FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Ondrej Škopek

Planning for Transportation Problems

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the bachelor thesis:	prof. RNDr. Roman Barták, Ph.D.
Study programme:	Computer Science
Study branch:	General Computer Science

Prague 2017

In Prague, May 15, 2017 Ondrej Škopek

This work is dedicated to my parents: thank you for supporting me on my every step, no matter how crazy my ideas were.

A big thank you also goes to my advisor, prof. RNDr. Roman Barták, Ph.D., for all the extensive advice and guidance.

Thank you to Professor Carlos Linares López for kindly providing access to the Transport dataset from the 2011 International Planning Competition and thank you to all organizers of International Planning Competitions for the hard work put into gathering and maintaining the vast resource of knowledge and data that made this work possible.

Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042), is greatly appreciated.

Title: Planning for Transportation Problems

Author: Ondrej Škopek

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: prof. RNDr. Roman Barták, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: Today, a vast amount of resources is spent globally on ineffective transportation planning. Using techniques of automated planning, we study simplified variants of logistic problems, where items are delivered to their destinations using a fleet of vehicles moving on an oriented, non-negatively weighted graph that represents a road network. We propose several planning systems for the effective solution of such problems. Experiments conducted on original planning competition data show that our planners are able to improve plan quality when compared to domain-independent planners from the competition. Last but not least, we developed TransportEditor, a visualizer and editor of transportation problems, for efficient problem analysis, planner construction, and plan introspection.

Keywords: planning, transport, logistics

Contents

Introduction

With the advent of self-driving cars and automated warehouses, transportation has become an even more essential topic than it was a few years ago. Cheaper air, bus, and rail travel has enabled people to travel more than they have ever before. Every day, thousands of companies need to deliver items from one place to another. Due to economic, ecological, and safety reasons, making the use of transportation resources more efficient is essential not just for saving money, but for the future of our planet. It is only natural to try to automate the planning processes for the various transportation problems. We usually want to calculate the shortest possible paths for our vehicles to take, make the calculated plans adhere to laws or regulations, and possibly even accommodate the drivers' wishes. Doing all of this by hand is a non-trivial task — today's computers are far more effective at solving these problems.

A major part of the solution process for these problems, *planning*, is usually defined as the "reasoning side of acting" (Ghallab et al., 2004, Section 1.1). This means that we want to come up with a *sequence of actions* that leads to a desired *goal*. In transportation, examples of planning problems include finding a sequence of requests for truck drivers to pick up and deliver packages in a given day, or orchestrating robots in a warehouse to wrap and transport ordered goods to the loading dock for shipping.

Automated planning has historically been focused on *domain-independent* planning — planning without the use of specific knowledge about the problem's domain. As Nau (2007) states, this is mostly due to the research field of planning wanting to establish itself generally — focusing on a set of domains would not be useful for that. They also believe that this bias against domain-dependent planning is not as useful anymore. We can now benefit from the attained theoretical results and an advancement in computing power, resulting in a wider range of practical problems that can now be solved using planning techniques.

In this thesis, we will study variants of a specific planning domain introduced in the 2008 International Planning Competition (IPC) called *Transport*. It serves as an abstract representation of a family of related transportation problems and an important benchmark for planning.

The Transport domain, in its basic form, consists of a road network with items located at specified locations. The items are to be delivered to their destinations using a fleet of vehicles. Our aim is to deliver all items with the least total cost, or in the shortest amount of time, where the cost and/or duration of individual actions is dependent on the domain variant.

A natural interpretation of the Transport domain is that it represents a set of trucks delivering packages. However, the exact same domain formulation could be used to model a ride-sharing service like Uber, where cars drive around a city, picking up and dropping off people along the way, or a means of modeling a rush-hour scenario in a public transportation system.

Using several variants of this domain, we will compare the performance of our custom-built planners to that of the planners taking part in the original competition and discuss various advantages or shortcomings of these approaches.

To aid in the construction of planners and analysis of generated plans, we will also develop a planning system called TransportEditor. It will consist of a problem visualizer and editor for the Transport domain. To give insight into plan deficiencies, it will be possible to trace plan actions and see how the planning state evolves as the plan is executed. For fast prototyping, it will be possible to generate plans by using built-in and external planners without leaving the system. Currently, only a handful of similar systems exist and to the best of our knowledge, none of them are specialized for problems that assume a general graph with transportation agents in its nodes.

We aim to show that domain-dependent planning has an important role to play in the future and there are many problems yet to be solved, despite the loss of generality when compared to domain-independent planning. To show this, we will design custom planners and try to come up with domain-specific heuristics and other features to aid our planners in solving Transport problems as well and as fast as possible.

First, we will formalize and define the specific form of our chosen planning domain and its variants. We will show how it compares to other similarly themed problems that are being solved. Afterward, we will describe the approaches we used to build planners for both a simple sequential transport domain and a more complicated temporal variant. Finally, we will run benchmarks, comparing the performance of our planners to current state-of-the-art domain-independent planners.

1. Formal background

In this chapter, we introduce the concept of planning to help us formalize the studied transportation problems.

1.1 Automated planning

As previously stated, *planning* is usually defined as the reasoning side of acting — an abstract deliberation process that chooses and organizes actions by anticipating their outcomes (Ghallab et al., 2004, Section 1.1). It seems only natural that we want to have computers do this strenuous activity for us. Automated planning is an attempt at just that — it is an area of Artificial Intelligence (AI) that studies the planning process computationally (Ghallab et al., 2004, Section 1.1).

Unfortunately, the specific situations in which we want to use automated planning are very diverse — from devising a sequence of actions to shut down a nuclear power plant, planning the movements of a robotic arm on an assembly line, or devising the complex pattern of motor activations for spacecraft positioning. Due to this, researchers are often interested in domain-independent planning, where the planner gets information about both the domain and the specific problem at run time and attempts to devise a plan using only the provided knowledge and the planner's previously built-in processes (Ghallab et al., 2004, Section 1.3).

On the contrary, domain-specific planning, where domain knowledge has been built into the planner, has obvious advantages when solving problems in that domain. However, it is almost useless on problems of other domains (Ghallab et al., 2004, Section 1.3).

1.2 Planning model

As a basis for the later-defined representation of planning, we first define a conceptual model similar to the restricted model in (Ghallab et al., 2004, Section 1.4, Section 1.5).

Definition 1 (State-transition system). *A (restricted) state-transition system is a 3-tuple* $\Sigma = (S, A, \gamma)$*, where:*

- $S = \{s_1, s_2, \ldots\}$ *is a finite and fully observable set of states;*
- $A = \{\textit{no-op}, a_1, a_2, \ldots\}$ *is a finite set of actions;*
- $\gamma : S \times A \to S \cup \{\emptyset\}$ *is a state-transition function, such that for all* $s \in S$ *the function* $\gamma(s, \textit{no-op}) = s$*; and*
- Σ *is static and offline, it only changes when an action is applied to it and does not change during planning.*

It is assumed that all actions have no duration.

For a state $s \in S$, the actions $A_s = \{a \in A \mid \gamma(s,a) \neq \emptyset\}$ are called *applicable* to the given state s. The no-op action is applicable to all states.

The state-transition function γ and the set of actions A together loosely correspond to what we will call a *planning domain.* Planning domains define an abstract representation of actions we work with and how they are related, but they do not state anything about specific states or actions.

Given a state-transition system Σ, planning aims to find a sequence of actions to apply to the initial state in order to achieve some objective. The objective can be defined in various ways — we might want the planner to devise a plan that does not enter chosen states, or contrary to that, visits each of a set of states, or one that just ends at a specified state. We will use the last option for formalizing the notion of a *planning problem.*

Definition 2 (Planning problem). *(Ghallab et al., 2004, Part I) A planning problem is a 5-tuple $\mathcal{P} = (S, A, \gamma, s_0, g)$, where:*

- *(S, A, γ) is a state-transition system;*
- *$s_0 \in S$ is an initial state; and*
- *$g \subseteq S$ is a set of goal states.*

Now that we have defined a *planning problem* we can specify what we mean by the planner generating a *sequence of actions* to achieve a goal — we will call this sequence a *plan.*

Definition 3 (Plan). *(Ghallab et al., 2004, Section 1.5) For a planning problem $\mathcal{P} = (S, A, \gamma, s_0, g)$, a plan is a finite sequence of actions $(a_1, a_2, \ldots, a_k)$, $k \in \mathbb{N}$ where $\forall i \in \{1, 2, \ldots, k\} : a_i \in A$ and $\forall i \in \{1, 2, \ldots, k\} : \gamma(s_{i-1}, a_i) = s_i \in S$, while $s_k \in g$.*

A basic *planning model*, i.e. the abstraction of a whole real-life scenario we want to plan for, consists of three components (Figure 1.1):

- A *state-transition system* Σ, that evolves by using its state-transition function on the actions it receives;
- A *controller*, that given an input state $s \in S$ and a generated plan, provides an action $a \in A$ as output to the state-transition system, and receives the new state as feedback; and
- A *planner*, that uses a description of the state-transition system Σ to synthesize a plan for the controller to execute in order to reach a goal state from the initial state.

1.3 Classical planning

Although the previously defined restricted state-transition system is a simplification of real-world domains, it is a useful one. This simplification has historically been studied as classical planning.

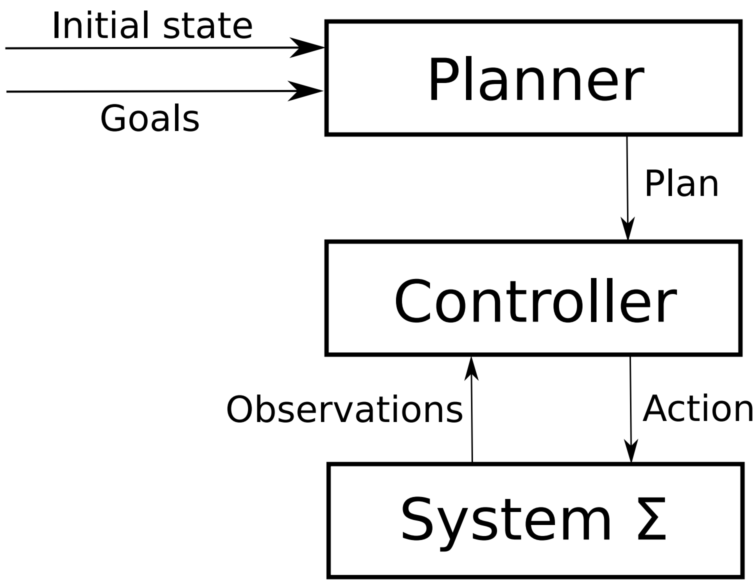


Figure 1.1: A typical planning model for offline planning — a state-transition system Σ, a controller executing a plan, and a planner devising the plan based on an initial state and goals. Adapted from (Ghallab et al., 2004, Figure 1.3).

A different branch of automated planning, *neoclassical planning*, uses largely the same theoretical foundations as classical planning. What is different is the approach to planning using those foundations — instead of search space nodes being a sequence of actions or a partially ordered set of actions, we view them as a set of several partial plans (Ghallab et al., 2004, Part II). One of the most famous results in neoclassical planning is the GraphPlan algorithm published by Blum and Furst (1997). It is out of the scope of this text to describe it in detail — see Ghallab et al. (2004, Section 6.3). GraphPlan makes heavy use of a data structure called a *planning graph*, which caused a breakthrough in the field of (domain-independent) planning, resulting in larger problems now being practically solvable.

We will now describe several theoretical domain-independent representations of planning problems used in classical planning (Ghallab et al., 2004, Chapter 2), so that we can formulate the Transport domain using them.

1.3.1 Set-theoretic representation

Leveraging propositional logic, both the planning domain and problem are represented with the notion of proposition symbols $L = \{p_1, p_2, \ldots\}$. Each state $s \in S = 2^L$ is defined as a subset of propositions of L — those propositions which hold in the given state. S is closed under the application of each action $a \in A$.

An action a is a triple of sets of propositions from L. We denote the triple $a = (\text{precond}(a), \text{effects}^-(a), \text{effects}^+(a))$, where:

- $\text{precond}(a)$ are the *preconditions* of an action: the set of propositions that must hold in the current state for the action to be applicable to it;
- $\text{effects}^-(a)$ are the *negative effects* of an action: the set of propositions that will no longer hold in the state once the action is applied; and

- similarly, $\text{effects}^+(a)$ are the *positive effects* of an action: the set of propositions that will be true in the state once the action is applied.

Note that an action cannot have the same proposition as a negative and positive effect at the same time — the sets $\text{effects}^+(a)$ and $\text{effects}^-(a)$ are disjoint for all actions a. The state-transition function is:

$$\gamma(s,a) = \begin{cases} (s \setminus \text{effects}^-(a)) \cup \text{effects}^+(a), & \text{if } a \text{ is applicable to } s, \\ \text{undefined}, & \text{else.} \end{cases}$$

Goal states S_g are defined as $S_g = \{s \in S \,|\, g \subseteq s\}$, where $g \subseteq L$ is any chosen set of propositions. The propositions g are called *goal propositions*.

1.3.2 Classical representation

The classical representation generalizes the set-theoretic representation using first-order logic, without functions. States are sets of ground atoms of a first-order language. Actions are ground instances of *planning operators*, triples denoted $o = (\text{name}(o), \text{precond}(o), \text{effects}(o))$:

- $\text{name}(o)$ is a syntactic expression of the given operator;
- $\text{precond}(o)$ and $\text{effects}(o)$ are sets of literals (atoms or their negations), similar in use to their equivalents in the set-theoretic case. $\text{precond}^+(o)$ and $\text{precond}^-(o)$ are the positive and negative preconditions of o. Likewise, $\text{effects}^+(o)$ and $\text{effects}^-(o)$ are the positive and negative effects of o.

Note that for a set of literals L, L^+ is the set of all atoms in L, and L^- is the set of all atoms whose negations are in L. The state-transition function is defined similarly to the set-theoretic representation, but using the updated definition of $\text{effects}(o)$. Goal states are defined as the set of states that satisfy g, the *goal*, where g is any set of ground literals.

The following is an example of a planning operator for driving a vehicle between two connected locations:

$$o_{\text{drive}} = (\text{drive}(v,f,t), \{\text{at}(v,f), \text{road}(f,t)\}, \{\text{at}(v,t), \neg\text{at}(v,f)\}).$$

The variable v denotes a vehicle, f and t denote the origin and destination locations, respectively. The at predicate is true if and only if the vehicle is located at that position in the given state and the road predicate is true if and only if a road exists between the two locations. An example of an action instantiated from the operator (sometimes referred to as an *operator instance*) for a vehicle v_1 and two locations l_1 and l_2 would be:

$$a_{\text{drive},\text{v}_1,\text{l}_1,\text{l}_2} = (\text{drive}(\text{v}_1,\text{l}_1,\text{l}_2), \{\text{at}(\text{v}_1,\text{l}_1), \text{road}(\text{l}_1,\text{l}_2)\}, \{\text{at}(\text{v}_1,\text{l}_2), \neg\text{at}(\text{v}_1,\text{l}_1)\}).$$

Both the set-theoretic and the classical representations follow the *Closed world assumption* — any atom/predicate not present in the state does not hold in that state.

1.3.3 State-variable representation

The state-variable representation substitutes the use of relations of the previous representation for functions, using the concept of state variables. State variables are functions that take the state as an input and serve as characteristic attributes, defining the state. We usually use a more practical way of defining these functions when planning — we assume the current state as an input without denoting it, and instead add different inputs.

For example, a useful set of state-variable functions for a domain that contains a road network and vehicles might be:

$$\mathrm{location}_v : S \to \mathrm{locations},$$

where $v \in$ vehicles. Instead, we could define a single function:

$$\mathrm{location}' : \mathrm{vehicles} \times S \to \mathrm{locations},$$

using $\mathrm{location}'(v, s) = \mathrm{location}_v(s)$, and afterwards:

$$\mathrm{location}'' : \mathrm{vehicles} \to \mathrm{locations},$$

using $\mathrm{location}''(v) = \mathrm{location}'(v, state_{cur})$, where $state_{cur}$ is the current state.

Planning operators are defined similarly to the classical representation, but $\mathrm{precond}(o)$ is now a set of expressions on state variables and relations. Also, $\mathrm{effects}(o)$ is defined as a set of assignments of values to state variables. For comparison, we show the same planning operator as in the classical representation:

$$o_{\mathrm{drive}} = (\mathrm{drive}(v, f, t), \{\mathrm{at}(v) = f, \mathrm{road}(f, t)\}, \{\mathrm{at}(v) \leftarrow t\})\,.$$

An example of an action instantiated from this operator for vehicle v_1 and two locations l_1 and l_2 is:

$$a_{\mathrm{drive},\mathrm{v}_1,\mathrm{l}_1,\mathrm{l}_2} = (\mathrm{drive}(\mathrm{v}_1, \mathrm{l}_1, \mathrm{l}_2), \{\mathrm{at}(\mathrm{v}_1) = \mathrm{l}_1, \mathrm{road}(\mathrm{l}_1, \mathrm{l}_2)\}, \{\mathrm{at}(\mathrm{v}_1) \leftarrow \mathrm{l}_2\})\,.$$

The state-transition function is defined analogously to the classical representation: an action a (ground instance of operator o) is applicable to a state s if the $\mathrm{precond}(o)$ condition is true given the values of state variables in state s. The resulting state is created by changing the state variables according to the assignments in $\mathrm{effects}(o)$ and the corresponding values of state variables in state s. The goal is defined as a set of ground state variables and their corresponding values (Ghallab et al., 2004, Section 2.5.2).

1.3.4 Extensions of representations

We will later extend the representations using types. To see how types fit into our previously defined representations, we can define a *type* as a unary predicate, which has the value true if and only if the predicate's argument is of the given type. We can then add these predicates as preconditions of actions. Adding types makes domain and problem formulations easier to read and gives additional information to planners, making them more efficient (Ghallab et al., 2004, Section 2.4.1). As an example of adding types, we show the previous planning operator for driving with the added vehicle type *veh* and location type *loc*:

$$o_{\mathrm{drive}} = (\mathrm{drive}(v, f, t), \{\mathrm{veh}(v), \mathrm{loc}(f), \mathrm{loc}(t), \mathrm{at}(v) = f, \mathrm{road}(f, t)\}, \{\mathrm{at}(v) \leftarrow t\})\,.$$

1.3.5 State-space planning and Plan-space planning

A different way of viewing the state-transition system $\Sigma = (S, A, \gamma)$ in a planning problem $\mathcal{P} = (S, A, \gamma, s_0, g)$ (Definition 1 and 2), is that of a labeled, directed graph $G(S, E, w)$, where:

- $E = \{(u, v) \in S^2 \mid \exists a \in A : \ \gamma(u, a) = v\}$; and
- $w : E \to 2^A$, such that $\forall (u, v) = e \in E : w(e) = \{a \in A \mid \gamma(u, a) = v\}$.

