5 s16)
(link s16 s5)
(link s6 s0)
(link s0 s6)
(link s6 s11)
(link s11 s6)
(link s6 s15)
(link s15 s6)
(link s6 s17)
(link s17 s6)
(link s7 s1)
```

```
(link s1 s7)
(link s7 s2)
(link s2 s7)
(link s7 s15)
(link s15 s7)
(link s7 s16)
(link s16 s7)
(link s8 s0)
(link s0 s8)
(link s8 s7)
(link s7 s8)
(link s8 s9)
(link s9 s8)
(link s8 s14)
(link s14 s8)
(link s8 s16)
(link s16 s8)
(link s8 s17)
(link s17 s8)
(link s9 s2)
(link s2 s9)
(link s9 s7)
(link s7 s9)
(link s9 s12)
(link s12 s9)
(link s10 s0)
(link s0 s10)
(link s10 s2)
(link s2 s10)
(link s10 s12)
(link s12 s10)
(link s10 s17)
(link s17 s10)
(link s11 s0)
(link s0 s11)
(link s11 s13)
(link s13 s11)
(link s12 s6)
```

```
(link s6 s12)
(link s12 s15)
(link s15 s12)
(link s14 s11)
(link s11 s14)
(link s14 s17)
(link s17 s14)
(link s15 s2)
(link s2 s15)
(link s16 s9)
(link s9 s16)
(link s16 s15)
(link s15 s16)
(link s17 s12)
(link s12 s17)
)
(:goal (and
(at driver1 s0)
(at driver2 s5)
(at driver3 s12)
(at driver4 s4)
(at driver5 s7)
(at driver6 s3)
(at truck1 s1)
(at truck2 s3)
(at truck5 s8)
(at truck7 s8)
(at package1 s16)
(at package2 s11)
(at package3 s15)
(at package4 s6)
(at package5 s0)
(at package6 s12)
(at package7 s10)
(at package8 s17)
(at package9 s1)
(at package10 s12)
(at package11 s14)
(at package12 s13)
(at package13 s7)
(at package14 s3)
(at package15 s12)
(at package16 s9)
))

(:metric minimize (total-time))

)
```

A.1.1.4 DLOG-8-8-19 Problem

```
(define (problem DLOG-8-8-19)
(:domain driverlog)
(:objects
driver1 - driver
driver2 - driver
driver3 - driver
driver4 - driver
driver5 - driver
driver6 - driver
driver7 - driver
driver8 - driver
truck1 - truck
truck2 - truck
truck3 - truck
truck4 - truck
truck5 - truck
truck6 - truck
truck7 - truck
truck8 - truck
package1 - obj
package2 - obj
package3 - obj
package4 - obj
```

```
package5 - obj
package6 - obj
package7 - obj
package8 - obj
package9 - obj
package10 - obj
package11 - obj
package12 - obj
package13 - obj
package14 - obj
package15 - obj
package16 - obj
package17 - obj
package18 - obj
package19 - obj
s0 - location
s1 - location
s2 - location
s3 - location
s4 - location
s5 - location
s6 - location
s7 - location
s8 - location
s9 - location
s10 - location
s11 - location
s12 - location
s13 - location
s14 - location
s15 - location
s16 - location
s17 - location
s18 - location
s19 - location
s20 - location
s21 - location
p0-1 - location
```

```
p1-2 - location
p1-10 - location
p1-15 - location
p2-0 - location
p2-3 - location
p2-10 - location
p2-14 - location
p2-19 - location
p3-7 - location
p3-13 - location
p5-10 - location
p5-11 - location
p6-5 - location
p7-8 - location
p8-2 - location
p8-6 - location
p8-12 - location
p8-18 - location
p8-21 - location
p9-1 - location
p10-6 - location
p10-9 - location
p10-11 - location
p11-15 - location
p12-13 - location
p12-16 - location
p12-17 - location
p13-9 - location
p14-0 - location
p14-4 - location
p16-0 - location
p16-5 - location
p16-15 - location
p17-0 - location
p17-5 - location
p17-9 - location
p17-20 - location
p19-10 - location
```

```
p19-13 - location
p20-0 - location
p20-1 - location
p20-9 - location
p20-15 - location
p21-9 - location
)
(:init
(at driver1 s14)
(at driver2 s17)
(at driver3 s10)
(at driver4 s18)
(at driver5 s0)
(at driver6 s4)
(at driver7 s16)
(at driver8 s20)
(at truck1 s9)
(empty truck1)
(at truck2 s6)
(empty truck2)
(at truck3 s8)
(empty truck3)
(at truck4 s15)
(empty truck4)
(at truck5 s0)
(empty truck5)
(at truck6 s12)
(empty truck6)
(at truck7 s1)
(empty truck7)
(at truck8 s6)
(empty truck8)
(at package1 s21)
(at package2 s14)
(at package3 s17)
(at package4 s19)
(at package5 s13)
(at package6 s4)
```

```
(at package7 s2)
(at package8 s2)
(at package9 s16)
(at package10 s0)
(at package11 s5)
(at package12 s11)
(at package13 s11)
(at package14 s6)
(at package15 s17)
(at package16 s15)
(at package17 s18)
(at package18 s13)
(at package19 s20)
(path s0 p0-1)
(path p0-1 s0)
(path s1 p0-1)
(path p0-1 s1)
(path s1 p1-2)
(path p1-2 s1)
(path s2 p1-2)
(path p1-2 s2)
(path s1 p1-10)
(path p1-10 s1)
(path s10 p1-10)
(path p1-10 s10)
(path s1 p1-15)
(path p1-15 s1)
(path s15 p1-15)
(path p1-15 s15)
(path s2 p2-0)
(path p2-0 s2)
(path s0 p2-0)
(path p2-0 s0)
(path s2 p2-3)
(path p2-3 s2)
(path s3 p2-3)
(path p2-3 s3)
(path s2 p2-10)
```

```
(path p2-10 s2)
(path s10 p2-10)
(path p2-10 s10)
(path s2 p2-14)
(path p2-14 s2)
(path s14 p2-14)
(path p2-14 s14)
(path s2 p2-19)
(path p2-19 s2)
(path s19 p2-19)
(path p2-19 s19)
(path s3 p3-7)
(path p3-7 s3)
(path s7 p3-7)
(path p3-7 s7)
(path s3 p3-13)
(path p3-13 s3)
(path s13 p3-13)
(path p3-13 s13)
(path s5 p5-10)
(path p5-10 s5)
(path s10 p5-10)
(path p5-10 s10)
(path s5 p5-11)
(path p5-11 s5)
(path s11 p5-11)
(path p5-11 s11)
(path s6 p6-5)
(path p6-5 s6)
(path s5 p6-5)
(path p6-5 s5)
(path s7 p7-8)
(path p7-8 s7)
(path s8 p7-8)
(path p7-8 s8)
(path s8 p8-2)
(path p8-2 s8)
(path s2 p8-2)
```