From the definition above, we see that applicable actions correspond to state transitions. During planning, the plan represented by the current position in the state space is the sequence of transitions from the start state s_0 to the current state (Ghallab et al., 2004, Section 4.1). *State-space planning* is a term used for planning techniques that use the state space for searching for a plan.

An alternative to using the state space is offered by *plan-space planning.* The state space is substituted for *plan space.* Nodes in this space represent *partially specified plans*, edges are *plan refinement operations* (Ghallab et al., 2004, Section 5.1). We will not explicitly use plan-space planning in this work, as according to Ghallab et al. (2004, Section 5.6) it is unfit for incorporating domain-specific knowledge.

1.3.6 Temporal planning

The addition of time makes modeling planning problems difficult and solving them even more difficult. However, it also makes most models more realistic and practically usable.

For an exhaustive introduction to temporal planning, see Ghallab et al. (2004, Chapter 13 and 14). We will only use the *durative action* modeling approach specified by Fox and Long (2003, Section 5). We adapt the following concepts presented in Ghallab et al. (2004, Section 14.2) that are relevant to modeling temporal transportation planning problems:

- A *temporally qualified expression* (tqe) is any expression in the form:

  $$p(x_1, x_2, \ldots, x_k)@[t_s, t_e),$$

  where p is a relation of the planning domain and $x_1, \ldots, x_k$ are constants or *object variables.* These are similar to state variables in the state-variable representation, but the values change in time, not between states.

  The tqe $p(x_1, \ldots, x_k)@[t_s, t_e)$ asserts that the relation $p(x_1, \ldots, x_k)$ holds for any time t, where $t_s \leq t < t_e$.

  The temporal variables t_s and t_e do not specifically represent a numerical time value, but together with other variables and constraints form a consistent set. It holds that $t_s < t_e$. For a more precise definition, see Ghallab et al. (2004, Section 14.2.1).

- A *temporal planning operator* is a tuple o, defined similarly to the planning operator in classical representations, but precond(o) and effects(o) are tqes.

1.4 PDDL

Originally proposed by McDermott et al. (1998) for the 1st International Planning Competition,[1] the Planning Domain Definition Language (PDDL) has become a de facto standard language for modeling planning domains and problems, continually evolving to the needs of the research community and the needs of the IPC itself throughout the years. We will use it as input for our planners.

PDDL was inspired by the language used to describe STRIPS (Fikes and Nilsson, 1971) and the numerous languages that sparked from it. It has a Lisp-like[2] declarative syntax and is very extensible. A basic PDDL domain and problem definition (without extensions) essentially correspond to the representations defined previously. Confusingly, we call PDDL planning operators *actions*. Each action has a list of *parameters* to be grounded by the planner, a *precondition*, and an *effect*. To denote multiple preconditions and effects, we use the n-ary predicate `and`. A full format specification applicable to our use case is available in Fox and Long (2003, Appendix A).

Not many planners support PDDL in its entirety — they usually support several "feature subsets", called *requirements*. One problem with the diversity of these requirements is that rarely does a single planner support more than a few, which makes comparing them on a diverse set of problems difficult.

An important version of PDDL, version 2.1, added support for temporal planning using *durative actions* (Fox and Long, 2003, Section 5), an analog of the previously defined temporal planning operators. Specifically, every durative action has a `duration`, specified either by a constant or a numeric fluent (a function with numerical values that can change over time). Also, instead of a precondition it introduces a *condition* (it is not necessary that the condition takes place before the action). We will only use so-called *discretized* durative actions, meaning that both the condition and the effect represent a temporally qualified expression, denoted using three unary PDDL predicates:

- `at start`, where the parameter predicate must hold or the parameter effect must be applied at the start of the action;
- `at end`, where it must hold or be applied at the end of the action; and
- `over all`, where the parameter must hold over the duration of the action, start and end non-inclusive. This temporal predicate is only applicable to preconditions of an action. When using *continuous* durative actions which have continuous effects (for example, generating heat and boiling water), effects are modeled using different syntactical constructs (Fox and Long, 2003, Section 5.3).

Over time, PDDL has evolved from the originally proposed version 1.2 to the now standard version 3.1. Several extensions and successors were proposed, like Multi-Agent PDDL (MA-PDDL) and Probabilistic PDDL (PPDDL).

PDDL does not specify a representation for plans. For a specific planning domain and problem in PDDL, we represent the plan in a format that is a field-wide consensus. We refer to it as the *VAL-like* format. VAL (Howey and Long,

[1] `http://ipc98.icaps-conference.org/`

[2] `https://en.wikipedia.org/wiki/Lisp_(programming_language)`

2003) is a plan validator created for the IPC. It takes as input (among several options) three filenames: filename of a planning domain and problem in PDDL, and a filename for a plan in the mentioned format. As described in Howey and Long (2003, Figure 2), the approximate format consists of multiple lines in the following format:

```
(action_name action_object_literals*)
```

For temporal domains, the format adds a start time and duration for each line:

```
start_time: (action_name action_object_literals*) [duration]
```

Both `start_time` and `duration` are floating-point numbers with a dot (`.`) as a decimal delimiter. Note that all text between a semicolon (`;`) and an end-of-line character sequence (`\r`, `\n`, or `\r\n`) is regarded as a comment and ignored by all PDDL parsers.

1.5 Planning in practice

In practice, many of the assumptions we made will get violated and many additional requirements will arise, due to various business or societal requirements. On the other hand, these assumptions allow us to work on problems that are more general and can, therefore, be applied to multiple scenarios. Businesses can often add minor tweaks on top of the obtained results so that their needs are satisfied. For example, online planning can often be foregone for some form of *windowed* planning, where we plan a certain time window offline and move on to the next window, repeating the process regularly.

Planners, in practice, are computer programs that are fed two text files as input — the domain file and the problem file. After that, they proceed with their internal calculations and upon finishing, return a plan (or not). We can then evaluate the plan, see if it meets our criteria, and, potentially, execute it in the real world.

What we are missing from a bare plan is the allocation of specific resources. *Scheduling* addresses the problem of how to perform a given set of actions (a plan) using a limited number of resources in a limited amount of time, and that is crucial to practical usage of any plan (Ghallab et al., 2004, Chapter 15).

In this text, we will only study the abstracted and simplified first part of this whole process — finding the “best” actions that lead to a specified goal.

2. Transport domain formulation

In this chapter, we will formulate and later formalize variants of the Transport domain. We will also mention a few related transportation problems that have been studied in the past.

2.1 Description of Transport domain variants

Transport is a planning domain designed for the International Planning Competition (IPC), which is part of the International Conference on Automated Planning and Scheduling (ICAPS). Originally, Transport appeared at IPC-6[1] which took place in 2008. Since then, it has been used in two IPCs, specifically IPC-7[2] in 2011 and IPC-8[3] in 2014.

There are a few basic formulations of the Transport domain family (i.e. similar Transport domain variants) which we will describe in the following sections.

2.1.1 Common traits of Transport domains

Transport is a logistics domain — vehicles drive around on a (generally asymmetric) positively-weighted oriented graph, picking up and dropping packages along the way. All vehicles have limited capacities (the sum of package sizes they can carry). Picking up or dropping a package costs 1 unit. The cost of driving along a road is equal to the edge weight (in other words, the road length). Road lengths are always positive integers. The general goal is to minimize the *total cost* while delivering all packages to their destination, where the total cost of a plan is defined as the sum of costs of all actions in the plan. A few Transport problems also request that the vehicles be positioned at certain locations in the graph after finishing their deliveries.

One variant which we will not study in this work is the NoMystery domain from IPC 2011, devised as a simplification of Transport (using only a single vehicle with no capacity constraints). The domain assumes fuel costs for driving on roads, with the vehicle having an initial fuel capacity (there is no refueling). All actions have a cost of 1. It is reasonably straightforward to solve problems of this variant using domain-specific knowledge as shown in Barták and Vodrážka (2016): the vehicle is always greedily loaded with all the packages present at a location when arriving at it and greedily unloaded when it contains a package which has the given location as a destination. Choosing which roads the vehicle drives along and thus determining the order of package loading and unloading while taking into account the fuel constraints is the task that is left for the planner to solve.

[1]`http://icaps-conference.org/ipc2008/deterministic/Domains.html`
[2]`http://www.plg.inf.uc3m.es/ipc2011-deterministic/`
[3]`https://helios.hud.ac.uk/scommv/IPC-14/`

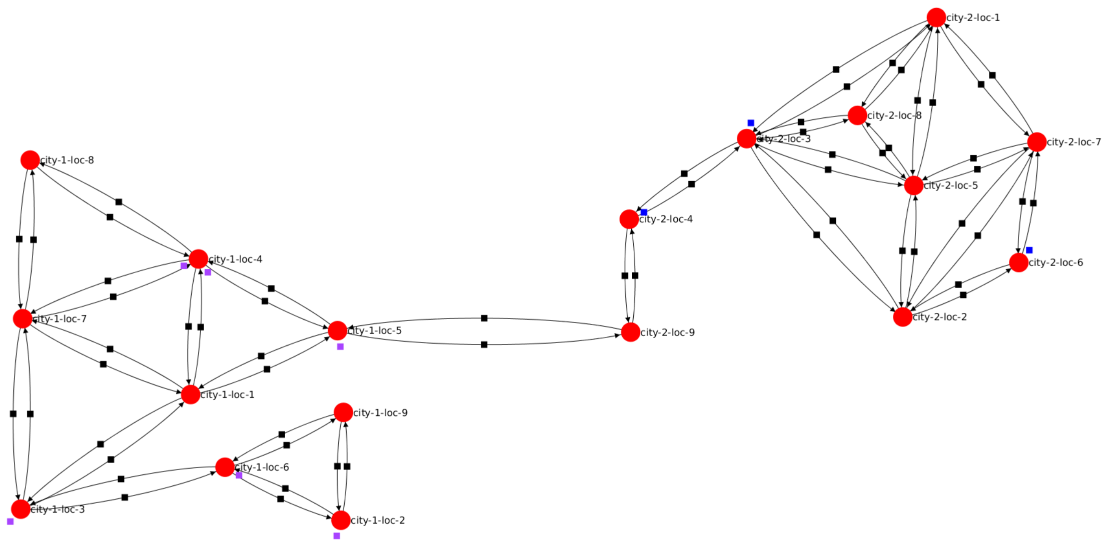


Figure 2.1: Road network visualization of the `p13` problem from the seq-sat track of IPC 2008. Red dots represent locations (graph nodes), roads (graph edges) are represented by black arrows, vehicles are plotted as blue squares, and packages as purple squares.

2.1.2 Transport STRIPS

STRIPS, the Stanford Research Institute Problem Solver, was a planner proposed by Fikes and Nilsson (1971). The influence of STRIPS was, however, not only due to the planner, but the language used to describe its inputs — the planning operators and goals. That is why we sometimes refer to classical planning (Section 1.3) as STRIPS planning. For the purposes of this text, we will use these terms interchangeably.

In the STRIPS variant of the Transport domain, all packages have a size of 1 and vehicles of a bounded capacity can drive around indefinitely (there is no notion of fuel or anything similar). The only reason for them not to, is that driving incurs a cost of its own, usually much larger than picking up or dropping off packages. This being a classical STRIPS domain, it does not assume time in any sense, so actions have no duration and are applied one after the other, sequentially.

This formulation contains three basic planning operators:

- `drive`, where a vehicle drives to an adjacent location along a road that is connected to its current location;
- `pick-up`, where a vehicle that is stationary at a location picks up a co-located package; and
- `drop`, where a stationary vehicle drops a package off at the vehicle's location.

In all the datasets, this domain variant is denoted as *Transport sequential* or *transport-strips* and we will alternate between these terms in this text. See Figure 2.1 for a visualization of an example problem for this domain.

2.1.3 Transport Numeric

The numeric variant adds the concept of fuel on top of the STRIPS variant. All roads have an additional cost, called `fuel-demand`, which is subtracted from a vehicle's `fuel-left` value if it chooses to drive along that road. Additionally, all vehicles have a maximum fuel capacity `fuel-max`, which they regain upon being the target of a `refuel` action. This action can only be executed at a location that is marked as having a petrol station. Petrol stations are static with respect to a given planning problem instance.

This variant is usually denoted as *Transport numeric* or *transport-numeric.*

2.1.4 Transport Temporal

The temporal Transport domain is usually denoted as *Transport temporal* or, confusingly, also *transport-numeric.* A major difference with respect to the numeric variant is the addition of time. All actions now have a duration (`pick-up` and `drop` both have a duration of 1, `refuel` has a duration of 10, and the duration of `drive` is equal to the length of the road we are driving along). Furthermore, packages now have various sizes (positive integers).

The addition of time poses numerous technical complications when formalizing this variant — its PDDL formulation significantly differs from the two previous ones, but only in technical details, not in objectives of the model. One important technicality is that a vehicle cannot pick up or drop packages concurrently — it always handles packages one at a time. Also, vehicles cannot do other actions while driving to another location (they are essentially placed "off the graph" for the duration of driving).

The overall goal remains largely the same (deliver packages to their destinations), but we no longer optimize the total cost. Instead, we now minimize the total duration of a plan, defined as the maximum time when an action is still taking place. In practice, this translates to minimizing maximum end time over all actions, which is often referred to as minimizing the *makespan.*

2.2 Formalizing the Transport domain

We will now translate the informal description of the Transport domain from the previous section to the formal representations we defined in Section 1.3. We will not formulate all the domain variants in all representations as they are very much alike and not needed for the comprehension of the following chapters.

2.2.1 Transport's classical representation

We are now able to show the sequential Transport domain in one of the representations previously defined, namely, the classical representation (Figure 2.2). For practical reasons, we will use a slight modification, obtained by adding a limited concept of functions with finite integer values. It is obvious that we could substitute these functions for the appropriate relations and a finite number of literals for the values for any given problem instance of the domain in this representation, so that it adheres to the definition of a classical formulation. For example, we

```
drive(v, l1, l2)
  ;; vehicle v moves from location l1 to an adjacent location l2
  precond: at(v, l1), road(l1, l2)
  effects: not at(v, l1), at(v, l2)

pick-up(v, l, p, s1, s2)
  ;; vehicle v picks up package p at location l,
  ;; decreasing its capacity from s2 to s1
  precond: at(v, l), at(p, l), capacity-predecessor(s1, s2),
           capacity(v, s2)
  effects: not at(p, l), in(p, v), capacity(v, s1),
           not capacity(v, s2)

drop(v, l, p, s1, s2)
  ;; vehicle v drops package p at location l,
  ;; increasing its capacity from s1 to s2
  precond: at(v, l), in(p, v), capacity-predecessor(s1, s2),
           capacity(v, s1)
  effects: not in(p, v), at(p, l), capacity(v, s2),
           not capacity(v, s1)
```

Figure 2.2: Classical formulation of `transport-strips`.

could add literals representing a finite set of natural numbers and a predicate that represents a successor relation, defined as $\texttt{successor}(a, b) \equiv a + 1 = b$.

Note that this representation does not contain the notion of a total cost of a plan that we will optimize for later. The predicates and functions used are:

- `at(o, l)`, the package or vehicle `o` is at the location `l`;
- `capacity(v, s)`, the vehicle `v` currently has `s` free space — `s` is a variable for space literals, a set of literals denoting the amount of space (essentially, these literals are a unary representation of a finite set of integers);
- `capacity-predecessor(s1, s2)`, the space literals represented by `s1` and `s2` satisfy the relation $\texttt{s1} + 1 = \texttt{s2}$ in the unary representation;
- `in(p, v)`, the package `p` is in the vehicle `v`;
- `road(l1, l2)`, the location `l1` is directly adjacent to the location `l2` by a road; and
- `road-length(l1, l2)`, the driving distance between location `l1` and `l2`, modeled as a numerical function. Does not change while planning.

The numeric variant adds the `refuel` operator, changes the `drive` operator, and adds a new fuel-related predicate `has-petrol-station(l)`, that is true when the given location `l` has a petrol station. To model fuel, we need the addition of a few functions, namely:

- `fuel-demand(l1, l2)`, the amount of fuel needed to drive from location `l1` to location `l2`;

```
drive(v, l1, l2)
  ;; vehicle v moves from location l1 to an adjacent location l2
  precond: at(v, l1), road(l1, l2), fuel-left(v) >= fuel-demand(l1, l2)
  effects: not at(v, l1), at(v, l2),
           decrease(fuel-left(v),  fuel-demand(l1, l2))

refuel(v, l)
  ;; vehicle v is refueled to the maximum at location l
  precond: at(v, l), has-petrol-station(l)
  effects: assign(fuel-left(v), fuel-max(v))
```

Figure 2.3: Classical formulation of `transport-numeric`'s differences compared to `transport-strips`.

- `fuel-left(v)`, the amount of fuel left in the vehicle `v`; and
- `fuel-max(v)`, the maximum amount of fuel the vehicle `v` can contain, i.e. its fuel tank capacity.

See Figure 2.3 for the exact differences in the representation after adding fuel.

We also slightly abuse the notation with `decrease` and `assign`; the left parameter's value is to be decreased by the right parameter's value or the left parameter's value is to be overridden by the right parameter's value, respectively.

2.2.2 Transport's state-variable representation

We are now also able to show the sequential Transport domain in the state-variable representation (Figure 2.4). Some predicates (`at`, `capacity`, and `in`) have been transformed into state-variable functions with largely the same semantics as in Section 2.2.1. Again, we leave out the total cost notion.

As before, the numeric variant adds the `refuel` operator along with a few fuel-related state-variable functions and predicates, and changes the `drive` operator (Figure 2.5).

We will represent the temporal variant of Transport using a variant of the state-variable representation using temporal planning operators, further referred to as the *temporal state-variable representation.* On top of fuel-related predicates and functions from numeric Transport, temporal Transport adds:

- `package-size(p)`, a function with positive integer values representing the size of the package `p` (does not change during planning); and
- `ready-loading(v)`, a predicate used for "locking" the vehicle `v` during `pick-up` and `drop` actions (enforcing the property of pairs of these two actions happening sequentially in time for a given vehicle). It is important to note that the `refuel` action does not lock the vehicle, which means that vehicles can be refueled while dropping off and picking up packages.

Figure 2.6 shows the temporal state-variable representation of Transport temporal, using a slightly shorter, but clearer notation. Note that both `pick-up` and

```
drive(v, l1, l2)
  ;; vehicle v moves from location l1 to an adjacent location l2
  precond: at(v) = l1, road(l1, l2)
  effects: at(v) <- l2

pick-up(v, l, p, s1, s2)
  ;; vehicle v picks up package p at location l,
  ;; decreasing its capacity from s2 to s1
  precond: at(v) = l, at(p) = l, s1 + 1 = s2, s2 > 0, capacity(v) = s2
  effects: at(p) <- nil, in(p) <- v, capacity(v) <- s1

drop(v, l, p, s1, s2)
  ;; vehicle v drops package p at location l,
  ;; increasing its capacity from s1 to s2
  precond: at(v) = l, in(p) = v, s1 = s2 - 1, capacity(v) = s1
  effects: in(p) <- nil, at(p) <- l, capacity(v) <- s2
```

Figure 2.4: State-variable formulation of `transport-strips`.

```
drive(v, l1, l2)
  ;; vehicle v moves from location l1 to an adjacent location l2
  precond: at(v) = l1, road(l1, l2),
           fuel-left(v) >= fuel-demand(l1, l2)
  effects: at(v) <- l2,
           fuel-left(v) <- fuel-left(v) - fuel-demand(l1, l2)

refuel(v, l)
  ;; vehicle v is refueled to the maximum at location l
  precond: at(v) = l, has-petrol-station(l)
  effects: fuel-left(v) <- fuel-max(v)
```

Figure 2.5: Partial state-variable formulation of `transport-numeric`. Shows the differences when compared to `transport-strips`.

`drop` cancel the `at` predicate at the start of the action, which forbids parallel picking up and dropping.

2.2.3 PDDL formulation of Transport

All formulations of the Transport domain use PDDL (Section 1.4) version 2.1, with the requirement `typing`, which adds the notion of types for individual literals. We will call these literals *action objects*.

The STRIPS variant additionally needs `action-costs`, a requirement adding integer costs to individual planning operators. These costs may be constant (like the ones for `pick-up`, `drop` or `refuel`), or they may be dependent on the parameters of the instantiated operator (like the cost of `drive`). The numeric variant requires `numeric-fluents`, which introduces native PDDL support for functions whose values correspond to numbers and can change over time. It

```
drive(v, l1, l2)
  ;; vehicle v moves from location l1 to an adjacent location l2
  duration: road-length(l1, l2)
  cond: (at(v) = l1)@s, (road(l1, l2))@s,
        (fuel-left(v) >= fuel-demand(l1, l2))@s
  effects: (at(v) <- nil)@s, (at(v) <- l2)@e,
           (fuel-left(v) <- fuel-left(v) - fuel-demand(l1, l2))@s

pick-up(v, l, p)
  ;; vehicle v picks up package p at location l
  duration: 1
  cond: (at(v) = l1)@[s, e), (at(p) = l1)@s, (ready-loading(v))@s,
        (capacity(v) >= package-size(p))@s
  effects: (at(p) <- nil)@s, (in(p) <- v)@e, (not ready-loading(v))@s,
           (capacity(v) <- capacity(v) - package-size(p))@s,
           (ready-loading(v))@e

drop(v, l, p)
  ;; vehicle v drops package p at location l
  duration: 1
  cond: (at(v) = l1)@[s, e), (in(p) = v)@s, (ready-loading(v))@s
  effects: (in(p) <- nil)@s, (at(p) <- l)@e, (not ready-loading(v))@s,
           (capacity(v) <- capacity(v) + package-size(p))@e,
           (ready-loading(v))@e

refuel(v, l)
  ;; vehicle v is refueled to the maximum at location l
  duration: 10
  cond: (at(v) = l1)@[s, e), (has-petrol-station(l))@s
  effects: (fuel-left(v) <- fuel-max(v))@e
```

Figure 2.6: Temporal state-variable formulation of temporal Transport. The characters `s` and `e` represent the start and end temporal variables of the given action, respectively.

also requires are `goal-utilities`, used for custom optimization functions and optional goal predicates. The temporal domain is similar in requirements to the numeric one, except for substituting `goal-utilities` for `durative-actions` (introduces time and the duration of actions).

For reference, the PDDL representations of sequential and temporal variants of the Transport domain are attached to this thesis in Attachment 2.

2.3 Related problems

To the best of our knowledge, there has been no attempt at producing domain-dependent planners for Transport (IPC 2008, 2011, 2014, and unsolvability IPC 2016) or any other similar IPC domain, like Logistics (IPC 1998 and 2000), Depots (IPC 2002), DriverLog (IPC 2002 and 2014), or Trucks (IPC 2006). All techniques

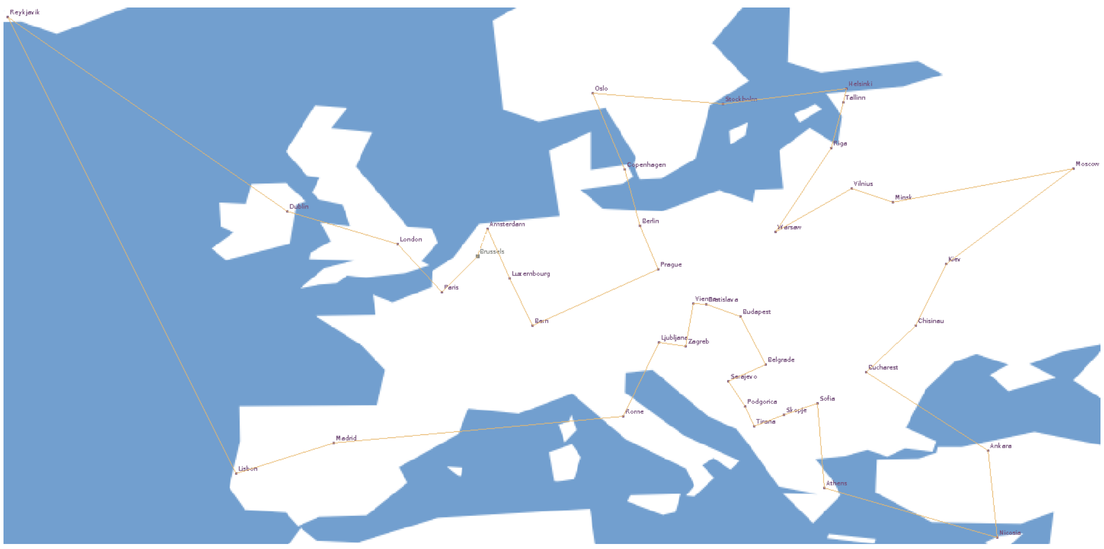


Figure 2.7: An example TSP solution through the capitals of Europe. Screenshot taken from OptaPlanner (De Smet et al., 2017).

we know of applied to Transport so far are the domain-independent planners used in the three aforementioned competitions.

Most of the research done on transportation related problems and their automation generally focuses on a famous combinatorial optimization problem, the *Traveling Salesman Problem* (TSP). An exhaustive amount of research has been done on the TSP (Applegate et al., 1998, 2011). Its precise origins are unknown, but the problem has been on the minds of researchers at least since the end of the 19th century. The TSP is defined by Applegate et al. (2011) as follows:

> Given a set of cities along with the cost of travel between each pair of them, the *traveling salesman problem*, or *TSP* for short, is to find the cheapest way of visiting all the cities and returning to the starting point. The "way of visiting all the cities" is simply the order in which the cities are visited; the ordering is called a *tour* or *circuit* through the cities.

However, the problem we aim to study is more similar to a different optimization problem similar to the TSP. See Figure 2.7 for an illustrative example of a TSP problem's solution.

2.3.1 The Vehicle Routing Problem

The *Vehicle Routing Problem* (VRP) was first formulated as the *Truck Dispatching Problem* by Dantzig and Ramser (1959), modeling a fleet of vehicles delivering gasoline to service stations. They described VRP as a generalization of the TSP with multiple vehicles, but it could equivalently be stated that the TSP is a specialization of the VRP with a single vehicle. The precise formulation of the Truck Dispatching Problem in Dantzig and Ramser (1959, Section 2) presents a model with a fleet of identical vehicles departing from a single depot. According to Braekers et al. (2016, Section 3), this defines what we would call *Capacitated VRP* (CVRP) today (Figure 2.8).

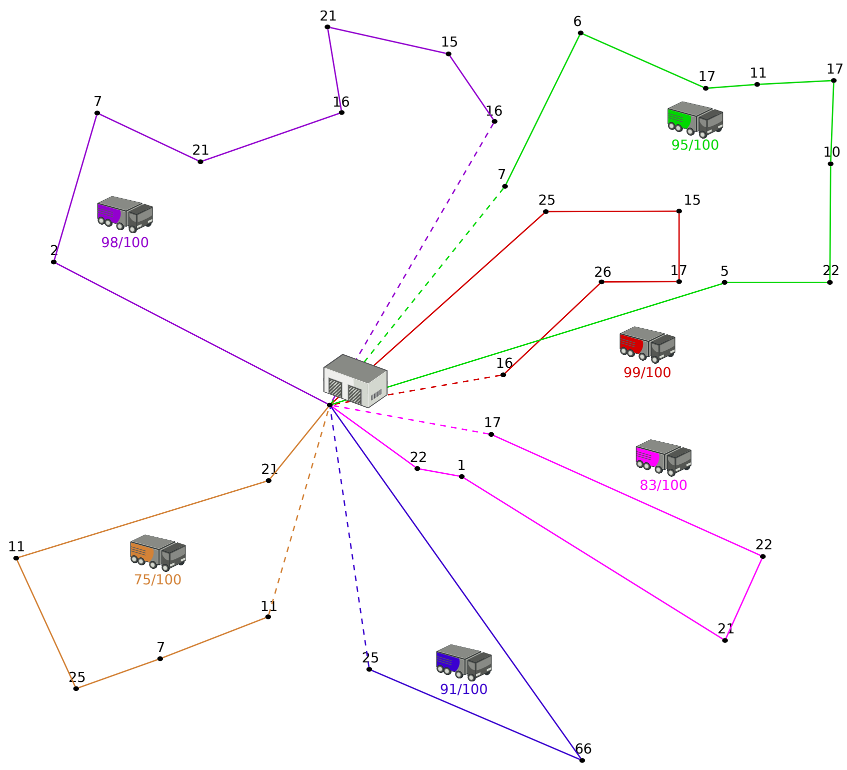


Figure 2.8: An example CVRP solution for 32 customers and one depot. The dashed line represents the last drive of each vehicle's route. Image adapted from OptaPlanner (De Smet et al., 2017).

Many VRP variants have emerged since. Eksioglu et al. (2009) and Braekers et al. (2016) both review and classify hundreds of papers related to the VRP, with many more left out. Most of the classified works tend to study the CVRP problem with minor modifications, hence creating a broad landscape of problems and a platform to build on in the future. According to the data provided by Braekers et al. (2016, Table 4), there has been a recent uptick in popularity for models relatively similar to Transport — specifically, VRPs with backhauls (returning items from customers to depots), multiple depots (multiple starting points for vehicles), and with allowed split deliveries (multiple vehicles can serve a single customer). The literature review on Multiple Depot VRP (MDVRP) in Montoya-Torres et al. (2015) suggests a big rise in popularity for MDVRP in the recent past, which provides further proof of relevance for studying the Transport domain.

Traditional solutions for the VRP include exact approaches like branch and bound or constraint satisfaction programming that explore the large parts of the feasible search space, classical heuristics which limit the search space, and also metaheuristics (general heuristics for devising specific heuristics), like genetic algorithms, tabu search, and many more.

2.3.2 Comparison of Transport and VRP

The work of NEO Research Group (2013) resulted in a website, which serves as a comprehensive resource on the history of VRP, definitions of its various flavors, and an overview of popular solution methods and state-of-the-art results. According to the taxonomy they propose, a Transport problem could be characterized as a *Multiple Depot, Split Delivery, Capacitated VRP with Satellite Facilities.* Multiple depot means that vehicles can start driving from multiple locations, split delivery means a single customer can be served by multiple vehicles, capacitated VRP adds maximum capacities to vehicles, and satellite facilities mean that vehicles can pick items up while on a delivery route. This does not characterize the Transport domain in every detail, but it is a fairly accurate approximation.

According to another VRP taxonomy and study of papers, presented in Eksioglu et al. (2009) and adapted in Braekers et al. (2016), no research has been done on a VRP variant with a similar subset of features to those of Transport in any single study, to the best of our knowledge. Usually, the studied problems are more constrained than Transport — for example, they make additional assumptions about places where vehicles start or end. Also, VRP in general makes cooperation of vehicles hard to model, whereas in Transport this is one of the fundamental elements.

Another important difference between Transport and the VRP is that Transport has a notion of single packages or items. In the VRP, transported goods are usually regarded as measurable, rather than countable (for example gasoline or milk vs. letters or parcels). This makes a difference not only in the interpretation, but also during problem-solving — *customers* in VRP usually request a quantity of the delivered item, not specific item instances, like packages being "requested" by their target locations in Transport.

3. Transport domain analysis

In this chapter, we will analyze the sequential and temporal variants of the Transport domain. To do this, we will describe the datasets used in our experiments, and discuss the domain's properties, which will help us when developing planners.

3.1 Problem complexity

When domain-independent planners solve a sequential Transport problem, they face a harder task than planners that have access to domain knowledge ahead of time. For domain-independent planners, deciding whether a plan of a given length exists (the PLAN-LENGTH decision problem) is a NEXPTIME-complete task. Deciding if a plan exists at all (the PLAN-EXISTENCE decision problem) is an EXPSPACE-complete task (Ghallab et al., 2004, Table 3.2).

That does not mean domain knowledge makes Transport easy, as is evident from the very thorough analysis by Helmert (2001a,b). We will categorize our problems using Helmert's notation to be able to apply their results to our domain. A TRANSPORT task is a 9-tuple $(V, E, M, P, fuel_0, l_0, l_G, cap, road)$, where:

- (V, E) is the road graph;
- M is a finite set of vehicles (mobiles);
- P is a finite set of packages (portables);
- $fuel_0 : V \to \mathbb{N}_0$ is the fuel function;
- $l_0 : (M \cup P) \to V$ is the initial location function;
- $l_G : P \to V$ is the goal location function;
- $cap : M \to \mathbb{N}$ is the capacity function; and
- $road : M \to 2^E$ is the movement constraints function.

V, M, and P are pairwise disjoint. No Transport domain variants assume movement constraints, therefore, $road$ is a constant function $\forall m \in M : road(m) = E$.

A simplified notation is also introduced in Helmert (2001b) for special cases of TRANSPORT tasks. For $i, j \in \{1, \infty, *\}$, $k \in \{1, +, *\}$ a TRANSPORT$_{i,j,k}$ task is defined as a general TRANSPORT task (defined above) that satisfies:

- if $i = 1$, then $\forall m \in M : cap(m) = 1$ (vehicles can only carry one package);
- if $i = \infty$, then $\forall m \in M : cap(m) = |P|$ (vehicles have unlimited capacity);
- if $j = 1$, then $\forall v \in V : fuel_0(v) = 1$ (one fuel unit per location);
- if $j = \infty$, then $\forall v \in V : fuel_0(v) = \infty$ (unlimited fuel per location);
- if $k = +$, then $\forall m \in M : road(m) = E$ (no movement restrictions); and
- if $k = 1$, then $M = \{m\}$ & $road(m) = E$ (single vehicle, no restrictions).

The $*$ value for i, j or k signifies no restriction on that property. Note that TRANSPORT refers to the notation from Helmert (2001b), while Transport refers to our studied domain.

Using this notation, the sequential Transport domain could be thought of as a TRANSPORT$_{c,\infty,+}$ task, where $c \in \mathbb{N}$ (equivalent to TRANSPORT$_{1,\infty,+}$). Similarly, the temporal variant represents a TRANSPORT$_{c,f,+}$ task, for $c, f \in \mathbb{N}$ (equivalent to TRANSPORT$_{1,1,+}$).

For sequential and temporal Transport without fuel, the PLAN-EXISTENCE problem reduces to verifying reachability of each package by at least one vehicle and the reachability of target locations from the starting locations of all packages, which we can do in polynomial time, as noted in Helmert (2001a, Theorem 8). With fuel, there is no straightforward way of determining if a plan exists and this problem is NP-complete, which is proven in Helmert (2001a, Theorem 9 and 10).

Even though fuel constraints are modeled differently than in Transport (constraints per location versus per vehicle), the proof of NP-completeness of PLAN-EXISTENCE for TRANSPORT$_{\infty,1,1}$ present in Helmert (2001b, Theorem 3.9) can be trivially edited to prove the NP-completeness of PLAN-EXISTENCE for temporal Transport. Instead of adding fuel conditions to the entrance and exit nodes of a location, we simply add it to road between them. The rest of the proof holds as was presented originally.

Similarly, the PLAN-LENGTH problem is NP-complete for all mentioned variants of Transport (Helmert, 2001b, Section 3.6). The fact that all the mentioned proofs work for temporal variants is explained in Helmert (2001b, Section 3.5). All of these results make clear that looking for an explicit planning algorithm is infeasible, despite the advantage we gain by only focusing on one planning domain.

3.2 Domain information

There are several interesting properties and invariants that hold in both sequential and temporal Transport, which might prove useful for designing planners:

1. **Do not pick up delivered packages**: The simplest and trivially correct decision is to never touch packages that are already at their destinations, since there is nothing we can do using those packages that would result in a plan with a lower total cost.

2. **Drop when at the destination**: Likewise, it is always correct for a vehicle containing a package with a destination equal to the vehicle's location to do a `drop` action immediately.

3. **Do not drop and pick up**: It never makes sense to plan a `drop` and `pick-up` action of the same package by the same vehicle in succession. We will only get to the same state by using a longer plan. This rule also applies if an action of a different vehicle gets between the two successive actions, even if it does an action with the dropped package. It is important to note that this is a symmetric property: picking up and then dropping equally results in a worse plan.

   (a) **Do not drop a package where we picked it up**: A generalization of the previous rule is that vehicles should never drop a package at the location they last picked it up, independent of the actions they took between the relevant `pick-up` and `drop`. This rule is also symmetric.

   (b) **Never drop after picking up at a location**: While the order of successive `pick-up` and `drop` actions does not influence the optimality of a plan, it makes the search space smaller and the implementation of these rules simpler, without loss of generality.

4. **Do not drive suboptimally**: If a vehicle does a series of `drive` actions from location A to B without "touching" packages or refueling at any of the locations it visits, it has to follow the shortest possible path from A to B. If it does not, the induced plan can be made less costly or shorter by swapping the actual `drive` actions for precalculated optimal `drive` actions along the shortest path. Do note, that it is not important for the application of this rule whether actions are in direct succession (in a sequential plan) or not.