```
(path p8-2 s2)
(path s8 p8-6)
(path p8-6 s8)
(path s6 p8-6)
(path p8-6 s6)
(path s8 p8-12)
(path p8-12 s8)
(path s12 p8-12)
(path p8-12 s12)
(path s8 p8-18)
(path p8-18 s8)
(path s18 p8-18)
(path p8-18 s18)
(path s8 p8-21)
(path p8-21 s8)
(path s21 p8-21)
(path p8-21 s21)
(path s9 p9-1)
(path p9-1 s9)
(path s1 p9-1)
(path p9-1 s1)
(path s10 p10-6)
(path p10-6 s10)
(path s6 p10-6)
(path p10-6 s6)
(path s10 p10-9)
(path p10-9 s10)
(path s9 p10-9)
(path p10-9 s9)
(path s10 p10-11)
(path p10-11 s10)
(path s11 p10-11)
(path p10-11 s11)
(path s11 p11-15)
(path p11-15 s11)
(path s15 p11-15)
(path p11-15 s15)
(path s12 p12-13)
```

```
(path p12-13 s12)
(path s13 p12-13)
(path p12-13 s13)
(path s12 p12-16)
(path p12-16 s12)
(path s16 p12-16)
(path p12-16 s16)
(path s12 p12-17)
(path p12-17 s12)
(path s17 p12-17)
(path p12-17 s17)
(path s13 p13-9)
(path p13-9 s13)
(path s9 p13-9)
(path p13-9 s9)
(path s14 p14-0)
(path p14-0 s14)
(path s0 p14-0)
(path p14-0 s0)
(path s14 p14-4)
(path p14-4 s14)
(path s4 p14-4)
(path p14-4 s4)
(path s16 p16-0)
(path p16-0 s16)
(path s0 p16-0)
(path p16-0 s0)
(path s16 p16-5)
(path p16-5 s16)
(path s5 p16-5)
(path p16-5 s5)
(path s16 p16-15)
(path p16-15 s16)
(path s15 p16-15)
(path p16-15 s15)
(path s17 p17-0)
(path p17-0 s17)
(path s0 p17-0)
```

```
(path p17-0 s0)
(path s17 p17-5)
(path p17-5 s17)
(path s5 p17-5)
(path p17-5 s5)
(path s17 p17-9)
(path p17-9 s17)
(path s9 p17-9)
(path p17-9 s9)
(path s17 p17-20)
(path p17-20 s17)
(path s20 p17-20)
(path p17-20 s20)
(path s19 p19-10)
(path p19-10 s19)
(path s10 p19-10)
(path p19-10 s10)
(path s19 p19-13)
(path p19-13 s19)
(path s13 p19-13)
(path p19-13 s13)
(path s20 p20-0)
(path p20-0 s20)
(path s0 p20-0)
(path p20-0 s0)
(path s20 p20-1)
(path p20-1 s20)
(path s1 p20-1)
(path p20-1 s1)
(path s20 p20-9)
(path p20-9 s20)
(path s9 p20-9)
(path p20-9 s9)
(path s20 p20-15)
(path p20-15 s20)
(path s15 p20-15)
(path p20-15 s15)
(path s21 p21-9)
```

```
(path p21-9 s21)
(path s9 p21-9)
(path p21-9 s9)
(link s0 s1)
(link s1 s0)
(link s0 s8)
(link s8 s0)
(link s0 s16)
(link s16 s0)
(link s1 s11)
(link s11 s1)
(link s1 s16)
(link s16 s1)
(link s2 s4)
(link s4 s2)
(link s2 s5)
(link s5 s2)
(link s2 s6)
(link s6 s2)
(link s2 s14)
(link s14 s2)
(link s2 s19)
(link s19 s2)
(link s2 s20)
(link s20 s2)
(link s3 s6)
(link s6 s3)
(link s3 s10)
(link s10 s3)
(link s3 s11)
(link s11 s3)
(link s3 s14)
(link s14 s3)
(link s3 s16)
(link s16 s3)
(link s3 s19)
(link s19 s3)
(link s3 s21)
```

```
(link s21 s3)
(link s4 s0)
(link s0 s4)
(link s4 s8)
(link s8 s4)
(link s4 s11)
(link s11 s4)
(link s4 s13)
(link s13 s4)
(link s4 s16)
(link s16 s4)
(link s5 s0)
(link s0 s5)
(link s5 s4)
(link s4 s5)
(link s5 s13)
(link s13 s5)
(link s5 s16)
(link s16 s5)
(link s6 s15)
(link s15 s6)
(link s6 s17)
(link s17 s6)
(link s6 s20)
(link s20 s6)
(link s7 s0)
(link s0 s7)
(link s7 s19)
(link s19 s7)
(link s8 s1)
(link s1 s8)
(link s8 s3)
(link s3 s8)
(link s8 s18)
(link s18 s8)
(link s8 s20)
(link s20 s8)
(link s9 s1)
```

```
(link s1 s9)
(link s9 s12)
(link s12 s9)
(link s9 s17)
(link s17 s9)
(link s9 s19)
(link s19 s9)
(link s10 s0)
(link s0 s10)
(link s10 s17)
(link s17 s10)
(link s10 s19)
(link s19 s10)
(link s10 s21)
(link s21 s10)
(link s11 s8)
(link s8 s11)
(link s11 s9)
(link s9 s11)
(link s11 s15)
(link s15 s11)
(link s12 s1)
(link s1 s12)
(link s12 s2)
(link s2 s12)
(link s12 s3)
(link s3 s12)
(link s12 s20)
(link s20 s12)
(link s12 s21)
(link s21 s12)
(link s13 s14)
(link s14 s13)
(link s14 s0)
(link s0 s14)
(link s14 s7)
(link s7 s14)
(link s14 s16)
```