   (a) **Do not drive in cycles**: A special but important case of the previous rule is that vehicles should not drive in cycles.

5. **Do not forward packages using other vehicles**: Let p be a package of size $|p|$ located at A. Let v be a vehicle which drove through location A to location $B \neq A$ and picked up p at B, without having less than $|p|$ free space in any intermediate state between leaving A and picking up p. If this sequence of events occurs, the plan is suboptimal in a sequential setting, because v could have picked up p when driving through A, and the total plan cost would have gone down by at least 2. The reason is that a different vehicle had to pick up, drive, and drop package p at B. While we cannot say if the `drive` actions themselves were redundant, the `pick-up` and `drop` actions definitely were. By removing them, we save 2 on the total cost.

   In a temporal domain without fuel, assuming that vehicles only drive along the shortest routes, the plan does not necessarily have to be suboptimal, but it is of equal length or longer: due to concurrent actions, the vehicles could have driven simultaneously. In a few cases, the other vehicle could have dropped p at B before v wants to pick it up, which means that the total makespan of the partial plan did not become longer, but stayed the same. The plan could not have become shorter, because v does not have any time in this scenario when it is available to do another action.

   In a temporal domain with fuel, it is not safe to say whether such a scenario hurts the plan duration. If there is a petrol station at B and v wants to refuel there, the other vehicle could have enabled the parallelization of the `refuel` and `pick-up` actions, therefore shaving off 1 time unit in total. However, if there is no petrol station at B, this situation reduces to the no-fuel variant. Given the relative rarity of petrol stations, this will reduce the search space somewhat.

Another insight can only be applied to the sequential variant of Transport:

1. **Drop from an active vehicle only**: Without loss of generality, we can prune all plans where a `drop` action of a vehicle happens right after an

action of a different vehicle. It is trivial to see that if we had a plan where a `drop` action occurs after an action of a different vehicle, we can swap that action with the `drop` action without changing the total plan cost or changing the validity of the plan.

Doing this repeatedly will yield an equivalent plan, in which the `drop` action occurs right after a different action of the same vehicle and the plan is of the same total cost and validity as the original plan. Repeating this process for each `drop` action will yield a plan equivalent to the original plan, which additionally satisfies this rule.

Finally, these are the properties that only meaningfully apply to the temporal variant:

1. **Refueling and dropping/picking up can occur at the same time**: A plan in which a vehicle starts to pick up a package at the same location it just refueled at is suboptimal, if there was a time point during the `refuel` action when the vehicle was not dropping or picking up packages and the package was already co-located with the vehicle at that time.

2. **No fuel left means refueling or ignoring the vehicle**: If a vehicle is stuck with no fuel left, or with less fuel than is required for any valid `drive` action, the correct thing to do is to either refuel or drop all packages and ignore the vehicle in further planning. Unfortunately, we cannot say anything about the (non-)optimality of a plan where this occurs.

3.3 Datasets & problem instances

For evaluation and comparison with other planners, we have acquired several problem datasets from previous runs of the IPC. Table 3.1 provides an overview of the individual datasets, their associated IPC competition, the track at the competition and the domain variant the problems are modeled in.

Short descriptions of the various tracks and subtracks can be found in the rule pages of IPC-6,[1] IPC-7,[2] and IPC-8.[3] We have decided to split our further research based on the tracks at the IPC: we will focus on constructing Transport-specific planners for the seq-sat-6, seq-sat-7, seq-sat-8, and tempo-sat-6 datasets, corresponding to the sequential and temporal variants of Transport.

The datasets labeled seq-opt correspond to sequential optimality planning tracks, where only optimal plans for problems are accepted as correct. Datasets labeled seq-mco are used in multi-core satisficing tracks (multi-threaded planners) and seq-agl are used in agile tracks (minimize the CPU time required to find a satisficing plan). The netben-opt-6 dataset contains Net Benefit problems, where the aim is to compensate between achieving *soft goals* and minimizing the total cost. Soft goals are goals that do not necessarily have to be satisfied in a goal state, but it is usually better for the total score if they are. Each problem usually specifies a metric used for calculation of the score. We will not focus on these problems in this work.

[1] `http://icaps-conference.org/ipc2008/deterministic/CompetitionRules.html`
[2] `http://www.plg.inf.uc3m.es/ipc2011-deterministic/CompetitionRules.html`
[3] `https://helios.hud.ac.uk/scommv/IPC-14/rules.html`

Dataset	Competition	IPC Track	Formulation
netben-opt-6	IPC-6	Net-benefit: optimization	Numeric
seq-opt-6		Sequential: optimization	STRIPS
seq-sat-6		Sequential: satisficing	STRIPS
tempo-sat-6		Temporal: satisficing	Temporal
seq-mco-7	IPC-7	Sequential: multi-core	STRIPS
seq-opt-7		Sequential: optimization	
seq-sat-7		Sequential: satisficing	
seq-agl-8	IPC-8	Sequential: agile	STRIPS
seq-mco-8		Sequential: multi-core	
seq-opt-8		Sequential: optimization	
seq-sat-8		Sequential: satisficing	

Table 3.1: Transport datasets from the 2008, 2011, and 2014 IPCs. All formulations assume capacitated vehicles. Numeric and temporal formulations also contain fuel demands and capacities. The temporal formulation additionally adds concurrent actions and a notion of time. More information can be found in Section 2.1.

In addition to the domain definition, we need to take a look at the individual problems to fully utilize our knowledge advantage. Both the seq-sat-6 and tempo-sat-6 contain 30 problems, while seq-sat-7 and seq-sat-8 only contain 20 problems each. Table 3.2 shows the dimensions of each problem instance for each mentioned dataset.

While the planners (including our domain-specific ones) do not know this, each problem was constructed with a scenario in mind. Locations in problems are not just placed randomly, but usually belong to cities. Inside a city, the road network tends to be dense and road lengths small, while roads connecting cities are rare and usually significantly longer.

All sequential problem instances in seq-sat datasets have symmetric roads and road lengths and can, therefore, be simplified by assuming the use of an undirected graph. All packages are always positioned at locations in the initial state, not in vehicles (in all domain variants).

The temporal problems in tempo-sat-6 do not have the same properties; the problems 1–20 have symmetric roads and lengths, but the 21–30 problems only have symmetric roads, not lengths in general. The same applies to fuel demands of roads. Additionally, these problems have vehicle target locations, which means that not only packages, but also vehicles will need to be positioned at specific locations after package delivery finishes. We can interpret this goal in a similar way as in a VRP, where a vehicle target location is thought to be a truck depot or hub. A visualization of such a problem can be seen in Figure 3.1. No sequential problem has this requirement, even though the domain formulation allows it.

Given a specific Transport problem, we can calculate the size of the set of states S. For sequential Transport, the state space size can be estimated as:

$$l^v \cdot (l+v)^p,$$

where l is the number of locations, v the number of vehicles, and p the number

#	Vehicles	Packages	Cities	Locations	Roads	States
1	2	2	1	5	12	10^{3}
2	2	4	1	10	28	10^{6}
3	3	6	1	15	66	10^{11}
4	3	8	1	20	70	10^{14}
5	3	10	1	25	92	10^{18}
6	4	12	1	30	138	10^{24}
7	4	14	1	35	136	10^{28}
8	4	16	1	40	164	10^{32}
9	4	18	1	45	170	10^{37}
10	4	20	1	50	210	10^{41}
11	2	2	2	6	12	10^{3}
12	2	4	2	12	32	10^{6}
13	3	6	2	18	58	10^{11}
14	3	8	2	24	88	10^{15}
15	3	10	2	30	132	10^{19}
16	4	12	2	36	168	10^{25}
17	4	14	2	42	152	10^{29}
18	4	16	2	48	192	10^{34}
19	4	18	2	54	220	10^{38}
20	4	20	2	60	256	10^{43}
21	2	2	3	6	12	10^{3}
22	2	4	3	12	30	10^{6}
23	3	6	3	18	64	10^{11}
24	3	8	3	24	74	10^{15}
25	3	10	3	30	114	10^{19}
26	4	12	3	36	166	10^{25}
27	4	14	3	42	170	10^{29}
28	4	16	3	48	248	10^{34}
29	4	18	3	54	234	10^{38}
30	4	20	3	60	226	10^{43}

(a) Problem dimensions of seq-sat-6.

#	Vehicles	Packages	Cities	Locations	Roads	Petrol	States
1	2	2	1	5	12	1	10^{8}
2	2	4	1	10	36	1	10^{11}
3	3	6	1	15	64	1	10^{19}
4	3	8	1	20	70	1	10^{22}
5	3	10	1	25	94	1	10^{26}
6	4	12	1	30	132	1	10^{35}
7	4	14	1	35	158	1	10^{39}
8	4	16	1	40	190	1	10^{43}
9	4	18	1	45	210	1	10^{48}
10	4	20	1	50	254	1	10^{52}
11	2	2	2	6	12	2	10^{8}
12	2	4	2	10	32	2	10^{11}
13	3	6	2	16	50	2	10^{19}
14	3	8	2	20	74	2	10^{22}
15	3	10	2	26	108	2	10^{27}
16	4	12	2	30	152	2	10^{35}
17	4	14	2	36	168	2	10^{39}
18	4	16	2	40	142	2	10^{43}
19	4	18	2	46	218	2	10^{48}
20	4	20	2	50	240	2	10^{52}
21	5	2	3	6	12	3	10^{18}
22	5	4	3	9	18	3	10^{21}
23	6	6	4	12	24	4	10^{28}
24	7	8	4	16	32	4	10^{36}
25	8	10	5	20	40	5	10^{44}
26	8	12	5	25	60	5	10^{49}
27	10	14	6	30	72	6	10^{62}
28	10	16	6	36	84	6	10^{67}
29	11	18	7	42	98	7	10^{76}
30	11	20	7	49	112	7	10^{81}

(b) Problem dimensions of tempo-sat-6.

#	Vehicles	Packages	Cities	Locations	Roads	States
1	4	16	1	40	82	10^{32}
2	4	18	1	45	85	10^{37}
3	4	18	3	54	117	10^{38}
4	4	12	2	36	84	10^{25}
5	4	14	2	42	76	10^{29}
6	4	16	2	48	96	10^{34}
7	4	18	2	54	110	10^{38}
8	4	20	1	50	105	10^{41}
9	4	20	2	60	128	10^{43}
10	4	20	3	60	113	10^{43}
11	4	22	1	50	92	10^{44}
12	4	20	1	53	101	10^{42}
13	4	22	1	53	101	10^{45}
14	4	22	2	120	229	10^{54}
15	4	20	2	124	239	10^{50}
16	4	22	2	124	239	10^{54}
17	4	22	3	180	347	10^{58}
18	4	20	3	189	361	10^{54}
19	4	22	3	189	361	10^{59}
20	4	22	3	198	392	10^{59}

(c) Problem dimensions of seq-sat-7.

#	Vehicles	Packages	Cities	Locations	Roads	States
1	4	25	1	50	200	10^{50}
2	4	30	1	53	212	10^{59}
3	4	25	1	53	212	10^{50}
4	4	25	2	136	512	10^{62}
5	4	30	2	134	536	10^{72}
6	4	25	2	136	512	10^{62}
7	4	25	3	204	764	10^{67}
8	4	30	3	201	796	10^{78}
9	4	25	3	204	764	10^{67}
10	4	25	3	198	794	10^{66}
11	5	25	1	60	238	10^{54}
12	4	30	1	63	254	10^{61}
13	3	25	1	63	254	10^{50}
14	5	25	2	120	454	10^{62}
15	4	30	2	124	500	10^{71}
16	3	25	2	124	500	10^{58}
17	5	25	3	180	696	10^{67}
18	4	30	3	189	764	10^{77}
19	3	25	3	189	764	10^{63}
20	5	25	3	198	794	10^{69}

(d) Problem dimensions of seq-sat-8.

Table 3.2: Problem dimensions of selected Transport IPC datasets. The "states" value is a state space size estimate as discussed in Section 3.3 (in temporal domains calculated with $f_{max} = 100$ and the GCD of `fuel-demands` equal to 1). Bold problem instances correspond to Figure 2.1 and Figure 3.1, respectively.

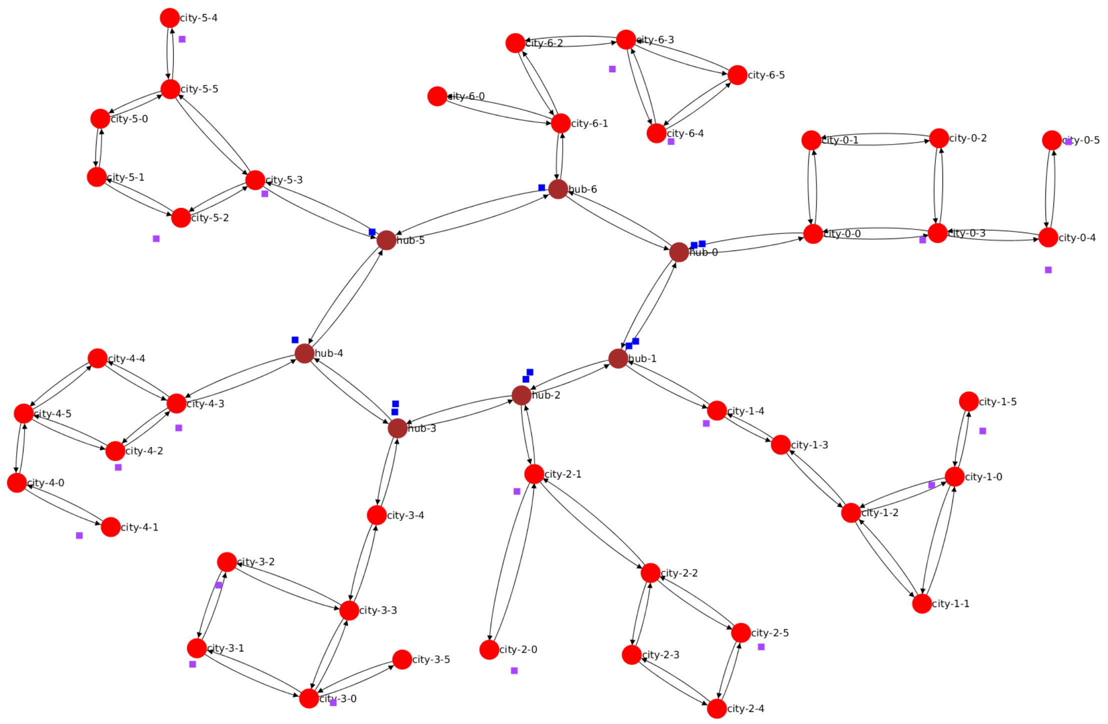


Figure 3.1: Road network visualization of the `p30` problem from the tempo-sat track of IPC 2008. Red dots represent locations (graph nodes), roads (graph edges) are represented by black arrows, vehicles are plotted as blue squares, and packages as purple squares. Darker red dots represent locations with petrol stations. In this specific problem, the circle of darker nodes in the center represents truck hubs and each of the attached subgraphs are individual cities.

Dataset	**Problems**	**Sym. road lengths**	**Sym. fuel demands**	**Vehicle fuel locations**	**$\approx$ # states**
seq-sat-6	01–30	Yes	N/A	No	$10^{3} \rightarrow 10^{43}$
seq-sat-7	01–30	Yes	N/A	No	$10^{25} \rightarrow 10^{59}$
seq-sat-8	01–30	Yes	N/A	No	$10^{50} \rightarrow 10^{78}$
tempo-sat-6	01–20	Yes	Yes	No	$10^{8} \rightarrow 10^{52}$
	21–30	No	No	Yes	$10^{18} \rightarrow 10^{81}$

Table 3.3: Summary of problem instance properties in IPC Transport datasets. State space size estimates in temporal domains are calculated using $f_{max} = 100$ and the GCD of `fuel-demand`s equal to 1.

of packages. The formula represents the number of choices for the location of vehicles, combined with the number of choices for the location of packages (these include being loaded onto a vehicle). We have eliminated invalid states arising from inconsistent in(p) and at(p) state variable values, but some invalid states are still left in the state size estimate (for example states, where vehicles are loaded beyond maximum capacity). We did not include a notion of capacity in this estimate because it can be computed from the locations of packages.

For temporal Transport, the problem state space size estimate is more complicated, due to actions being parallel. A reasonable estimate could be:

$$(l+r)^v \cdot \left(\frac{f_{max}}{\text{GCD}\{\text{fuel-demand}(l_1,l_2) | (l_1,l_2) \in R\}}\right)^v \cdot (l+v)^p,$$

where R represents the set of roads, $r = |R|$ is the number of roads, GCD is the greatest common divisor function, and f_{max} is the maximum fuel capacity for vehicles. The f_{max} value presents a simplification, where all vehicles have an equal maximum fuel capacity. The formula expresses the choice of positions of vehicles (vehicles can now be in the middle of a `drive` action), the choice of the current fuel capacity of vehicles (cannot be simply calculated from the other state variables, only from all previous actions), and the choice of location for packages.

We can see that problems vary not only in size but also in what features they include and what assumptions they make. A summary of the acquired dataset-specific insights is available in Table 3.3.

4. Sequential Transport planning

In this chapter, we describe the planning approaches we selected, implemented, and tested for the STRIPS variant of Transport. Throughout this process, we will leverage the acquired Transport domain knowledge as much as possible.

4.1 State-space forward planning

One of the most straightforward approaches to automated planning is forward search (Ghallab et al., 2004, Section 4.2) in state space (Figure 4.1). Although the algorithm is defined on a classical representation, it can be used on any planning problem, where we can:

- determine whether a state is a goal state or not;
- iterate over all actions applicable to a state; and
- compute a successor state by applying an action to the current state.

Forward search, despite its simplicity, is one of the most frequently used approaches for domain-independent planners. There are two key steps of the algorithm, which cause the most problems in practice: representing the applicable actions (step 6) and choosing the next action to apply (step 8).

Representing state transitions in step 6 is a technical problem of representing successor states in state space search (Russell and Norvig, 1995, Section 3.2). Due to limited memory, applicable actions are grounded from operators on demand, as are the corresponding successor states (Russell and Norvig, 1995, Section 3.4).

The forward search algorithm, in its specified form, is nondeterministic. If we knew which action to choose in step 8, we would know how to solve the planning problem. Since we generally do not know which action (state transition) to choose in a given state, the choice is usually delegated to a suitable search algorithm. When forward search is implemented using such a search algorithm, the result is a deterministic forward search algorithm.

4.1.1 Deterministic search algorithms

State space search algorithms are a heavily studied area of computer science and any reasonable search algorithm applied to forward search will yield results. Examples of such algorithms are Breadth-First Search (BFS), Depth-First Search (DFS), and many more (Russell and Norvig, 1995, Section 3.5). The choice of a search algorithm greatly influences the quality of resulting plans when applied to forward search.

As sizes of planning problems grow, choosing a search algorithm is even more crucial. Several well-performing algorithms on small problems (like BFS) exceed reasonable run times and become unusable for practical application on larger problems. An important and practically useful search algorithm withstanding larger problem sizes is the *A^* algorithm* (Figure 4.2) introduced in Hart et al. (1968).

```
Algorithm Forward Search
Input: a planning problem in a classical representation $\mathcal{P} = (S, O, \gamma, s_0, g)$
Output: a plan $\pi$
 1: function FORWARD-SEARCH($\mathcal{P}$)
 2:     $s \leftarrow s_0$
 3:     $\pi \leftarrow$ empty plan
 4:     loop
 5:         if $s$ satisfies $g$ then return $\pi$
 6:         $A_s \leftarrow \{a \mid o \in O, a$ is a ground instance of $o$ & precond$(a)$ is true in $s\}$
 7:         if $A_s = \emptyset$ then return failure
 8:         (nondeterministically) choose an action $a \in A_s$
 9:         $s \leftarrow \gamma(s, a)$
10:         $\pi \leftarrow$ append $a$ to $\pi$
```

Figure 4.1: A forward search planning algorithm for Transport. Adapted from Ghallab et al. (2004, Figure 4.1).

A* has many important properties. We pinpoint one important to us, namely its admissibility. A search algorithm is *admissible* if it is guaranteed to find an optimal path from a state s to a goal state s_g for any state space (Hart et al., 1968). A* is admissible and optimal given an admissible heuristic.

An *admissible* heuristic never overestimates the true value it is approximating. During planning in state space, when examining a state s, we want to estimate the total cost of the best plan that gets us to a goal state from state s. In other words, because we are trying to minimize the total cost, a planning heuristic $h : S \rightarrow \mathbb{N}_0$ is admissible if and only if:

$$\forall s \in S : h(s) \leq h^*(s),$$

where h^* is the true total cost (i.e. the optimal heuristic). A similar definition is applicable for minimizing makespan in the temporal variant.

Furthermore, some heuristics have the property of being consistent. A heuristic h is consistent if and only if:

$$\forall s \in S : h(s) \leq \mathit{cost}(a) + h(s_n),$$

where $a \in A : \gamma(s, a) = s_n$, and for all goal states s_g, it holds that $h(s_g) = 0$. Consistent heuristics are sometimes called monotonic, because their value does not increase along the best path to a goal state. Additionally, it can be proved that consistent heuristics are always admissible (Russell and Norvig, 1995, Section 4.1).

A slight modification of A*, called *Weighted A** (Pohl, 1970), tends to yield good quality plans in a shorter amount of time, at the expense of sacrificing admissibility. The only difference when compared to A* is that the heuristic $h(x)$ is substituted for $h_w(x) = w \cdot h(x)$, where $w \in \mathbb{N}_0$ is a weight constant. Choosing a weight greater than 1 makes the heuristic inadmissible, but guides the search towards a goal state faster.

Algorithm Forward Search with A*

Input: a classical planning problem $\mathcal{P} = (S, O, \gamma, s_0, g)$, a heuristic h

Output: a plan π

```
 1: function COLLECT-PLAN($s, \pi$)
 2:     $\pi' \leftarrow$ empty list
 3:     while $s \neq \emptyset$ do
 4:         $(s', a) \leftarrow \pi[s]$
 5:         $\pi' \leftarrow$ prepend $a$ to $\pi'$
 6:         $s \leftarrow s'$
 7:     return $\pi'$
 8: function FORWARD-SEARCH-ASTAR($\mathcal{P}$)
 9:     $\pi \leftarrow$ empty map, $f[*] \leftarrow \infty$, $g[*] \leftarrow \infty$, $o \leftarrow \{s_0\}$, $c \leftarrow \emptyset$
10:     $\pi[s_0] \leftarrow \emptyset$, $g[s_0] \leftarrow 0$, $f[s_0] \leftarrow h(s_0)$
11:     while $o \neq \emptyset$ do
12:         $s \leftarrow \text{argmin}_{s' \in o} f[s']$, $o \leftarrow o \setminus \{s\}$, $c \leftarrow c \cup \{s\}$
13:         if $s$ satisfies $g$ then return COLLECT-PLAN($s$, $\pi$)
14:         for all actions $a \in$ GENERATE-ACTIONS($s, \pi$) do
15:             $s_n \leftarrow \gamma(s, a)$                                  ▷ Neighbor state
16:             if $s_n \notin c$ then                                     ▷ Not visited yet
17:                 if $s_n \notin o$ then $o \leftarrow o \cup \{s_n\}$          ▷ Discovered a new state
18:                 if $g[s] + \text{cost}(a) < g[s_n]$ then                 ▷ Found a better path
19:                     $\pi[s_n] \leftarrow (s, a)$
20:                     $g[s_n] \leftarrow g[s] + \text{cost}(a)$
21:                     $f[s_n] \leftarrow g[s_n] + h(s_n)$
        return failure
```

Figure 4.2: A forward search planning algorithm using A*. GENERATE-ACTIONS is a function that produces actions applicable to the state s. The notation $x[*] \leftarrow y$ represents initialization of all values of the map x to y.

4.1.2 Heuristics for forward search in Transport

When designing a heuristic, we want to provide an estimate of the total plan cost or makespan that is as precise as possible, which will help guide the search to a goal state as quickly as possible.

We will now describe several heuristics for sequential Transport using the state-variable representation. In the following, the value of the target(p) function represents the target location of a package p in the set of packages P.

Trivially admissible Transport heuristic

The simplest domain-specific heuristic that is applicable to all variants of Transport, apart from the zero heuristic $h_0 \equiv 0$, is one that counts the minimum number of `pick-up` and `drop` actions necessary to reach a goal state.

To obtain the correct count, we simply add 1 for each package that is not yet at its destination (it will need to be dropped there), and another 1 for each package that is, additionally, not in a vehicle (it will need to be picked up):

$$h'_0(s) = \sum_{\substack{p\in P\\ \mathrm{at}(p)\neq\mathrm{target}(p)}} 1 + \sum_{\substack{p\in P\\ \mathrm{at}(p)\neq\mathrm{target}(p)\\ \mathrm{at}(p)\neq\mathrm{nil}}} 1.$$

The heuristic h'_0 is admissible, but it is practically unusable, as it very poorly approximates the cost of the optimal remaining actions to a goal state — recall that costs of `drive` actions are generally much higher than the costs of `pick-up` and `drop` actions.

Package distance heuristic

In `transport-strips`, the only thing we want is to deliver packages to their destinations. Therefore, a straightforward heuristic is one that calculates the length of a shortest path of each package to its destination and sums the lengths for all packages. To make the heuristic more precise, we can add the value of h'_0 to it, as the `pick-up` and `drop` actions also have to occur in the optimal plan:

$$h_1(s) = h'_0(s) + \sum_{p\in P} \mathrm{spd}(\mathit{location}(p), \mathit{target}(p)),$$

where the $\mathit{location} : P \to L$ function, with values in the set of all locations L, is defined as:

$$\mathit{location}(p) = \begin{cases} \mathrm{at}(p), & \text{if } \mathrm{at}(p) \neq \mathrm{nil},\\ \mathrm{at}(\mathrm{in}(p)), & \text{else.} \end{cases}$$

The location of a package is, therefore, defined as the location it is at, or, if it is loaded in a vehicle, the location of the vehicle. The function $\mathrm{spd} : L \times L \to \mathbb{N}_0$ represents the shortest path distance between the two locations.

This heuristic is definitely not optimal, meaning that there are states, where we will need to add actions to reach a goal state with a higher total cost than the value of the heuristic in that state.

However, it is important to note, the heuristic is not even admissible, so its value might sometimes overestimate the total cost needed. To see why, let us consider a network with just two locations A and B. A vehicle of capacity 2 and two packages are located at A and both packages want to be transported to B. The road between A and B is symmetric and has length of a 1. It is trivial to see that the optimal plan consists of two `pick-up` actions, followed by a `drive` and two `drop` actions. This plan has a total cost of $2 + 1 + 2 = 5$, but the heuristic would estimate that we need actions that cost 6.

Minimum spanning tree marking heuristic

A different extension of the h'_0 heuristic is the heuristic h_2, based on finding the shortest paths on a *minimum spanning tree* (MST). Using an MST calculated by the algorithm presented in Kruskal (1956), we can solve one of the largest problems with h_1, namely that we count an excessive amount of `drive` actions for packages to their targets.

For each package p, the h_2 heuristic calculates shortest path distances to the package's target, but only using roads in the MST. Also, we do not calculate any

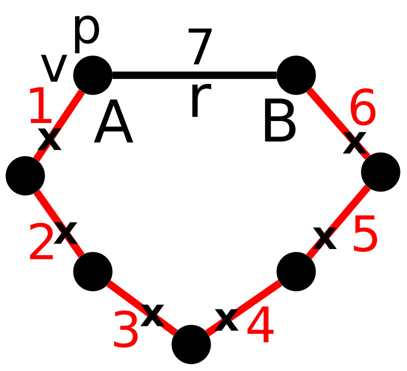


Figure 4.3: Visualization of the counterexample road network for admissibility of the heuristic h_2 for $n = 7$. Red roads represent the MST, roads marked with a cross were marked by h_2.

road twice — instead of adding its length directly to the total value, we mark it. After marking such roads for each package, we sum the lengths of all marked roads. In the same way as in h_1, we add the value of h'_0 to the final sum.

Do note that this heuristic is yet again inadmissible: let the road network be a circle of n locations with roads of lengths $1, 2, \ldots, n$ assigned clockwise. An MST on this network consists of all roads except the road with length n, let us denote the road $r = (A, B)$. Now, assume there is a package p located at A, with a target location of B, and a vehicle v, also located at A. The optimal plan is, obviously, to pick up the package at A, drive along road r to B, and drop the package there. The plan has a cost of $n + 2$. The h_2 heuristic will, however, mark all roads except r, because that is the only path from A to B in the MST. Therefore, in the initial state, the estimate given by h_2 will be:

$$h_2(s_0) = 2 + \sum_{i=1}^{n-1} i = \frac{n^2}{2} - \frac{n}{2} + 2,$$

which is evidently greater than $n + 2$ for $n > 3$. Figure 4.3 shows an example of such a road network for $n = 7$.

Package and vehicle distance heuristic

As an extension of the package distance heuristic, we will also add the distance of the nearest vehicle for each package:

$$h_3(s) = h_1(s) + \sum_{p \in P} \min_{v \in V} \operatorname{spd}(\mathit{location}(p), \operatorname{at}(v)),$$

where V is the set of all vehicles. As follows from the inadmissibility of h_1, h_3 is also inadmissible and non-optimal.

Package or vehicle distance heuristic

A variation on the package and distance heuristic is one that does not sum the shortest path distances, but instead takes the minimum. Specifically, for each package, the minimum is taken from the distance to its target location, distance to the nearest vehicle, and the distance to the nearest package:

$$h_4(s) = h'_0(s) + \sum_{p \in P} \min\{\mathrm{spd}(location(p), target(p)), \min_{v \in V} \mathrm{spd}(location(p), \mathrm{at}(v)),$$
$$\min_{\substack{p' \in P \\ p' \neq p}} \mathrm{spd}(location(p), location(p'))\}.$$

We will now show the admissibility of this heuristic.

Theorem 1. *The heuristic h_4 is admissible for sequential Transport problems.*

Proof. Let s be a state such that $h_4(s) > h^*(s)$, where h^* is a function of the real distance to the nearest goal state. State s is not a goal state, because for all goal states s' it holds that $h_4(s') = 0 = h^*(s')$, due to all packages being at their target locations. Let s_g be the nearest goal state to s.

Because $h_4(s) > h^*(s)$, there exists a finite plan π from the initial state s that ends in s_g, such that the total cost of π is equal to $h^*(s)$. Let $h_4^{(p)}(s)$ denote the value of a package's term in the sum of $h_4(s)$, ignoring the contribution of h'_0. Let $p \in P$ be any package that is not yet at its destination, not loaded in a vehicle, and not at a location with any other packages (all of those have $h_4^{(p)}(s) = 0$).

In the plan π, p had to be delivered. That means a vehicle (possibly more) had to arrive at the package's location, pick it up, drive it somewhere else and drop it, possibly several times. However, any vehicle could not have arrived at p's current location from a location that is closer than:

$$\min\{\min_{v \in V} \mathrm{spd}(location(p), \mathrm{at}(v)), \min_{\substack{p' \in P \\ p' \neq p}} \mathrm{spd}(location(p), location(p'))\} \leq h_4^{(p)}(s),$$

and at least one vehicle had to arrive at the package's location. The `pick-up` and `drop` actions for each package have to be planned in π as well, at least in the corresponding counts that h'_0 adds.

This means, that π had to have actions that correspond at least to the costs of:

$$h'_0(s) + \sum_{p \in P} h_4^{(p)}(s) = h_4(s),$$

which implies $h_4(s) \leq cost(\pi) = h^*(s)$, and that is a contradiction. $\square$

Despite its admissibility, the h_4 heuristic is not consistent. Let us assume the following road network (Figure 4.4a):

$$\{\{A, B, 1\}, \{B, C, 2\}, \{B, D, 2\}\},$$

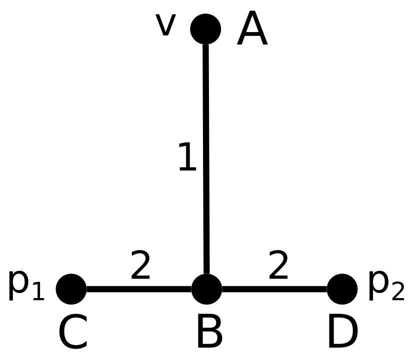
where $\mathrm{at}(v) = A$, $\mathrm{at}(p_1) = C$, and $\mathrm{at}(p_2) = D$ hold in the initial state s_0. The target of p_1 is D and the target of p_2 is C. The value:

$$h_4(s_0) = 3 + 3 + 4 = 10$$

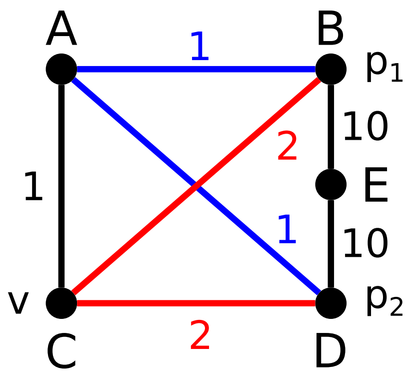
is greater than the cost of the applicable `drive` action for vehicle v from A to B added to the heuristic value in the updated state:

$$cost(a) + h_4(s') = 1 + (2 + 2 + 4) = 9.$$

The heuristic is also not optimal, as is evident from the same example situation (for s_0, the optimal plan is of length 13).



(a) Counterexample for h_4.



(b) Counterexample for h_5.

Figure 4.4: Visualization of the counterexample road networks for consistency of heuristics h_4 and h_5.

General marking heuristic

The general marking heuristic h_5 is a generalization of the MST marking heuristic h_2. We calculate h_5 in the exact same way by marking roads on shortest paths, but we do not use the calculated MST — instead, the whole road network is used. Essentially, this heuristic behaves like the the package or vehicle distance heuristic h_4, without adding any road's length to the sum more than once.

When calculating the value of h_5, we consider the shortest path from each package to the nearest vehicle, package, or target location. Roads on the shortest path from those are marked and the weights of marked roads are summed after we mark the shortest path for each package. Shortest path ties are broken arbitrarily. Do note that if a package is loaded in a vehicle, no roads are marked for it.

The described heuristic is not optimal, but it is admissible. Both properties hold because $\forall s \in S : h_5(s) \leq h_4(s)$ holds and they hold for h_4.

Even though h_5 is consistent in far more situations (combinations of states and applicable actions) than h_4, it is still not consistent in general. Assume the following road network (Figure 4.4b):

$$\{\{A, B, 1\}, \{A, C, 1\}, \{A, D, 1\}, \{B, C, 2\}, \{B, E, 10\}, \{C, D, 2\}, \{D, E, 10\}\},$$

where $\text{at}(v) = C$, $\text{at}(p_1) = B$, and $\text{at}(p_2) = D$ hold in the initial state s_0. The target of both p_1 and p_2 is E. If the heuristic selects the red roads for s_0, the value:

$$h_5(s_0) = 2 + 2 + 4 = 8$$

is greater than the cost of the applicable `drive` action for vehicle v from C to A added to the heuristic value in the updated state (assuming selection of blue roads):

$$cost(a) + h_5(s') = 1 + (1 + 1 + 4) = 7.$$

Note that the h_5 heuristic could have likewise selected the blue roads along with the $\{A, C, 1\}$ road, which would have resulted in $h_5(s_0) = 7$, and this case would not be a counterexample. This points to the possibility that the h_5 heuristic will work reasonably "consistently" in practice.

Summary

In our tests, we found that the heuristics h_0, h'_0, h_1, and h_2 were too simple for practical usage. The h_3 heuristic, while having no significant theoretical properties performs surprisingly well in practice. The h_4 heuristic on the other hand, while being admissible, does not perform well empirically. Due to its closeness to consistency, the h_5 heuristic has the best properties out of all the discussed heuristics and also performs well in practice. We will include planners utilizing the h_3 and h_5 heuristics in our final evaluation (Chapter 6).

4.1.3 Sequential Forward A*

*Sequential Forward A** (SFA) is a planner for sequential Transport based on forward search using A* (Figure 4.2). It utilizes most of the domain knowledge described in Section 3.2 and 3.3 to prune the search space as much as possible without sacrificing admissibility.

However, domain knowledge alone does not prune away enough search space to generate plans efficiently. With the addition of heuristics from Section 4.1.2, implementations of the planner become reasonably useful on practical problems, as will be demonstrated later during experimental evaluation.

To limit memory usage, we add an if statement to line 17 of the algorithm (Figure 4.2), which checks if the open set is larger than a given hyperparameter. If it is, it removes the state with the largest f value in the open set, making room for the addition of the new state. In our experiments, the hyperparameter is always set to 800 000.

A variant of SFA, *Weighted Sequential Forward A** (WSFA) swaps A* search for Weighted A* in the SFA planner.

4.1.4 Meta-heuristically weighted SFA*

*Meta-heuristically weighted SFA** (MSFA) is a meta-planner built on top of a WSFA planner with a given heuristic.