```
(link s16 s14)
(link s14 s18)
(link s18 s14)
(link s17 s14)
(link s14 s17)
(link s17 s16)
(link s16 s17)
(link s17 s21)
(link s21 s17)
(link s18 s3)
(link s3 s18)
(link s19 s11)
(link s11 s19)
(link s20 s3)
(link s3 s20)
(link s20 s15)
(link s15 s20)
(link s20 s19)
(link s19 s20)
(link s21 s14)
(link s14 s21)
)
(:goal (and
(at driver2 s8)
(at driver3 s0)
(at driver4 s14)
(at driver6 s21)
(at driver7 s1)
(at truck2 s16)
(at truck3 s8)
(at truck4 s4)
(at truck5 s14)
(at truck6 s11)
(at package1 s11)
(at package2 s10)
(at package3 s19)
(at package4 s3)
(at package5 s12)
(at package6 s5)
(at package7 s6)
(at package8 s19)
(at package9 s21)
(at package10 s5)
(at package11 s4)
(at package12 s19)
(at package13 s15)
(at package14 s12)
(at package15 s4)
(at package16 s8)
(at package17 s11)
(at package18 s13)
(at package19 s8)
))

(:metric minimize (total-time))

)
```

A.1.2 RTAM

A.1.2.1 RTAM Domain

```
(define (domain rtam)
(:requirements :typing :durative-actions)
(:types
ambulance police_car tow_truck fire_brigade - vehicle
acc_victim vehicle car - subject
city_location city - location
accident_location hospital police_station - city_location
garage fire_station - city_location
route
accident)

(:predicates
(at ?physical_obj1 - subject ?location1 - location)
(available ?vehicle1 - vehicle)
(busy ?vehicle1 - vehicle)
(waiting ?subject1 - subject)
(certified ?subject1 - subject)
(aided ?subject1 - acc_victim)
(uncertified ?subject1 - subject)
(delivered ?subject1 - subject)
(loaded ?subject1 - subject ?vehicle1 - vehicle)
(identified ?accident1 - accident)
(vehicle_involve ?vehicle1 - vehicle)
(connects ?route1 - route ?location1 - location ?location2 - location)
(in_city ?location1 - location ?city1 - city)
(route_available ?route1 - route)
(trapped ?hum - acc_victim)
(untrapped ?hum - acc_victim)
(on_fire ?car_acc - car)
(off_fire ?car_acc - car)
)

(:functions
; (distance ?O - location ?L - location)
(route-length ?O - route)
; (confirmation-time)
; (firstaid-time)
(speed ?V - vehicle)
; (loading-time)
; (loading-time-car)
; (unloading-time)
; (delivery-time)
; (untrapping-time)
; (extinguishing-time)
)

(:durative-action confirm_accident
:parameters (?V - police_car ?P - subject ?A - accident_location)
:duration (= ?duration 10)
:condition (and
(at start (at ?V ?A))
(at start (at ?P ?A))
(at start (uncertified ?P))
)
:effect (and
(at start (not (uncertified ?P)))
(at end (waiting ?P))
(at end (certified ?P))
)
)

(:durative-action untrap
:parameters (?V - fire_brigade ?P - acc_victim ?A - accident_location)
:duration (= ?duration 25)
:condition (and
(at start (at ?P ?A))
(at start (at ?V ?A))
(at start (certified ?P))
(at start (available ?V))
(at start (waiting ?P))
(at start (trapped ?P))
)
:effect (and
(at start (not (available ?V)))
(at end (not (trapped ?P)))
(at end (untrapped ?P))
(at end (available ?V))
)
)

(:durative-action extinguish_fire
:parameters (?V - fire_brigade ?P - car ?A - accident_location)
:duration (= ?duration 20)
:condition (and
(at start (at ?P ?A))
(at start (at ?V ?A))
(at start (available ?V))
(at start (certified ?P))
```

```
(at start (waiting ?P))
(at start (on_fire ?P))
)
:effect (and
(at start (not (available ?V)))
(at end (not (on_fire ?P)))
(at end (off_fire ?P))
(at end (available ?V))
)
)

(:durative-action first_aid
:parameters (?V - ambulance ?P - acc_victim ?A - accident_location )
:duration (= ?duration 20)
:condition (and
(at start (at ?P ?A))
(at start (at ?V ?A))
(at start (certified ?P))
(at start (waiting ?P))
(at start (untrapped ?P))
)
:effect (and
(at start (not (waiting ?P)))
(at end (waiting ?P))
(at end (aided ?P))
)
)

(:durative-action load_victim
:parameters ( ?V - ambulance ?L - accident_location ?P - acc_victim)
:duration (= ?duration 5)
:condition (and
(at start (at ?V ?L))
(at start (at ?P ?L))
(at start (certified ?P))
(at start (waiting ?P))
(at start (aided ?P))
(at start (available ?V))
```

```
)
:effect (and
(at start (not (available ?V)))
(at start (busy ?V))
(at start (not (waiting ?P)))
(at start (not (at ?P ?L)))
(at end (loaded ?P ?V))
)
)

(:durative-action move
:parameters ( ?V - vehicle ?O - location ?City - city ?L - location
?City1 - city ?R - route)
:duration (= ?duration (/ (route-length ?R) (speed ?V)))
:condition (and
(at start (at ?V ?O))
(at start (in_city ?O ?City))
(at start (in_city ?L ?City1))
(at start (connects ?R ?City ?City1))
)
:effect (and
(at start (not (at ?V ?O)))
(at end (at ?V ?L))
)
)

; (:durative-action move_in_city
; :parameters ( ?V - vehicle ?O - location ?City - city ?L - location)
; :duration(=?duration(/(distance ?O ?L) (speed ?V)))
; :condition (and
; (at start (at ?V ?O))
; (at start (in_city ?O ?City))
; (at start (in_city ?L ?City))
; )
; :effect (and
; (at start (not (at ?V ?O)))
; (at end (at ?V ?L))
; )
; )

(:durative-action load_car
:parameters ( ?V - tow_truck ?L - accident_location ?P - car)
:duration (= ?duration 5)
:condition (and
(at start (at ?V ?L))
(at start (at ?P ?L))
(at start (waiting ?P))
(at start (certified ?P))
(at start (available ?V))
(at start (off_fire ?P))
)
:effect (and
(at start (not (available ?V)))
(at start (busy ?V))
(at start (not (waiting ?P)))
(at start (not (at ?P ?L)))
(at end (loaded ?P ?V))
)
)

(:durative-action unload_car
:parameters ( ?P - car ?L - garage ?V - tow_truck )
:duration (= ?duration 5)
:condition (and
(at start (at ?V ?L))
(at start (loaded ?P ?V))
(at start (busy ?V))
)
:effect (and
(at start (not (loaded ?P ?V)))
(at end (at ?P ?L))
(at start (not (busy ?V)))
(at end (waiting ?P))
(at end (available ?V))
)
)
```