Given two hyperparameters $\alpha \in [0,1]$ and $w_0 \in [1,\infty)$, it runs the WSFA planner with the heuristic weight $w \leftarrow w_0$, waits for the planner to find a plan, and then (exponentially) decays the weight of the heuristic function with a minimum at 1:

$$w \leftarrow \max(1, \lfloor \alpha \cdot w \rfloor).$$

Do note that this is followed up by a complete restart of search in the internal WSFA planner. If only a recalculation of values in the f map of forward search with A* was done, all successive weight runs would almost immediately return, because we are in the vicinity of a goal state, but not necessarily the nearest one to the initial state. In some sense, this simulates a quick DFS run followed up by local search around the path found by DFS. Generally, better results are obtained by reexploring the state space from the initial state with an updated weight.

Even though this meta-planning approach breaks admissibility guarantees, it works very well on practical problems, even the larger ones.

Algorithm Randomized Restart planning

```
Input: a Transport problem in a classical representation $\mathcal{P} = (S, O, \gamma, s_0, g)$
Output: a plan $\pi$
function RANDOMIZED-RESTART($\mathcal{P}$)
    $\pi \leftarrow$ empty plan, $\Pi \leftarrow \infty$
    while cancel not requested do                ▷ Canceled by an external request
        $s \leftarrow s_0$, $\pi' \leftarrow$ empty plan
        while $s$ doesn't satisfy $g$ & $score(\pi') < \Pi$ do
            $A \leftarrow$ GENERATE-DRIVE-SEQUENCE($s$)
            for all action $a \in A$ do              ▷ Apply all actions to the state
                $s \leftarrow \gamma(s, a)$
            if $s = \emptyset$ then break            ▷ At least one $a \in A$ was not applicable
            $\pi' \leftarrow$ append all $a \in A$ to $\pi'$
        if $s$ satisfies $g$ & $score(\pi') < \Pi$ then
            $\pi \leftarrow \pi'$, $\Pi \leftarrow score(\pi')$          ▷ Update the best plan
    return $\pi$
```

Figure 4.5: A Randomized Restart planning algorithm. The GENERATE-DRIVE-SEQUENCE function generates a partial plan beginning in state s. The exact definition of the function depends on the specific algorithm variant as described in Section 4.2.1.

4.2 Ad-hoc planning

An approach that domain-independent planners by definition cannot utilize is ad-hoc planning. We propose several ad-hoc planners that generate decent quality plans very fast, although they are usually suboptimal. This becomes an advantage mainly when dealing with very large problems or time-constrained planning, like in the agile track of IPC.

4.2.1 Randomized Restart Planning

We will now describe a family of planners which we will refer to as *Randomized Restart* planners. Each of these planners essentially performs the same algorithm (Figure 4.5), with minor tweaks. The motivation behind the algorithm is that the "hardest" part of Transport planning is choosing where to drive with what vehicle.

The algorithm essentially does domain-dependent plan space planning by iteratively adding sequences of `drive` actions intertwined with `pick-up` and `drop` actions until all packages are delivered. To do all of this at speed, the vehicle used for the given sequence is usually chosen randomly, as are the packages that are picked up and dropped along the driving path. The biggest advantage, however, is gained by precomputing a matrix of shortest paths and always using those to drive to the selected locations.

Our randomized restart planners will generate many suboptimal plans, but by always keeping a copy of the so far best found plan, we can prune away the current partial plan if it becomes worse than the best plan after adding a sequence.

Afterwards, we simply restart from the initial state and iteratively generate a new candidate plan. We utilize domain knowledge in all the planners, for example by not picking up packages that are at their destination, and using most of the other insights discussed in Section 3.2.

Randomized Restart From Location Planner

In each iteration of the aforementioned algorithm, the Randomized Restart From Location planner (RRFL) randomly chooses a vehicle and a location. It adds `drive` actions along the shortest path to that location and greedily picks up as many packages as its capacity allows. It then calculates the optimal path through all the picked up package target locations, by trying the shortest paths through all possible permutations of the locations. Finally, it adds the appropriate `drive` and `drop` actions.

Randomized Restart On Path Planner

The Randomized Restart On Path planner (RROP), randomly chooses a vehicle and a package that fits into the vehicle. It then calculates the shortest path from the vehicle to the package and the shortest path from the package to its target location. Afterwards, it finds all packages that have current and target locations on the path, taking into account the direction of driving. It then tries all combinations of those packages that do not make the vehicle become over capacitated over the whole course of the path, in order to find the combination that maximizes the minimum free capacity over the course of the path. If there are several such combinations, it minimizes the maximum package drop location index in the path. If there are still several combinations, it chooses one arbitrarily. Finally, it adds the appropriate `drive` and `drop` actions.

Randomized Restart On Path Nearby Planner

The Randomized Restart On Path Nearby planner (RROPN) is a slight modification of RROP. It first chooses a package randomly and then, based on a probability $\varepsilon \in [0, 1]$, either chooses a random vehicle that the package fits into, or the nearest such vehicle. The rest of the algorithm is identical to RROP. The probability ε is a hyperparameter, which in our experiments is fixed to 0.2, which means that the probability of selecting the nearest vehicle is 80%.

Randomized Restart Around Path Nearby Planner

Another randomized restart planner that is built on top of RROPN, is the Randomized Restart Around Path Nearby planner (RRAPN). It uses the same vehicle and package selection, but it changes the way packages on the path are chosen to be loaded onto the vehicle.

We still load as many packages that are completely on the path as possible, using the same selection mechanism as in RROPN. Additionally, for each package we precalculate the location on the path that is closest to its target, limited to locations after the package was picked up in the correct driving direction. Finally, we plan the `pick-up` and `drop` actions of the packages that greedily fit into the vehicle and have the smallest precalculated distance to their target.

Randomized Backtrack Around Path Nearby Planner

We also developed a backtracking variant of RRAPN called Randomized Backtrack Around Path Nearby (RBAPN). Instead of choosing random vehicles and packages it backtracks over the choices it makes, guaranteeing to find an optimal plan in the subspace of plans generated by adding such action sequences as described in the section about RRAPN. This approach is very time-consuming and not practically usable.

Randomized Restart Around Path Distribution Planner

Another variant of RRAPN, the Randomized Restart Around Path Distribution planner (RRAPD), changes the vehicle choice inherited from RROPN. Instead of using a biased coin flip based on the ε hyperparameter, RRAPD samples the vehicle from a discrete probability distribution obtained by applying the *softmax* function on the inverse distances of vehicles to the selected package:

$$\begin{aligned}\varphi_i' &= \frac{1}{\mathrm{spd}(location(p), \mathrm{at}(v_i)) + 1},\\ \varphi_i &= \frac{\exp(\varphi_i'/T)}{\sum_{j=1}^{|V|} \exp(\varphi_j'/T)}.\end{aligned}$$

The value φ_i is the resulting probability of each vehicle $v_i \in V = \{v_1, v_2, \ldots, v_n\}$. The hyperparameter $T \in (0, \infty)$ is the *temperature* of the softmax. If set to higher values than 1, it evens out the distribution — higher probabilities shrink and smaller probabilities become larger. As T grows large, the distribution will resemble a uniform distribution. If T is set to lower values, the distribution will prefer larger values. In our experiments, we used a fixed $T = 0.1$, to prefer the nearest vehicles.

Summary

The implementations of RRFL, RROP, and RROPN have empirically shown worse performance in our tests than RRAPN and RRAPD. RRAPD, however, performs worse than RRAPN on average. As mentioned previously, RBAPN is unusable on larger problems due to its long run time in practice. We will evaluate the Nearby planner of the Randomized Restart Around Path kind in our experiments in Chapter 6, as they produce good quality plans in a very short time even for larger problem instances.

All planners of the Randomized Restart family are suboptimal, not only meaning that they sometimes produce suboptimal plans, but also that for some problems, even the best plan they can produce will be suboptimal. This is easiest to prove for each planner individually, by constructing a counterexample for a corner case of the greedy choices the planners make.

5. Temporal Transport planning

The temporal domain variant not only has the added challenge of time, but fuel demands and vehicle target locations are also present. In this chapter, we describe planning approaches used for tackling these additional challenges.

5.1 Scheduling actions of sequential plans

A simple temporal planning technique that is surprisingly effective in practice is one that simply forgets about time, finds a plan, and reintroduces time and concurrency as an afterthought into the generated plan. The sequential variant of Transport, however, also does not assume fuel. As a result, not all valid sequentially relaxed plans are valid temporal Transport plans when scheduled.

To be able to precisely formulate our algorithm, we have to define the concept of a *mutex* first. We say that a pair of instantiated temporal operators a, b is in a *mutual exclusion* relation (mutex) if and only if a and b cannot overlap in a valid plan. For example, a pair of `pick-up` actions of the same vehicle is mutex, because the at start effect of each action is to set the `ready-loading` predicate to false and the action has an at start condition that requires `ready-loading` to be true.

Leveraging our domain knowledge, we pre-construct mutex relations for Transport. The following actions are mutex:

- any pair of actions of the same vehicle, except a `refuel` and `pick-up`/`drop` pair (in any order); and
- a `drop` and `pick-up` pair of the same package (in any order).

Since we only schedule valid sequential plans, it cannot happen that a package is scheduled to be concurrently loaded into two different vehicles — in such sequential plans, pairs of `pick-up` and `drop` actions for a single package do not overlap each other. Therefore, a mutex relation will be constructed between the `drop` action of the first pair and the `pick-up` action of the second pair.

Our algorithm starts with relaxing the temporal problem to a sequential one by removing fuel demands and any notion of time. After running a sequential planner on the relaxed problem, we schedule the plan by finding a topological ordering of a directed acyclic graph (DAG) of actions, where edges are specified by mutex relations. Using the graph, we add actions to the temporal plan at the earliest available time, based on the topological order.

The DAG described above is constructed from a sequential plan π by adding all actions of the plan as nodes of the graph and all mutex relations as edges. The direction of added edges is based on the order the actions of a mutex appear in the original plan π.

To find a topological ordering of the mutex DAG, we use the algorithm described in Kahn (1962). We traverse actions of the sequential plan π (nodes of the DAG) in topological order and add each action a to the temporal plan π^T at the maximum end time in π^T of all actions who are mutex with a, where a is second in the pair. Thanks to topological ordering, all of these actions are already

Algorithm Sequential plan scheduling

```
Input: a temporal Transport problem $\mathcal{P} = (S, O, \gamma, s_0, g)$, a sequential plan $\pi$
Output: a temporal plan $\pi^T$
 1: function Build-Mutex-Graph($\pi$)
 2:     $G \leftarrow$ empty directed graph
 3:     for all action $a \in \pi$ do
 4:         $G \leftarrow$ add node $a$ to $G$
 5:     for $i \leftarrow 1$ to $|\pi|$ do
 6:         for $j \leftarrow i$ to $|\pi|$ do
 7:             $a \leftarrow \pi[i]$, $b \leftarrow \pi[j]$
 8:             if $veh(a) = veh(b)$ and not pick-up/drop and refuel then
 9:                 $G \leftarrow$ add edge $(a, b)$ to $G$
10:             else if $a$ and $b$ are pick-up and drop actions then
11:                 $G \leftarrow$ add edge $(a, b)$ to $G$
12:     return $G$
13: function Schedule($\mathcal{P}$, $\pi$, $\delta$ $(= 0.001)$)
14:     $\pi^T \leftarrow$ empty temporal plan, $t_s, t_e \leftarrow$ empty maps
15:     $\pi \leftarrow$ Add-Refuel-Actions($\mathcal{P}$, $\pi$)
16:     if $\pi$ is failure then return failure
17:     $G \leftarrow$ Build-Mutex-Graph($\pi$)
18:     $G' \leftarrow$ Topological-Sort($G$)
19:     for all action $b \in G'$ do
20:         $t_s[b] \leftarrow \delta + \max_{(a,b) \in G'} t_e[a]$        ▷ If no such edge exists, defaults to 0
21:         $t_e[b] \leftarrow t_s[b] + duration(b)$
22:         $\pi^T \leftarrow$ append $(t_s[b], t_e[b], b)$ to $\pi^T$
23:     if $\exists$ vehicle $v$ : fuel-left$(v) < 0$ at any point during $\pi^T$ then return failure
24:     return $\pi^T$
```

Figure 5.1: A scheduling algorithm for temporal planning using sequential planners. The maps t_s and t_e represent the assigned start and end times of planned actions. The parameter $\delta > 0$ is a constant used for separating different actions. Topological-Sort returns the same graph, but with nodes sorted using Kahn's algorithm. Add-Refuel-Actions is briefly described in Section 5.1.

planned in π^T and hence we know their end times. The described algorithm is summarized in Figure 5.1.

Previously left out was an important step of the algorithm, which consists of adding fuel constraints back into the plan. Just before constructing the mutex DAG, we attempt to add the least amount of `refuel` actions into the plan, so that no fuel capacity constraints are broken. This is done iteratively for each vehicle, by looking at combinations of possible places to refuel for that vehicle in a plan. This approach, however, does not always lead to a valid plan, because not all sequential plans go through enough locations that have a petrol station to allow all vehicles to be sufficiently fueled. The problem can be avoided by simply giving up on the current plan and attempting to generate another sequential plan. We keep track of the best-known plan and always update it if we find a better scheduled and valid plan. Despite this, some combinations of sequential planners and specific problem instances might never generate a sequential plan that when scheduled produces a valid temporal plan (due to fuel capacities of vehicles).

A big advantage of using a scheduling approach to temporal planning is that it is possible to reuse sequential planners which have already been created and optimized. The choice of an underlying planner makes a big difference in the quality of scheduled plans. Planners that generate various different plans generally perform better in such a scenario, because the scheduler has more opportunities to find an extension of the plan with respect to fuel demands. In case it is efficient to change the internals of sequential planners, we could incorporate fuel directly into the sequential planner, and only employ scheduling for parallelizing actions in the plan.

Also, suboptimal plans in the sequential domain sometimes yield better scheduled plans than optimal ones. For example, consider a triangular road network with three locations A, B, and C, connected by roads of length 1. There are two packages p_A, p_B and two vehicles v_A, v_B, positioned at A and B respectively. The target location of both packages is C. The capacities and fuel capacities of all vehicles are 2 and both packages have a size of 1. In the sequential variant, the optimal plan consists of either vehicle picking up the package at their location, driving over to the location with the other package, and delivering both packages to C. The cost of such a plan is 6. However, in the temporal variant, both of the vehicles can drive at the same time, meaning that the optimal plan makespan is only 3. If the sequential planner never generates a plan using both vehicles, the temporal scheduler has no way of parallelizing the generated actions, hence it will never schedule a plan with a lower makespan than 6.

We will test the scheduling algorithm with a Randomized Restart planner (Section 4.2.1) and a meta-heuristically weighted forward planner (Section 4.1.4) in Chapter 6.

5.2 Ad-hoc temporal planning

The problem with simply adding fuel to sequential plans generated by planners that have no notion of fuel is that for some problems, the generated plans may never be valid even if vehicles refuel at every opportunity they get.

One possible solution to this issue is to create a fuel-aware ad-hoc planner similar to the ones created in Section 4.2.1. We design the Temporal Randomized Restart Around Path Nearby Planner (TRRAPN) based on this idea, building on top of the RRAPN planner.

Just before choosing between a random vehicle and the nearest vehicle in the original algorithm of RROPN, we uniformly sample a random number from the interval $[0, 1]$ and if it is smaller than some parameter $\Delta \in [0, 1]$, we do a fuel run instead. A *fuel run* is a series of actions in which a vehicle drives to a petrol station. The rest of the algorithm stays the same.

In TRRAPN, the vehicle is chosen at random from a uniform distribution. The petrol station is chosen at random from the inverse distance distribution of the vehicle's location towards the petrol stations, similar to the distribution defined for RRAPD in Section 4.2.1. The Δ parameter is exponentially increased during a planner run. After χ sequential plans are generated, which are all invalid fuel-wise (after adding back fuel and scheduling them), Δ is multiplied by δ and capped at 0.5:

$$\Delta \leftarrow \min(0.5, \delta \cdot \Delta).$$

In all our experiments, the step $\delta = 2$, $\chi = 1000$, and Δ is set to $3 \cdot 10^6$ initially. The temperature parameter T in the inverse distance distribution of petrol stations is set to 0.05. We will also evaluate this planner in Chapter 6.

6. Experimental evaluation

In this chapter, we will describe and run experiments that compare our planners from the last two chapters with domain-independent planners from the IPC. We will briefly discuss the acquired results and interpret them.

6.1 Methodology

Using our benchmarking software (see Attachment 4), we will now run experiments in an environment similar to the original IPCs. The rule pages state that planners have to be single-threaded and use a maximum of 2, 4, or 6 GB of memory (depending on the competition year), with a maximum run time of 30 minutes.

Since the IPC rule pages vary on the exact processor specification or the amount of memory available to each planner between competitions, we adjust the parameters slightly by running our planners for 30 minutes on each problem, using 4 GB of RAM. Our planners will get canceled and prompted for a plan after the time runs out. All our experiments are run on the clusters of MetaCentrum.[1] Due to the nature of computing on MetaCentrum, we were not able to guarantee that all problems and planners run on the exact same processor, only on very similar ones (approximately equivalent to `Intel Xeon E5-2650 v2 2.6 GHz`). The performance of our planners does not change significantly when changing the run time from 30 to 15 minutes or when run on slightly different processors.

Given that the hardware platforms of past IPCs are sufficiently similar to our environment and the fact that our planners perform well even in shorter runs, we do not rerun the planners from the original competitions — instead, we use the scores from IPC results.

We run all our planner implementations in Java using Oracle's OpenJDK version `1.8.0_131-b11`. The results presented here were obtained with the **`0.9.2`** version of the TransportEditor project.[2] The `NOTICE.txt` files in the project module directories specify the exact versions of libraries used. In all planners where nondeterminism occurs, we set the initial random seed to 2017 (on all individual problem runs).

The evaluation criteria remain the same as in the IPC: we focus on plan *quality* in favor of planner run time. The quality of a plan for a specific planner and sequential problem p is defined as:

$$\frac{\text{total-cost}(\text{planner}(p))}{\text{total-cost}(BEST)},$$

where the results called $BEST$ are either precalculated outside of the competition environment or they are the best result of one of the planners in the competition, depending on which plan has a lower total cost. Quality is, therefore, a number between 0 and 1. The overall goal for planners is to maximize the sum of qualities over the problem instances in a given dataset, called the *total quality*.

[1] `https://www.metacentrum.cz/en/`

[2] Git tag `v0.9.2`, available at `https://github.com/oskopek/TransportEditor`

For temporal domains, quality is calculated in the same way, just by substituting total cost for total time. We sometimes refer to total cost and total time as the *score* of the planner, a term that is not dependent on the domain variant.