```
(:durative-action unload_victim
:parameters ( ?P - acc_victim ?L - hospital ?V - ambulance)
:duration (= ?duration 5)
:condition (and
(at start (at ?V ?L))
(at start (loaded ?P ?V))
(at start (certified ?P))
(at start (aided ?P))
(at start (busy ?V))
)
:effect (and
(at start (not (loaded ?P ?V)))
(at end (at ?P ?L))
(at end (not (busy ?V)))
(at end (waiting ?P))
(at end (available ?V))
)
)

(:durative-action deliver_victim
:parameters ( ?P - acc_victim ?L - hospital )
:duration (= ?duration 10)
:condition (and
(at start (at ?P ?L))
(at start (waiting ?P))
(at start (certified ?P))
(at start (aided ?P))
)
:effect (and
(at start (not (waiting ?P)))
(at start (not (certified ?P)))
(at start (not (aided ?P)))
(at end (delivered ?P))
)
)

(:durative-action deliver_vehicle
:parameters ( ?P - car ?L - garage )
:duration (= ?duration 10)
:condition (and
(at start (at ?P ?L))
(at start (waiting ?P))
(at start (certified ?P))
)
:effect (and
(at start (not (waiting ?P)))
(at start (not (certified ?P)))
(at end (delivered ?P))
)
)

)
```

A.1.2.2 RTAM_5_1_35 Problem

```
(define (problem RTAM_5_1_35) (:domain RTAM)
(:objects
accident0 - accident
accident1 - accident
accident2 - accident
accident3 - accident
accident4 - accident
accident_location0 - accident_location
accident_location1 - accident_location
accident_location2 - accident_location
accident_location3 - accident_location
accident_location4 - accident_location
acc_victim0 - acc_victim
acc_victim1 - acc_victim
acc_victim2 - acc_victim
acc_victim3 - acc_victim
acc_victim4 - acc_victim
```

```
acc_victim5 - acc_victim
acc_victim6 - acc_victim
acc_victim7 - acc_victim
acc_victim8 - acc_victim
acc_victim9 - acc_victim
acc_victim10 - acc_victim
acc_victim11 - acc_victim
acc_victim12 - acc_victim
acc_victim13 - acc_victim
acc_victim14 - acc_victim
acc_victim15 - acc_victim
acc_victim16 - acc_victim
acc_victim17 - acc_victim
acc_victim18 - acc_victim
acc_victim19 - acc_victim
acc_victim20 - acc_victim
acc_victim21 - acc_victim
acc_victim22 - acc_victim
acc_victim23 - acc_victim
acc_victim24 - acc_victim
acc_victim25 - acc_victim
acc_victim26 - acc_victim
acc_victim27 - acc_victim
acc_victim28 - acc_victim
acc_victim29 - acc_victim
acc_victim30 - acc_victim
acc_victim31 - acc_victim
acc_victim32 - acc_victim
acc_victim33 - acc_victim
acc_victim34 - acc_victim
ambulance0 - ambulance
fire_brigade0 - fire_brigade
police_car0 - police_car
tow_truck0 - tow_truck
tow_truck1 - tow_truck
tow_truck2 - tow_truck
car0 - car
car1 - car
```

```
car2 - car
car3 - car
car4 - car
car5 - car
car6 - car
car7 - car
car8 - car
car9 - car
car10 - car
car11 - car
car12 - car
car13 - car
car14 - car
car15 - car
car16 - car
car17 - car
car18 - car
car19 - car
car20 - car
car21 - car
car22 - car
car23 - car
car24 - car
car25 - car
car26 - car
car27 - car
car28 - car
car29 - car
car30 - car
car31 - car
car32 - car
fire_Hud - fire_station
fire_Hal - fire_station
police_Queen - police_station
police_Bradley - police_station
police_Halifax - police_station
police_Huddersfield - police_station
huddersfield_hospital - hospital
```

```
halifax_hospital - hospital
brighouse_hospital - hospital
garage_halifax - garage
garage_huddersfield - garage
garage_brighouse - garage
garage_queensbury - garage
ainley_top - city
huddersfield - city
halifax - city
bradley - city
greetland - city
brighouse - city
baliff_bridge - city
queensbury - city
hud_bradley - route
bradley_ainley - route
hud_brigh - route
a629 - route
ainley_greet - route
ainley_brigh - route
greet_halifax - route
brigh_baliff - route
brigh_queen - route
baliff_halifax - route
queen_halifax - route
ainley_halifax - route
)
(:init
(= (speed ambulance0) 1)
(= (speed police_car0) 1.2)
(= (speed fire_brigade0) 0.8)
(= (speed tow_truck0) 0.8)
(= (speed tow_truck1) 0.8)
(= (speed tow_truck2) 0.8)
(in_city accident_location0 queensbury)
(in_city accident_location1 ainley_top)
(in_city accident_location2 bradley)
(in_city accident_location3 bradley)
```

```
(in_city accident_location4 halifax)
(in_city huddersfield_hospital huddersfield)
(in_city garage_huddersfield huddersfield)
(in_city police_Huddersfield huddersfield)
(in_city fire_Hud huddersfield)
(in_city halifax_hospital halifax)
(in_city garage_halifax halifax)
(in_city police_Halifax halifax)
(in_city fire_Hal halifax)
(in_city police_Queen queensbury)
(in_city garage_queensbury queensbury)
(in_city police_Bradley bradley)
(in_city garage_brighouse brighouse)
(in_city brighouse_hospital brighouse)
(route_available ainley_halifax)
(connects ainley_halifax halifax ainley_top)
(connects ainley_halifax ainley_top halifax)
(route_available hud_bradley)
(connects hud_bradley huddersfield bradley)
(connects hud_bradley bradley huddersfield)
(route_available bradley_ainley)
(connects bradley_ainley bradley ainley_top)
(connects bradley_ainley ainley_top bradley)
(route_available hud_brigh)
(connects hud_brigh huddersfield brighouse)
(connects hud_brigh brighouse huddersfield)
(route_available a629)
(connects a629 huddersfield ainley_top)
(connects a629 ainley_top huddersfield)
(route_available ainley_greet)
(connects ainley_greet ainley_top greetland)
(connects ainley_greet greetland ainley_top)
(route_available ainley_brigh)
(connects ainley_brigh ainley_top brighouse)
(connects ainley_brigh brighouse ainley_top)
(route_available greet_halifax)
(connects greet_halifax greetland halifax)
(connects greet_halifax halifax greetland)
```

```
(route_available brigh_baliff)
(connects brigh_baliff brighouse baliff_bridge)
(connects brigh_baliff baliff_bridge brighouse)
(route_available brigh_queen)
(connects brigh_queen brighouse queensbury)
(connects brigh_queen queensbury brighouse)
(route_available baliff_halifax)
(connects baliff_halifax baliff_bridge halifax)
(connects baliff_halifax halifax baliff_bridge)
(route_available queen_halifax)
(connects queen_halifax queensbury halifax)
(connects queen_halifax halifax queensbury)
(= (route-length hud_bradley) 10)
(= (route-length bradley_ainley) 10)
(= (route-length hud_brigh) 6)
(= (route-length a629) 5)
(= (route-length ainley_greet) 10)
(= (route-length ainley_brigh) 10)
(= (route-length greet_halifax) 2)
(= (route-length brigh_baliff) 8)
(= (route-length brigh_queen) 8)
(= (route-length baliff_halifax) 10)
(= (route-length queen_halifax) 2)
(= (route-length ainley_halifax) 4)
(at acc_victim0 accident_location2)
(uncertified acc_victim0)
(trapped acc_victim0)
(at acc_victim1 accident_location1)
(uncertified acc_victim1)
(trapped acc_victim1)
(at acc_victim2 accident_location0)
(uncertified acc_victim2)
(untrapped acc_victim2)
(at acc_victim3 accident_location0)
(uncertified acc_victim3)
(trapped acc_victim3)
(at acc_victim4 accident_location0)
(uncertified acc_victim4)
```

```
(trapped acc_victim4)
(at acc_victim5 accident_location3)
(uncertified acc_victim5)
(untrapped acc_victim5)
(at acc_victim6 accident_location4)
(uncertified acc_victim6)
(trapped acc_victim6)
(at acc_victim7 accident_location1)
(uncertified acc_victim7)
(untrapped acc_victim7)
(at acc_victim8 accident_location3)
(uncertified acc_victim8)
(untrapped acc_victim8)
(at acc_victim9 accident_location4)
(uncertified acc_victim9)
(trapped acc_victim9)
(at acc_victim10 accident_location2)
(uncertified acc_victim10)
(trapped acc_victim10)
(at acc_victim11 accident_location0)
(uncertified acc_victim11)
(untrapped acc_victim11)
(at acc_victim12 accident_location4)
(uncertified acc_victim12)
(untrapped acc_victim12)
(at acc_victim13 accident_location1)
(uncertified acc_victim13)
(trapped acc_victim13)
(at acc_victim14 accident_location3)
(uncertified acc_victim14)
(untrapped acc_victim14)
(at acc_victim15 accident_location4)
(uncertified acc_victim15)
(untrapped acc_victim15)
(at acc_victim16 accident_location4)
(uncertified acc_victim16)
(untrapped acc_victim16)
(at acc_victim17 accident_location4)
```

```
(uncertified acc_victim17)
(untrapped acc_victim17)
(at acc_victim18 accident_location4)
(uncertified acc_victim18)
(untrapped acc_victim18)
(at acc_victim19 accident_location2)
(uncertified acc_victim19)
(trapped acc_victim19)
(at acc_victim20 accident_location2)
(uncertified acc_victim20)
(trapped acc_victim20)
(at acc_victim21 accident_location2)
(uncertified acc_victim21)
(untrapped acc_victim21)
(at acc_victim22 accident_location0)
(uncertified acc_victim22)
(untrapped acc_victim22)
(at acc_victim23 accident_location1)
(uncertified acc_victim23)
(untrapped acc_victim23)
(at acc_victim24 accident_location3)
(uncertified acc_victim24)
(trapped acc_victim24)
(at acc_victim25 accident_location1)
(uncertified acc_victim25)
(untrapped acc_victim25)
(at acc_victim26 accident_location4)
(uncertified acc_victim26)
(untrapped acc_victim26)
(at acc_victim27 accident_location1)
(uncertified acc_victim27)
(untrapped acc_victim27)
(at acc_victim28 accident_location0)
(uncertified acc_victim28)
(untrapped acc_victim28)
(at acc_victim29 accident_location4)
(uncertified acc_victim29)
(untrapped acc_victim29)
```

```
(at acc_victim30 accident_location2)
(uncertified acc_victim30)
(untrapped acc_victim30)
(at acc_victim31 accident_location4)
(uncertified acc_victim31)
(trapped acc_victim31)
(at acc_victim32 accident_location3)
(uncertified acc_victim32)
(untrapped acc_victim32)
(at acc_victim33 accident_location2)
(uncertified acc_victim33)
(untrapped acc_victim33)
(at acc_victim34 accident_location4)
(uncertified acc_victim34)
(untrapped acc_victim34)
(at car0 accident_location0)
(uncertified car0)
(on_fire car0)
(at car1 accident_location1)
(uncertified car1)
(on_fire car1)
(at car2 accident_location4)
(uncertified car2)
(off_fire car2)
(at car3 accident_location3)
(uncertified car3)
(off_fire car3)
(at car4 accident_location1)
(uncertified car4)
(off_fire car4)
(at car5 accident_location2)
(uncertified car5)
(off_fire car5)
(at car6 accident_location1)
(uncertified car6)
(off_fire car6)
(at car7 accident_location0)
(uncertified car7)
```

```
(on_fire car7)
(at car8 accident_location0)
(uncertified car8)
(on_fire car8)
(at car9 accident_location2)
(uncertified car9)
(off_fire car9)
(at car10 accident_location0)
(uncertified car10)
(off_fire car10)
(at car11 accident_location2)
(uncertified car11)
(on_fire car11)
(at car12 accident_location2)
(uncertified car12)
(off_fire car12)
(at car13 accident_location4)
(uncertified car13)
(off_fire car13)
(at car14 accident_location1)
(uncertified car14)
(on_fire car14)
(at car15 accident_location0)
(uncertified car15)
(on_fire car15)
(at car16 accident_location1)
(uncertified car16)
(off_fire car16)
(at car17 accident_location1)
(uncertified car17)
(on_fire car17)
(at car18 accident_location0)
(uncertified car18)
(off_fire car18)
(at car19 accident_location0)
(uncertified car19)
(on_fire car19)
(at car20 accident_location3)
```

```
(uncertified car20)
(off_fire car20)
(at car21 accident_location4)
(uncertified car21)
(on_fire car21)
(at car22 accident_location3)
(uncertified car22)
(on_fire car22)
(at car23 accident_location0)
(uncertified car23)
(on_fire car23)
(at car24 accident_location2)
(uncertified car24)
(on_fire car24)
(at car25 accident_location4)
(uncertified car25)
(on_fire car25)
(at car26 accident_location1)
(uncertified car26)
(on_fire car26)
(at car27 accident_location1)
(uncertified car27)
(on_fire car27)
(at car28 accident_location0)
(uncertified car28)
(on_fire car28)
(at car29 accident_location2)
(uncertified car29)
(on_fire car29)
(at car30 accident_location2)
(uncertified car30)
(off_fire car30)
(at car31 accident_location4)
(uncertified car31)
(off_fire car31)
(at car32 accident_location1)
(uncertified car32)
(on_fire car32)
```