We will use four datasets for our experiments — the seq-sat-6, seq-sat-7, and seq-sat-8 datasets for sequential and the tempo-sat-6 for temporal planners (Section 3.3). All the datasets used are available in the software project sources (see Attachment 1). Descriptions of planners that we will refer to by their competition names can be found in the respective competition results or booklets for IPC 2008,[3] IPC 2011 (García-Olaya et al., 2011), and IPC 2014 (Vallati et al., 2015). Due to space constraints, we only show the three best non-baseline planners from each competition in the result quality tables and plots, based on total quality. In the temporal dataset tempo-sat-6 quality table, we only show the two best and add another external domain-independent planner to the comparison, the 2014 version of Temporal Fast Downward (TFD2014).

6.2 Sequential Transport

In this section, we present the results of our sequential planners on the seq-sat-6, seq-sat-7, and seq-sat-8 datasets. Specifically, these planners are included in the experiment:

MSFA3 The meta-heuristically weighted SFA planner (Section 4.1.4) with the package and vehicle distance heuristic (Section 4.1.2)

MSFA5 The meta-heuristically weighted SFA planner with the general marking heuristic (Section 4.1.2)

RRAPN The Randomized Restart Around Path Nearby planner (Section 4.2.1)

6.2.1 Results

We show an IPC quality table and a quality plot for the experimental runs on the seq-sat-6 dataset (Figure 6.1), seq-sat-7 dataset (Figure 6.2), and the seq-sat-8 dataset (Figure 6.3). Details about the specific plans along with the benchmark results can be found in Attachment 1.

IPC 2008

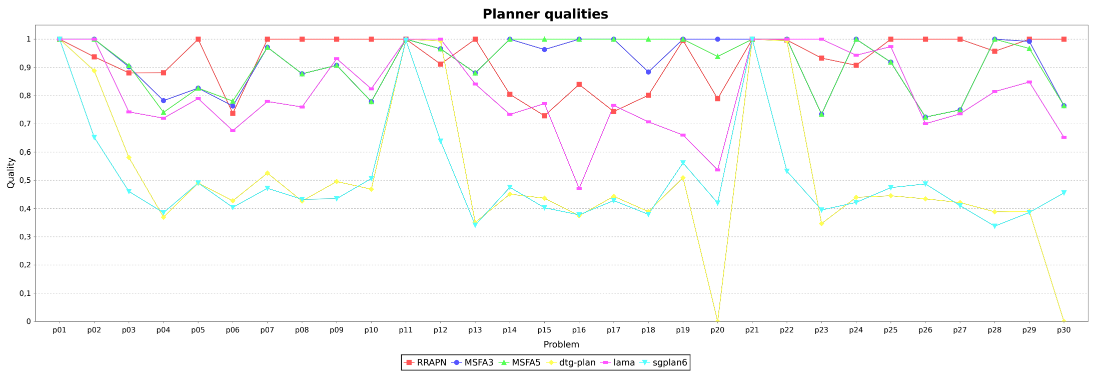
In the updated results of the sequential satisficing track of IPC 2008[4] published after the competition, the overall winner LAMA (a Fast Downward based planner) was hands-down the best planner on the sequential Transport domain, winning with a total quality of 28.93/30, where all other planners had less than 20/30. Only 5 plans generated by LAMA had a worse total cost than the best known plans.

After adding our planners to the results, the total quality of LAMA drops to 24.38/30, because several larger problems were solved better than the best known solution from IPC 2008. Our best planner on the IPC 2008 dataset, RRAPN,

[3]`http://icaps-conference.org/ipc2008/deterministic/Planners.html`
[4]`http://icaps-conference.org/ipc2008/deterministic/Results.html`

#	MSFA3	MSFA5	RRAPN	dtg-plan	lama	sgplan6	BEST
p01	54 **1.00**	54 **1.00**	54 **1.00**	54 **1.00**	54 **1.00**	54 **1.00**	54
p02	270 **1.00**	270 **1.00**	288 **0.94**	304 **0.89**	270 **1.00**	414 **0.65**	270
p03	409 **0.90**	407 **0.91**	419 **0.88**	635 **0.58**	497 **0.74**	801 **0.46**	369
p04	464 **0.78**	490 **0.74**	412 **0.88**	983 **0.37**	504 **0.72**	942 **0.39**	363
p05	704 **0.83**	704 **0.83**	582 **1.00**	1187 **0.49**	737 **0.79**	1186 **0.49**	582
p06	989 **0.76**	967 **0.78**	1024 **0.74**	1766 **0.43**	1117 **0.68**	1868 **0.40**	755
p07	1011 **0.97**	1011 **0.97**	982 **1.00**	1868 **0.53**	1260 **0.78**	2081 **0.47**	982
p08	1053 **0.88**	1053 **0.88**	924 **1.00**	2166 **0.43**	1216 **0.76**	2135 **0.43**	924
p09	1027 **0.91**	1027 **0.91**	932 **1.00**	1880 **0.50**	1001 **0.93**	2143 **0.43**	932
p10	1360 **0.78**	1360 **0.78**	1059 **1.00**	2260 **0.47**	1285 **0.82**	2091 **0.51**	1059
p11	473 **1.00**	473 **1.00**	473 **1.00**	473 **1.00**	473 **1.00**	475 **1.00**	473
p12	823 **0.97**	823 **0.97**	872 **0.91**	800 **0.99**	795 **1.00**	1244 **0.64**	795
p13	1096 **0.88**	1096 **0.88**	965 **1.00**	2751 **0.35**	1147 **0.84**	2827 **0.34**	965
p14	1582 **1.00**	1582 **1.00**	1966 **0.80**	3507 **0.45**	2157 **0.73**	3328 **0.48**	1582
p15	2367 **0.96**	2280 **1.00**	3129 **0.73**	5221 **0.44**	2954 **0.77**	5659 **0.40**	2280
p16	2321 **1.00**	2321 **1.00**	2764 **0.84**	6199 **0.37**	4928 **0.47**	6144 **0.38**	2321
p17	3209 **1.00**	3209 **1.00**	4315 **0.74**	7239 **0.44**	4193 **0.77**	7494 **0.43**	3209
p18	3322 **0.88**	2936 **1.00**	3663 **0.80**	7542 **0.39**	4151 **0.71**	7737 **0.38**	2936
p19	5051 **1.00**	5051 **1.00**	5073 **1.00**	9921 **0.51**	7648 **0.66**	8991 **0.56**	5051
p20	3636 **1.00**	3873 **0.94**	4607 **0.79**	uns.	6773 **0.54**	8663 **0.42**	3636
p21	431 **1.00**	431 **1.00**	431 **1.00**	431 **1.00**	431 **1.00**	431 **1.00**	431
p22	675 **1.00**	675 **1.00**	677 **1.00**	679 **0.99**	675 **1.00**	1268 **0.53**	675
p23	1140 **0.73**	1140 **0.73**	897 **0.93**	2414 **0.35**	837 **1.00**	2119 **0.39**	837
p24	1227 **1.00**	1227 **1.00**	1352 **0.91**	2790 **0.44**	1301 **0.94**	2909 **0.42**	1227
p25	1943 **0.92**	1943 **0.92**	1785 **1.00**	4007 **0.45**	1833 **0.97**	3764 **0.47**	1785
p26	2421 **0.72**	2421 **0.72**	1753 **1.00**	4036 **0.43**	2502 **0.70**	3598 **0.49**	1753
p27	3255 **0.75**	3255 **0.75**	2440 **1.00**	5791 **0.42**	3317 **0.74**	5948 **0.41**	2440
p28	2465 **1.00**	2465 **1.00**	2575 **0.96**	6346 **0.39**	3027 **0.81**	7300 **0.34**	2465
p29	2817 **0.99**	2890 **0.97**	2795 **1.00**	7168 **0.39**	3294 **0.85**	7237 **0.39**	2795
p30	4703 **0.76**	4703 **0.76**	3595 **1.00**	uns.	5513 **0.65**	7892 **0.46**	3595
total	**27.39**	**27.43**	**27.85**	**15.48**	**24.38**	**15.16**	

(a) Quality and score of sequential planners on the seq-sat-6 dataset.

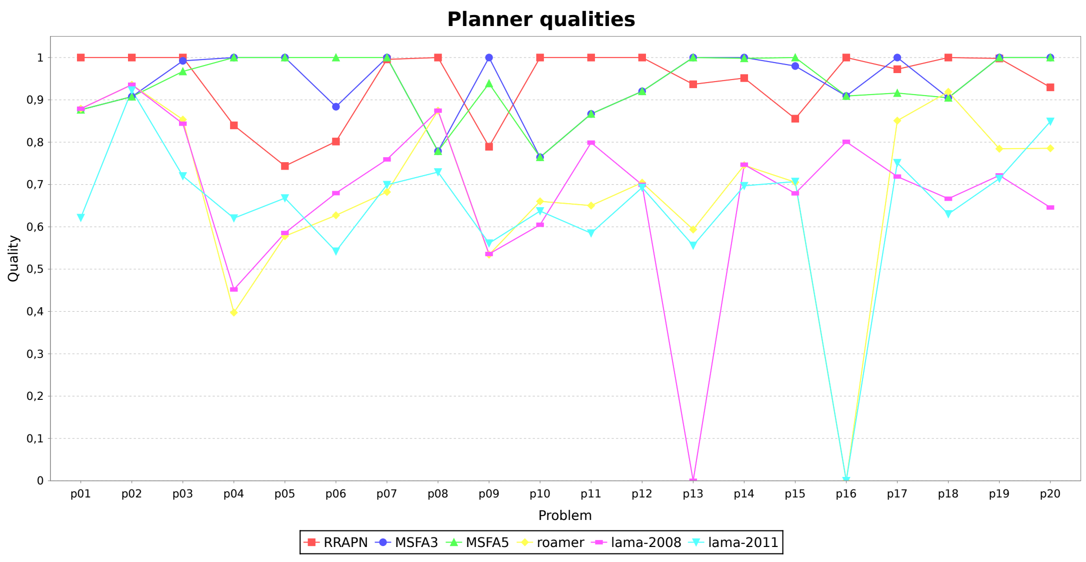


(b) Quality plot of sequential planners on the seq-sat-6 dataset.

Figure 6.1: Planner results on seq-sat-6.

#	MSFA3	MSFA5	RRAPN	lama-2008	lama-2011	roamer	BEST
p01	1053 **0.88**	1053 **0.88**	923 **1.00**	1050 **0.88**	1485 **0.62**	1050 **0.88**	923
p02	1027 **0.91**	1027 **0.91**	932 **1.00**	996 **0.94**	1010 **0.92**	996 **0.94**	932
p03	2817 **0.99**	2890 **0.97**	2795 **1.00**	3313 **0.84**	3882 **0.72**	3275 **0.85**	2795
p04	2321 **1.00**	2321 **1.00**	2764 **0.84**	5135 **0.45**	3741 **0.62**	5841 **0.40**	2321
p05	3209 **1.00**	3209 **1.00**	4315 **0.74**	5481 **0.59**	4805 **0.67**	5553 **0.58**	3209
p06	3322 **0.88**	2936 **1.00**	3663 **0.80**	4320 **0.68**	5415 **0.54**	4681 **0.63**	2936
p07	5051 **1.00**	5051 **1.00**	5073 **1.00**	6652 **0.76**	7222 **0.70**	7403 **0.68**	5051
p08	1360 **0.78**	1360 **0.78**	1059 **1.00**	1211 **0.87**	1452 **0.73**	1211 **0.87**	1059
p09	3636 **1.00**	3873 **0.94**	4607 **0.79**	6786 **0.54**	6479 **0.56**	6806 **0.53**	3636
p10	4703 **0.76**	4703 **0.76**	3595 **1.00**	5943 **0.60**	5641 **0.64**	5445 **0.66**	3595
p11	1426 **0.87**	1426 **0.87**	1236 **1.00**	1547 **0.80**	2113 **0.58**	1901 **0.65**	1236
p12	1466 **0.92**	1466 **0.92**	1349 **1.00**	1929 **0.70**	1947 **0.69**	1915 **0.70**	1349
p13	1630 **1.00**	1630 **1.00**	1740 **0.94**	uns.	2932 **0.56**	2746 **0.59**	1630
p14	5919 **1.00**	5930 **1.00**	6221 **0.95**	7925 **0.75**	8493 **0.70**	7940 **0.75**	5919
p15	4984 **0.98**	4884 **1.00**	5709 **0.86**	7192 **0.68**	6909 **0.71**	6924 **0.71**	4884
p16	6124 **0.91**	6124 **0.91**	5567 **1.00**	6951 **0.80**	uns.	uns.	5567
p17	4432 **1.00**	4838 **0.92**	4558 **0.97**	6166 **0.72**	5899 **0.75**	5209 **0.85**	4432
p18	3963 **0.90**	3963 **0.90**	3586 **1.00**	5381 **0.67**	5690 **0.63**	3902 **0.92**	3586
p19	4124 **1.00**	4124 **1.00**	4133 **1.00**	5716 **0.72**	5777 **0.71**	5257 **0.78**	4124
p20	3765 **1.00**	3765 **1.00**	4050 **0.93**	5831 **0.65**	4435 **0.85**	4793 **0.79**	3765
total	**18.78**	**18.75**	**18.81**	**13.63**	**12.90**	**13.76**	

(a) Quality and score of sequential planners on the seq-sat-7 dataset.

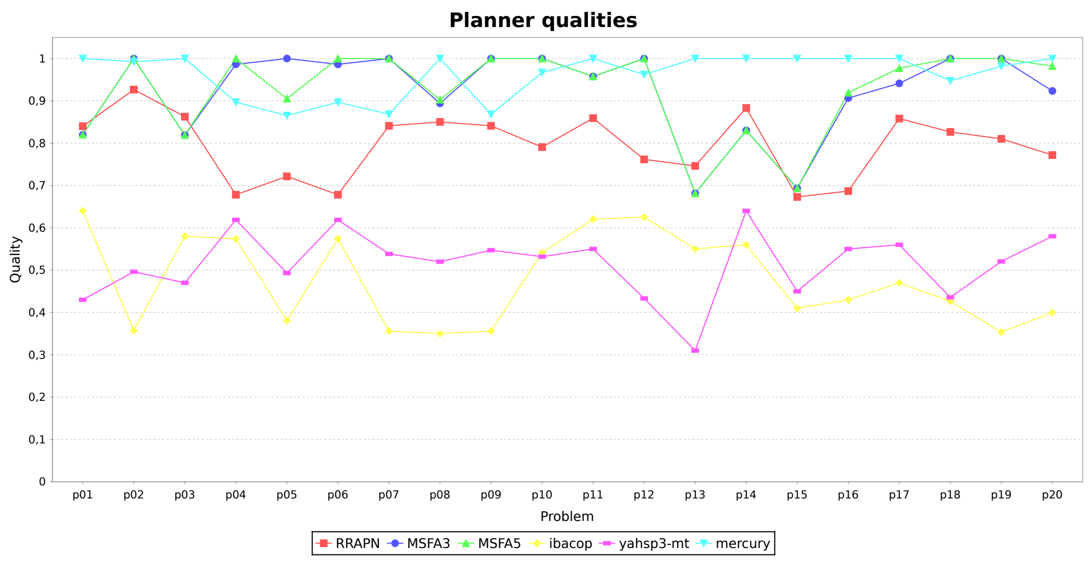


(b) Quality plot of sequential planners on the seq-sat-7 dataset.

Figure 6.2: Planner results on seq-sat-7.

#	MSFA3	MSFA5	RRAPN	ibacop	mercury	yahsp3-mt	BEST
p01	1596 **0.82**	1596 **0.82**	1558 **0.84**	2045 **0.64**	1309 **1.00**	3044 **0.43**	1309
p02	2109 **1.00**	2109 **1.00**	2276 **0.93**	5902 **0.36**	2125 **0.99**	4250 **0.50**	2109
p03	1879 **0.82**	1879 **0.82**	1784 **0.86**	2653 **0.58**	1539 **1.00**	3274 **0.47**	1539
p04	5163 **0.99**	5092 **1.00**	7508 **0.68**	8871 **0.57**	5678 **0.90**	8228 **0.62**	5092
p05	5394 **1.00**	5958 **0.91**	7475 **0.72**	14170 **0.38**	6235 **0.87**	10938 **0.49**	5394
p06	5163 **0.99**	5092 **1.00**	7508 **0.68**	8871 **0.57**	5678 **0.90**	8228 **0.62**	5092
p07	4202 **1.00**	4202 **1.00**	4996 **0.84**	11802 **0.36**	4839 **0.87**	7804 **0.54**	4202
p08	4996 **0.89**	4948 **0.90**	5254 **0.85**	12762 **0.35**	4467 **1.00**	8590 **0.52**	4467
p09	4202 **1.00**	4202 **1.00**	4996 **0.84**	11802 **0.36**	4839 **0.87**	7680 **0.55**	4202
p10	4473 **1.00**	4473 **1.00**	5657 **0.79**	8260 **0.54**	4626 **0.97**	8410 **0.53**	4473
p11	1395 **0.96**	1395 **0.96**	1555 **0.86**	2154 **0.62**	1336 **1.00**	2429 **0.55**	1336
p12	1579 **1.00**	1579 **1.00**	2073 **0.76**	2524 **0.63**	1641 **0.96**	3646 **0.43**	1579
p13	1683 **0.68**	1683 **0.68**	1537 **0.75**	2085 **0.55**	1147 **1.00**	3700 **0.31**	1147
p14	7196 **0.83**	7196 **0.83**	6764 **0.88**	10667 **0.56**	5974 **1.00**	9334 **0.64**	5974
p15	7671 **0.69**	7671 **0.69**	7906 **0.67**	12975 **0.41**	5320 **1.00**	11822 **0.45**	5320
p16	5179 **0.91**	5107 **0.92**	6836 **0.69**	10918 **0.43**	4695 **1.00**	8536 **0.55**	4695
p17	4823 **0.94**	4646 **0.98**	5290 **0.86**	9659 **0.47**	4540 **1.00**	8107 **0.56**	4540
p18	4585 **1.00**	4585 **1.00**	5547 **0.83**	10755 **0.43**	4840 **0.95**	10521 **0.44**	4585
p19	3812 **1.00**	3812 **1.00**	4705 **0.81**	10780 **0.35**	3881 **0.98**	7322 **0.52**	3812
p20	4173 **0.92**	3923 **0.98**	4991 **0.77**	9632 **0.40**	3853 **1.00**	6643 **0.58**	3853
total	**18.44**	**18.49**	**15.91**	**9.56**	**19.25**	**10.29**	

(a) Quality and score of sequential planners on the seq-sat-8 dataset.