```
(available ambulance0)
(available fire_brigade0)
(available police_car0)
(available tow_truck0)
(available tow_truck1)
(available tow_truck2)
(at ambulance0 huddersfield_hospital)
(at fire_brigade0 fire_Hud)
(at police_car0 police_Queen)
(at tow_truck0 garage_halifax)
(at tow_truck1 garage_huddersfield)
(at tow_truck2 garage_brighouse)
)
(:goal (and
(delivered acc_victim0)
(delivered acc_victim1)
(delivered acc_victim2)
(delivered acc_victim3)
(delivered acc_victim4)
(delivered acc_victim5)
(delivered acc_victim6)
(delivered acc_victim7)
(delivered acc_victim8)
(delivered acc_victim9)
(delivered acc_victim10)
(delivered acc_victim11)
(delivered acc_victim12)
(delivered acc_victim13)
(delivered acc_victim14)
(delivered acc_victim15)
(delivered acc_victim16)
(delivered acc_victim17)
(delivered acc_victim18)
(delivered acc_victim19)
(delivered acc_victim20)
(delivered acc_victim21)
(delivered acc_victim22)
(delivered acc_victim23)
```

```
(delivered acc_victim24)
(delivered acc_victim25)
(delivered acc_victim26)
(delivered acc_victim27)
(delivered acc_victim28)
(delivered acc_victim29)
(delivered acc_victim30)
(delivered acc_victim31)
(delivered acc_victim32)
(delivered acc_victim33)
(delivered acc_victim34)
(delivered car0)
(delivered car1)
(delivered car2)
(delivered car3)
(delivered car4)
(delivered car5)
(delivered car6)
(delivered car7)
(delivered car8)
(delivered car9)
(delivered car10)
(delivered car11)
(delivered car12)
(delivered car13)
(delivered car14)
(delivered car15)
(delivered car16)
(delivered car17)
(delivered car18)
(delivered car19)
(delivered car20)
(delivered car21)
(delivered car22)
(delivered car23)
(delivered car24)
(delivered car25)
(delivered car26)
```

```
(delivered car27)
(delivered car28)
(delivered car29)
(delivered car30)
(delivered car31)
(delivered car32)
(at ambulance0 huddersfield_hospital)
(at fire_brigade0 fire_Hud)
(at police_car0 police_Queen)
(at tow_truck0 garage_halifax)
(at tow_truck1 garage_huddersfield)
(at tow_truck2 garage_brighouse)
))
(:metric minimize (total-time)))
```

A.1.2.3 RTAM_5_2_35 Problem

```
(define (problem RTAM_5_2_35) (:domain RTAM)
(:objects
accident0 - accident
accident1 - accident
accident2 - accident
accident3 - accident
accident4 - accident
accident_location0 - accident_location
accident_location1 - accident_location
accident_location2 - accident_location
accident_location3 - accident_location
accident_location4 - accident_location
acc_victim0 - acc_victim
acc_victim1 - acc_victim
acc_victim2 - acc_victim
acc_victim3 - acc_victim
acc_victim4 - acc_victim
acc_victim5 - acc_victim
```

```
acc_victim6 - acc_victim
acc_victim7 - acc_victim
acc_victim8 - acc_victim
acc_victim9 - acc_victim
acc_victim10 - acc_victim
acc_victim11 - acc_victim
acc_victim12 - acc_victim
acc_victim13 - acc_victim
acc_victim14 - acc_victim
acc_victim15 - acc_victim
acc_victim16 - acc_victim
acc_victim17 - acc_victim
acc_victim18 - acc_victim
acc_victim19 - acc_victim
acc_victim20 - acc_victim
acc_victim21 - acc_victim
acc_victim22 - acc_victim
acc_victim23 - acc_victim
acc_victim24 - acc_victim
acc_victim25 - acc_victim
acc_victim26 - acc_victim
acc_victim27 - acc_victim
acc_victim28 - acc_victim
acc_victim29 - acc_victim
acc_victim30 - acc_victim
acc_victim31 - acc_victim
acc_victim32 - acc_victim
acc_victim33 - acc_victim
acc_victim34 - acc_victim
ambulance0 - ambulance
ambulance1 - ambulance
ambulance2 - ambulance
ambulance3 - ambulance
fire_brigade0 - fire_brigade
fire_brigade1 - fire_brigade
fire_brigade2 - fire_brigade
police_car0 - police_car
police_car1 - police_car
```

```
police_car2 - police_car
police_car3 - police_car
police_car4 - police_car
tow_truck0 - tow_truck
tow_truck1 - tow_truck
tow_truck2 - tow_truck
tow_truck3 - tow_truck
tow_truck4 - tow_truck
tow_truck5 - tow_truck
tow_truck6 - tow_truck
car0 - car
car1 - car
car2 - car
car3 - car
car4 - car
car5 - car
car6 - car
car7 - car
car8 - car
car9 - car
car10 - car
car11 - car
car12 - car
car13 - car
car14 - car
car15 - car
car16 - car
car17 - car
car18 - car
car19 - car
car20 - car
car21 - car
car22 - car
car23 - car
car24 - car
car25 - car
car26 - car
car27 - car
```

```
car28 - car
car29 - car
car30 - car
car31 - car
car32 - car
fire_Hud - fire_station
fire_Hal - fire_station
police_Queen - police_station
police_Bradley - police_station
police_Halifax - police_station
police_Huddersfield - police_station
huddersfield_hospital - hospital
halifax_hospital - hospital
brighouse_hospital - hospital
garage_halifax - garage
garage_huddersfield - garage
garage_brighouse - garage
garage_queensbury - garage
ainley_top - city
huddersfield - city
halifax - city
bradley - city
greetland - city
brighouse - city
baliff_bridge queensbury - city
hud_bradley - route
bradley_ainley - route
hud_brigh - route
a629 - route
ainley_greet - route
ainley_brigh - route
greet_halifax - route
brigh_baliff - route
brigh_queen - route
baliff_halifax - route
queen_halifax - route
ainley_halifax - route
)
```

```
(:init
(= (speed ambulance0) 1)
(= (speed ambulance1) 1)
(= (speed ambulance2) 1)
(= (speed ambulance3) 1)
(= (speed police_car0) 1.2)
(= (speed police_car1) 1.2)
(= (speed police_car2) 1.2)
(= (speed police_car3) 1.2)
(= (speed police_car4) 1.2)
(= (speed fire_brigade0) 0.8)
(= (speed fire_brigade1) 0.8)
(= (speed fire_brigade2) 0.8)
(= (speed tow_truck0) 0.8)
(= (speed tow_truck1) 0.8)
(= (speed tow_truck2) 0.8)
(= (speed tow_truck3) 0.8)
(= (speed tow_truck4) 0.8)
(= (speed tow_truck5) 0.8)
(= (speed tow_truck6) 0.8)
(in_city accident_location0 queensbury)
(in_city accident_location1 ainley_top)
(in_city accident_location2 bradley)
(in_city accident_location3 bradley)
(in_city accident_location4 halifax)
(in_city huddersfield_hospital huddersfield)
(in_city garage_huddersfield huddersfield)
(in_city police_Huddersfield huddersfield)
(in_city fire_Hud huddersfield)
(in_city halifax_hospital halifax)
(in_city garage_halifax halifax)
(in_city police_Halifax halifax)
(in_city fire_Hal halifax)
(in_city police_Queen queensbury)
(in_city garage_queensbury queensbury)
(in_city police_Bradley bradley)
(in_city garage_brighouse brighouse)
(in_city brighouse_hospital brighouse)
```

```
(route_available ainley_halifax)
(connects ainley_halifax halifax ainley_top)
(connects ainley_halifax ainley_top halifax)
(route_available hud_bradley)
(connects hud_bradley huddersfield bradley)
(connects hud_bradley bradley huddersfield)
(route_available bradley_ainley)
(connects bradley_ainley bradley ainley_top)
(connects bradley_ainley ainley_top bradley)
(route_available hud_brigh)
(connects hud_brigh huddersfield brighouse)
(connects hud_brigh brighouse huddersfield)
(route_available a629)
(connects a629 huddersfield ainley_top)
(connects a629 ainley_top huddersfield)
(route_available ainley_greet)
(connects ainley_greet ainley_top greetland)
(connects ainley_greet greetland ainley_top)
(route_available ainley_brigh)
(connects ainley_brigh ainley_top brighouse)
(connects ainley_brigh brighouse ainley_top)
(route_available greet_halifax)
(connects greet_halifax greetland halifax)
(connects greet_halifax halifax greetland)
(route_available brigh_baliff)
(connects brigh_baliff brighouse baliff_bridge)
(connects brigh_baliff baliff_bridge brighouse)
(route_available brigh_queen)
(connects brigh_queen brighouse queensbury)
(connects brigh_queen queensbury brighouse)
(route_available baliff_halifax)
(connects baliff_halifax baliff_bridge halifax)
(connects baliff_halifax halifax baliff_bridge)
(route_available queen_halifax)
(connects queen_halifax queensbury halifax)
(connects queen_halifax halifax queensbury)
(= (route-length hud_bradley) 10)
(= (route-length bradley_ainley) 10)
(= (route-length hud_brigh) 6)
(= (route-length a629) 5)
(= (route-length ainley_greet) 10)
(= (route-length ainley_brigh) 10)
(= (route-length greet_halifax) 2)
(= (route-length brigh_baliff) 8)
(= (route-length brigh_queen) 8)
(= (route-length baliff_halifax) 10)
(= (route-length queen_halifax) 2)
(= (route-length ainley_halifax) 4)
(at acc_victim0 accident_location2)
(uncertified acc_victim0)
(trapped acc_victim0)
(at acc_victim1 accident_location1)
(uncertified acc_victim1)
(trapped acc_victim1)
(at acc_victim2 accident_location0)
(uncertified acc_victim2)
(untrapped acc_victim2)
(at acc_victim3 accident_location0)
(uncertified acc_victim3)
(trapped acc_victim3)
(at acc_victim4 accident_location0)
(uncertified acc_victim4)
(trapped acc_victim4)
(at acc_victim5 accident_location3)
(uncertified acc_victim5)
(untrapped acc_victim5)
(at acc_victim6 accident_location4)
(uncertified acc_victim6)
(trapped acc_victim6)
(at acc_victim7 accident_location1)
(uncertified acc_victim7)
(untrapped acc_victim7)
(at acc_victim8 accident_location3)
(uncertified acc_victim8)
(untrapped acc_victim8)
(at acc_victim9 accident_location4)
```

```
(uncertified acc_victim9)
(trapped acc_victim9)
(at acc_victim10 accident_location2)
(uncertified acc_victim10)
(trapped acc_victim10)
(at acc_victim11 accident_location0)
(uncertified acc_victim11)
(untrapped acc_victim11)
(at acc_victim12 accident_location4)
(uncertified acc_victim12)
(untrapped acc_victim12)
(at acc_victim13 accident_location1)
(uncertified acc_victim13)
(trapped acc_victim13)
(at acc_victim14 accident_location3)
(uncertified acc_victim14)
(untrapped acc_victim14)
(at acc_victim15 accident_location4)
(uncertified acc_victim15)
(untrapped acc_victim15)
(at acc_victim16 accident_location4)
(uncertified acc_victim16)
(untrapped acc_victim16)
(at acc_victim17 accident_location4)
(uncertified acc_victim17)
(untrapped acc_victim17)
(at acc_victim18 accident_location4)
(uncertified acc_victim18)
(untrapped acc_victim18)
(at acc_victim19 accident_location2)
(uncertified acc_victim19)
(trapped acc_victim19)
(at acc_victim20 accident_location2)
(uncertified acc_victim20)
(trapped acc_victim20)
(at acc_victim21 accident_location2)
(uncertified acc_victim21)
(untrapped acc_victim21)
```

```
(at acc_victim22 accident_location0)
(uncertified acc_victim22)
(untrapped acc_victim22)
(at acc_victim23 accident_location1)
(uncertified acc_victim23)
(untrapped acc_victim23)
(at acc_victim24 accident_location3)
(uncertified acc_victim24)
(trapped acc_victim24)
(at acc_victim25 accident_location1)
(uncertified acc_victim25)
(untrapped acc_victim25)
(at acc_victim26 accident_location4)
(uncertified acc_victim26)
(untrapped acc_victim26)
(at acc_victim27 accident_location1)
(uncertified acc_victim27)
(untrapped acc_victim27)
(at acc_victim28 accident_location0)
(uncertified acc_victim28)
(untrapped acc_victim28)
(at acc_victim29 accident_location4)
(uncertified acc_victim29)
(untrapped acc_victim29)
(at acc_victim30 accident_location2)
(uncertified acc_victim30)
(untrapped acc_victim30)
(at acc_victim31 accident_location4)
(uncertified acc_victim31)
(trapped acc_victim31)
(at acc_victim32 accident_location3)
(uncertified acc_victim32)
(untrapped acc_victim32)
(at acc_victim33 accident_location2)
(uncertified acc_victim33)
(untrapped acc_victim33)
(at acc_victim34 accident_location4)
(uncertified acc_victim34)
```

```
(untrapped acc_victim34)
(at car0 accident_location0)
(uncertified car0)
(on_fire car0)
(at car1 accident_location1)
(uncertified car1)
(on_fire car1)
(at car2 accident_location4)
(uncertified car2)
(off_fire car2)
(at car3 accident_location3)
(uncertified car3)
(off_fire car3)
(at car4 accident_location1)
(uncertified car4)
(off_fire car4)
(at car5 accident_location2)
(uncertified car5)
(off_fire car5)
(at car6 accident_location1)
(uncertified car6)
(off_fire car6)
(at car7 accident_location0)
(uncertified car7)
(on_fire car7)
(at car8 accident_location0)
(uncertified car8)
(on_fire car8)
(at car9 accident_location2)
(uncertified car9)
(off_fire car9)
(at car10 accident_location0)
(uncertified car10)
(off_fire car10)
(at car11 accident_location2)
(uncertified car11)
(on_fire car11)
(at car12 accident_location2)
```

```
(uncertified car12)
(off_fire car12)
(at car13 accident_location4)
(uncertified car13)
(off_fire car13)
(at car14 accident_location1)
(uncertified car14)
(on_fire car14)
(at car15 accident_location0)
(uncertified car15)
(on_fire car15)
(at car16 accident_location1)
(uncertified car16)
(off_fire car16)
(at car17 accident_location1)
(uncertified car17)
(on_fire car17)
(at car18 accident_location0)
(uncertified car18)
(off_fire car18)
(at car19 accident_location0)
(uncertified car19)
(on_fire car19)
(at car20 accident_location3)
(uncertified car20)
(off_fire car20)
(at car21 accident_location4)
(uncertified car21)
(on_fire car21)
(at car22 accident_location3)
(uncertified car22)
(on_fire car22)
(at car23 accident_location0)
(uncertified car23)
(on_fire car23)
(at car24 accident_location2)
(uncertified car24)
(on_fire car24)
```

```
(at car25 accident_location4)
(uncertified car25)
(on_fire car25)
(at car26 accident_location1)
(uncertified car26)
(on_fire car26)
(at car27 accident_location1)
(uncertified car27)
(on_fire car27)
(at car28 accident_location0)
(uncertified car28)
(on_fire car28)
(at car29 accident_location2)
(uncertified car29)
(on_fire car29)
(at car30 accident_location2)
(uncertified car30)
(off_fire car30)
(at car31 accident_location4)
(uncertified car31)
(off_fire car31)
(at car32 accident_location1)
(uncertified car32)
(on_fire car32)
(available ambulance0)
(available ambulance1)
(available ambulance2)
(available ambulance3)
(available fire_brigade0)
(available fire_brigade1)
(available fire_brigade2)
(available police_car0)
(available police_car1)
(available police_car2)
(available police_car3)
(available police_car4)
(available tow_truck0)
(available tow_truck1)
```

```
(available tow_truck2)
(available tow_truck3)
(available tow_truck4)
(available tow_truck5)
(available tow_truck6)
(at ambulance0 huddersfield_hospital)
(at ambulance1 halifax_hospital)
(at ambulance2 brighouse_hospital)
(at ambulance3 huddersfield_hospital)
(at fire_brigade0 fire_Hud)
(at fire_brigade1 fire_Hal)
(at fire_brigade2 fire_Hud)
(at police_car0 police_Queen)
(at police_car1 police_Bradley)
(at police_car2 police_Halifax)
(at police_car3 police_Huddersfield)
(at police_car4 police_Queen)
(at tow_truck0 garage_halifax)
(at tow_truck1 garage_huddersfield)
(at tow_truck2 garage_brighouse)
(at tow_truck3 garage_queensbury)
(at tow_truck4 garage_halifax)
(at tow_truck5 garage_huddersfield)
(at tow_truck6 garage_brighouse)
)
(:goal (and
(delivered acc_victim0)
(delivered acc_victim1)
(delivered acc_victim2)
(delivered acc_victim3)
(delivered acc_victim4)
(delivered acc_victim5)
(delivered acc_victim6)
(delivered acc_victim7)
(delivered acc_victim8)
(delivered acc_victim9)
(delivered acc_victim10)
(delivered acc_victim11)
```

```
(delivered acc_victim12)
(delivered acc_victim13)
(delivered acc_victim14)
(delivered acc_victim15)
(delivered acc_victim16)
(delivered acc_victim17)
(delivered acc_victim18)
(delivered acc_victim19)
(delivered acc_victim20)
(delivered acc_victim21)
(delivered acc_victim22)
(delivered acc_victim23)
(delivered acc_victim24)
(delivered acc_victim25)
(delivered acc_victim26)
(delivered acc_victim27)
(delivered acc_victim28)
(delivered acc_victim29)
(delivered acc_victim30)
(delivered acc_victim31)
(delivered acc_victim32)
(delivered acc_victim33)
(delivered acc_victim34)
(delivered car0)
(delivered car1)
(delivered car2)
(delivered car3)
(delivered car4)
(delivered car5)
(delivered car6)
(delivered car7)
(delivered car8)
(delivered car9)
(delivered car10)
(delivered car11)
(delivered car12)
(delivered car13)
(delivered car14)
```

```
(delivered car15)
(delivered car16)
(delivered car17)
(delivered car18)
(delivered car19)
(delivered car20)
(delivered car21)
(delivered car22)
(delivered car23)
(delivered car24)
(delivered car25)
(delivered car26)
(delivered car27)
(delivered car28)
(delivered car29)
(delivered car30)
(delivered car31)
(delivered car32)
(at ambulance0 huddersfield_hospital)
(at ambulance1 halifax_hospital)
(at ambulance2 brighouse_hospital)
(at ambulance3 huddersfield_hospital)
(at fire_brigade0 fire_Hud)
(at fire_brigade1 fire_Hal)
(at fire_brigade2 fire_Hud)
(at police_car0 police_Queen)
(at police_car1 police_Bradley)
(at police_car2 police_Halifax)
(at police_car3 police_Huddersfield)
(at police_car4 police_Queen)
(at tow_truck0 garage_halifax)
(at tow_truck1 garage_huddersfield)
(at tow_truck2 garage_brighouse)
(at tow_truck3 garage_queensbury)
(at tow_truck4 garage_halifax)
(at tow_truck5 garage_huddersfield)
(at tow_truck6 garage_brighouse)))
(:metric minimize (total-time)))
```