(b) Quality plot of sequential planners on the seq-sat-8 dataset.

Figure 6.3: Planner results on seq-sat-8.

achieves a total quality of 27.85/30, which is a slight improvement over LAMA and other planners. The biggest gain of RRAPN is in being able to calculate solutions of larger problems fast, which can be observed on the results on problems 7–10 or 25–27, which are some of the largest problems. On the other hand, RRAPN fails to achieve optimal scores on some smaller problems like problem 2 or 12, due to its explicit nature.

MSFA3 and MSFA5 are quite similar both in their construction and results on this dataset. They generally obtain better results than RRAPN on smaller problems (problems 2–3, 12, 22), but they can generate very good results even on larger problems, like 14–20 or 28–29. The reason why RRAPN occasionally obtains better plans than the admissible heuristic of MSFA5 is that we weight it with weights greater than or equal to 1, therefore, making the heuristic inadmissible. Based on total quality, MSFA5 marginally comes out on top as the better one of the two MSFA planners on this dataset. All three of our planners beat all planners from the original competition based on total quality.

IPC 2011

The 2011 competition featured 20 sequential Transport problems, with 4 planners (dae_yahsp, LAMA 2008 and 2011, and roamer) achieving a total quality of more than 15/20. Interestingly, LAMA 2008 was able to produce better plans than its 2011 version in 12 out of 20 problems. The overall winner on Transport in 2011, roamer, achieved comparable scores on most problems to both versions of LAMA.

RRAPN consistently achieves better scores than all domain-independent planners from the original competition in all the 20 problems. This can again be attributed to the size of the problems (see Table 3.2).

Even though RRAPN is better than the original planners more often than both MSFA planners, MSFA3 comes out on top based on total cost. Nonetheless, the differences in performance between all three of our planners are almost indistinguishable, even on most problem instances individually. Even more interesting, the problems where MSFA planners perform well are complementary to the ones where RRAPN performs well, as is visible on the results of problems 4–6, 10–12, and 13–15.

IPC 2014

In the sequential satisficing track of IPC 2014, the winner on the Transport domain was without a doubt the Mercury planner, achieving a stunning 20/20 total quality. Even more interesting is the fact that the runner-up yahsp3-mt achieved a score of only 10.74/20 and all other planners achieved sub 10/20 total quality, accentuating the performance of Mercury even more.

After adding the results of our planners to the quality table, the total quality of yahsp3-mt is lowered to 10.29/20. Mercury loses its spotless result, but still significantly dominates all other planners, including ours, at 19.25/20.

RRAPN manages to outperform yahsp3-mt with 15.91/20, yet it fails to match the results of Mercury, not even in one problem (it does come close in problems 7 and 9). Both MSFA planners outperform RRAPN on this dataset with qualities just under 18.50/20, but still do not come reasonably close to beating Mercury.

However, they do (marginally) outperform Mercury on some problems, like problems 4–7, 9–10, 12, and 18–19. The results of MSFA3 and MSFA5 on this dataset are almost identical.

6.3 Temporal Transport

In this section, we present the results of our temporal planners on the tempo-sat-6 dataset. The following planners are included in the experiment:

MSFA5Sched The scheduled MSFA5 planner (Sections 4.1.4 and 4.1.2)

RRAPNSched The scheduled (Section 5.1) RRAPN planner (Section 4.2.1)

TFD2014 The Temporal Fast Downward planner, version 0.4 from IPC 2014 (Vallati et al., 2015, Preferring Preferred Operators in Temporal Fast Downward)

TRRAPN The Temporal RRAPN planner (Section 5.2)

An external temporal planner, TFD2014, was added to the experiments to be able to compare our planners to more recent systems. TFD2014 is a newer version of the original TFD planner that took part in IPC 2008.

6.3.1 Results

We show an IPC quality table and a quality plot of an experimental run of these planners on the tempo-sat-6 dataset (Figure 6.4). Additionally, sample Gantt charts (Gantt, 1910) of two chosen plans are shown in Figure 6.5. The generated plans and benchmark results can be found in Attachment 1.

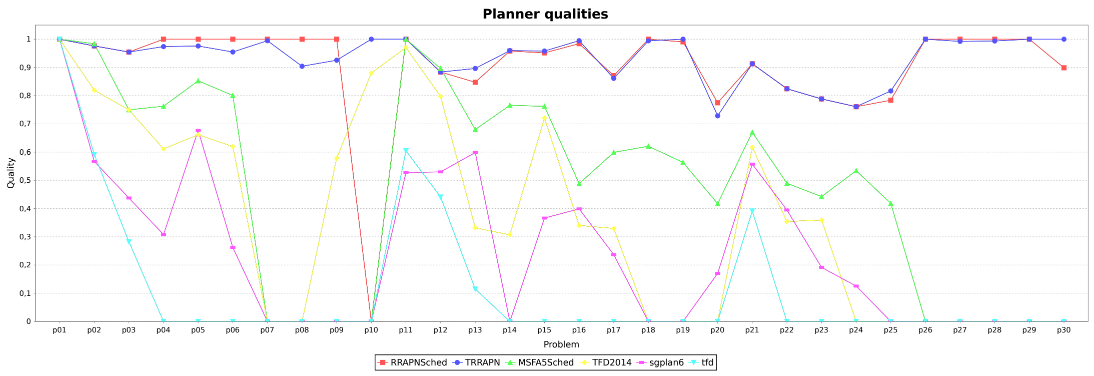
Planners that entered the 2008 temporal satisficing track at the IPC did not cope well with the Transport domain — only two non-baseline planners (SGPlan$_6$ and TFD) were able to produce at least one plan for any problem. Additionally, only the smallest problem, problem 1, was solved to the best known score by any planner. The best total quality was only 7.5/30, achieved by SGPlan$_6$. No other domain in the temporal track had a lower best total quality than Transport, which, assuming reasonably generated problem instances, hints at Transport being one of the harder domains for domain-independent temporal planners. We observe an evident performance increase of Temporal Fast Downward, when comparing the qualities of plans of TFD (from 2008) and TFD2014.

Our results further show that using a simple domain-dependent scheduling approach yields an improvement over domain-independent temporal planners that took part in IPC 2008. Our scheduling planners RRAPNSched and MSFA5Sched achieve total qualities of 27.16/30 and 14.50/30, respectively. The scheduled MSFA5 planner does not generate plans of such variety as RRAPN, and therefore produces worse results when scheduled using our algorithm, mostly due to the inability of the scheduler to add enough `refuel` actions to the existing plans to make them feasible.

RRAPNSched, on the other hand, is able to beat even newer temporal planners like TFD2014 by a significant margin. We see that RRAPNSched produces plans with worse scores than the best known score on smaller problems, which

#	MSFA5Sched	RRAPNSched	TFD2014	TRRAPN	sgplan6	tfd	BEST
p01	52 **1.00**	52 **1.00**	52.02 **1.00**	52 **1.00**	52 **1.00**	52 **1.00**	52
p02	125.01 **0.98**	126.01 **0.98**	150.11 **0.82**	126.01 **0.98**	217 **0.57**	208 **0.59**	123
p03	252.02 **0.75**	198.01 **0.95**	252.14 **0.75**	198.01 **0.95**	432 **0.44**	669 **0.28**	189
p04	341.02 **0.76**	260.02 **1.00**	425.29 **0.61**	267.01 **0.97**	845 **0.31**	uns.	260.02
p05	285.03 **0.85**	243.02 **1.00**	367.32 **0.66**	249.02 **0.98**	359 **0.68**	uns.	243.02
p06	316.02 **0.80**	253.01 **1.00**	408.31 **0.62**	265.02 **0.95**	965 **0.26**	uns.	253.01
p07	uns.	367.03 **1.00**	uns.	369.03 **0.99**	uns.	uns.	367.03
p08	uns.	481.04 **1.00**	uns.	532.04 **0.90**	uns.	uns.	481.04
p09	uns.	286.03 **1.00**	494.44 **0.58**	309.03 **0.93**	uns.	uns.	286.03
p10	uns.	uns.	939.8 **0.88**	827.07 **1.00**	uns.	uns.	827.07
p11	332.01 **1.00**	332.01 **1.00**	342.09 **0.97**	332.01 **1.00**	629 **0.53**	549 **0.60**	332
p12	483.01 **0.90**	490.01 **0.88**	543.13 **0.80**	490.01 **0.88**	817 **0.53**	982 **0.44**	433
p13	572.02 **0.68**	459.01 **0.85**	1172.38 **0.33**	434.01 **0.90**	650 **0.60**	3383 **0.11**	389
p14	777.03 **0.77**	621.02 **0.96**	1938.75 **0.31**	620.02 **0.96**	uns.	uns.	595
p15	1081.04 **0.76**	866.04 **0.95**	1143.45 **0.72**	860.04 **0.96**	2249 **0.37**	uns.	824
p16	1532.07 **0.49**	760.03 **0.98**	2198.97 **0.34**	752.03 **0.99**	1875 **0.40**	uns.	748
p17	1317.07 **0.60**	906.03 **0.87**	2393.97 **0.33**	916.04 **0.86**	3331 **0.24**	uns.	789
p18	1960.09 **0.62**	1217.05 **1.00**	uns.	1224.06 **0.99**	uns.	uns.	1217.05
p19	2226.12 **0.56**	1266.06 **0.99**	uns.	1254.06 **1.00**	uns.	uns.	1254.06
p20	2596.13 **0.42**	1399.07 **0.77**	uns.	1488.08 **0.73**	6362 **0.17**	uns.	1084
p21	94.02 **0.67**	69.01 **0.91**	102.14 **0.62**	69.01 **0.91**	113 **0.56**	161 **0.39**	63
p22	192.03 **0.49**	114.01 **0.82**	265.38 **0.35**	114.01 **0.82**	238 **0.39**	uns.	94
p23	278.04 **0.44**	156.02 **0.79**	342.44 **0.36**	156.02 **0.79**	642 **0.19**	uns.	123
p24	262.04 **0.53**	184.02 **0.76**	uns.	184.02 **0.76**	1116 **0.13**	uns.	140
p25	373.05 **0.42**	199.02 **0.78**	uns.	191.02 **0.82**	uns.	uns.	156
p26	uns.	234.03 **1.00**	uns.	234.02 **1.00**	uns.	uns.	234.02
p27	uns.	254.03 **1.00**	uns.	256.03 **0.99**	uns.	uns.	254.03
p28	uns.	312.03 **1.00**	uns.	314.03 **0.99**	uns.	uns.	312.03
p29	uns.	314.03 **1.00**	uns.	314.03 **1.00**	uns.	uns.	314.03
p30	uns.	385.04 **0.90**	uns.	346.03 **1.00**	uns.	uns.	346.03
total	**14.50**	**27.16**	**11.05**	**28.02**	**7.35**	**3.43**	

(a) Quality and score of temporal planners on the tempo-sat-6 dataset.



(b) Quality plot of temporal planners on the tempo-sat-6 dataset.

Figure 6.4: Planner results on tempo-sat-6.

is mainly due to the fact that plans for smaller problems are easier to precalculate and hence the best known score estimate is closer to the optimum than the estimates for larger problems.

An interesting case is problem 10, which was not solved by any of our planners based purely on scheduling. This problem contains a single petrol station at location `city-loc-1`, which, unfortunately, is not present in many precomputed shortest paths and the vehicles, therefore, do not drive through it often. Furthermore, even fewer of those paths coincide with ones on which the planners deliver packages. This results in most generated plans being infeasible fuel-wise.

One way to solve the issue is to use TRRAPN planner's approach, which knows about fuel during planning and sometimes adds fuel runs to the plan, during which a chosen vehicle drives to a petrol station to refuel. TRRAPN achieves the best total quality from all our planners, beating RRAPNSched by about 0.8, even though it is marginally worse on some problems (for example, problems 4–7). TRRAPN makes up for this by getting better scores on larger problems, where it takes RRAPNSched more effort to plan with fuel feasibly, as can be observed on problems 10, 19, 25, and 30.

In Figure 6.5, we see a comparison of the plans of RRAPNSched and TFD2014 for problem 12. Observe that the important difference between the two plans is that `truck-1` chooses to pick up and move `package-4` while it is delivering `package-2`. It then later picks it up again and delivers it while `truck-2` is delivering `package-3`. In the plan of TFD2014, `truck-1` did not pick up `package-4` while delivering `package-2` and then had to travel further to deliver it — this is basically the only difference between the two plans, and it makes a difference in makespan of more than 50. Finally, observe that due to the nature of RRAPN, `truck-1` dropped `package-4` and then went back to pick it up, even though it had enough capacity to carry it for the whole time. Nonetheless, it is important to note that this had no effect on the total makespan of the plan.

6.4 Overall results

The attained results show that domain-specific information can be leveraged to generate plans of better quality. We have designed and implemented Transport planners that are able to beat all results from the sequential and temporal satisficing tracks of the 2008 and 2011 IPCs. In the 2014 IPC, we would have attained second place on overall quality in the Transport domain, behind the impressive result of Mercury. Our planners achieve satisfactory results across datasets, as can be observed in Table 6.1.

Another major advantage previously unmentioned is that our planners generate good solutions quite fast. To show this, we present results from running all our planners on all problems for 3 seconds each (Table 6.2). The achieved scores are very close to the scores from long planning runs, except for problems where the planner did not find a plan at all in the given time limit.

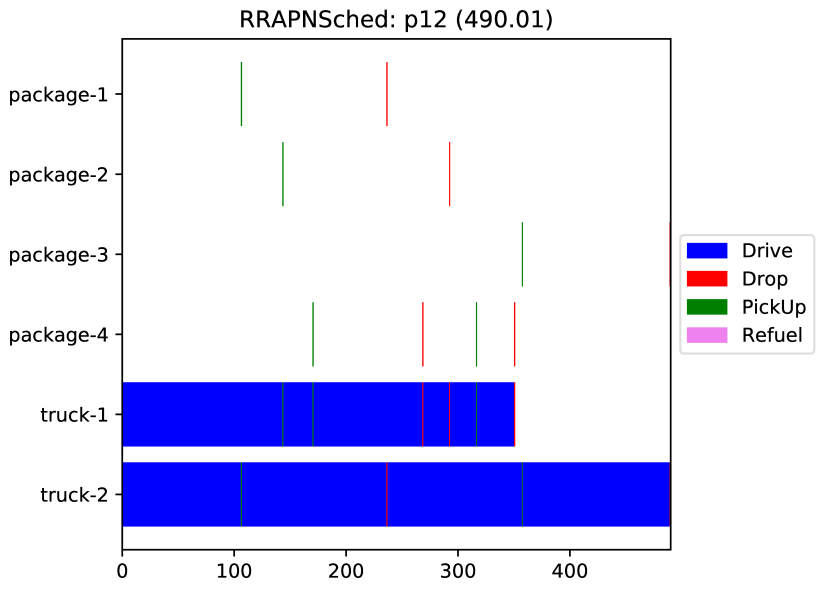
Planner	**Average quality**
MSFA3	0.923
MSFA5	**0.924**
RRAPN	0.894

(a) Avg. quality on sequential datasets.

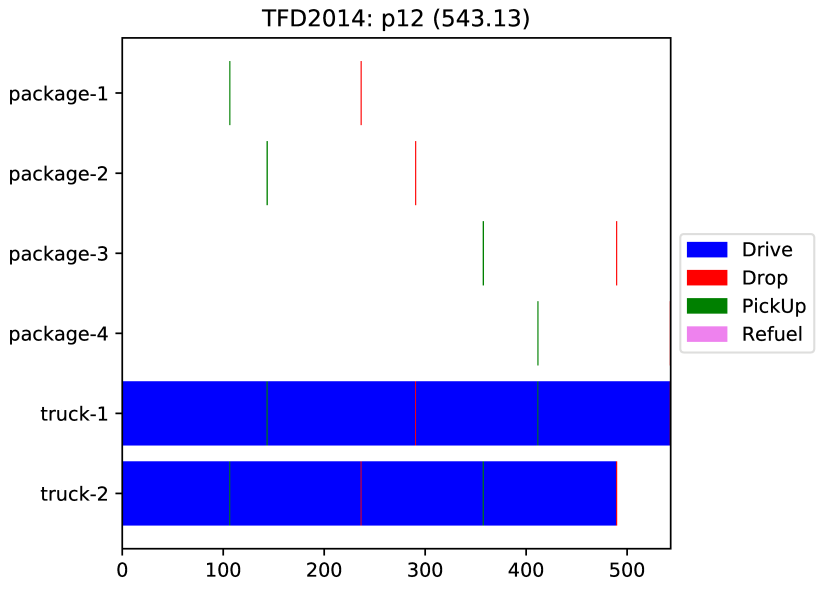
Planner	**Average quality**
MSFA5Sched	0.483
RRAPNSched	0.905
TRRAPN	**0.934**

(b) Avg. quality on the temporal dataset.

Table 6.1: Average quality of our planners across datasets.



(a) RRAPNSched



(b) TFD2014

Figure 6.5: Gantt charts of the RRAPNSched and TFD2014 planners on the tempo-sat-6 `p12` problem.

#	MSFA3	MSFA5	RRAPN	BEST
p01	54 **1.00**	54 **1.00**	54 **1.00**	54
p02	270 **1.00**	270 **1.00**	288 **0.94**	270
p03	419 **0.85**	408 **0.87**	419 **0.85**	355
p04	464 **0.78**	490 **0.74**	412 **0.88**	363
p05	732 **0.80**	732 **0.80**	609 **0.96**	582
p06	989 **0.76**	967 **0.78**	1091 **0.69**	755
p07	1011 **0.98**	1011 **0.98**	1060 **0.93**	988
p08	1053 **0.88**	1053 **0.88**	1004 **0.92**	925
p09	1027 **0.93**	1027 **0.93**	1048 **0.91**	955
p10	1360 **0.78**	1360 **0.78**	1164 **0.91**	1059
p11	473 **1.00**	473 **1.00**	473 **1.00**	473
p12	823 **0.97**	823 **0.97**	872 **0.91**	795
p13	1096 **0.88**	1096 **0.88**	965 **1.00**	965
p14	1582 **1.00**	1582 **1.00**	1966 **0.80**	1582
p15	2367 **0.96**	2280 **1.00**	3147 **0.72**	2280
p16	2321 **1.00**	2321 **1.00**	2832 **0.82**	2321
p17	3209 **1.00**	3209 **1.00**	4413 **0.73**	3209
p18	3406 **0.86**	2936 **1.00**	3754 **0.78**	2936
p19	5051 **1.00**	5051 **1.00**	5187 **0.97**	5051
p20	4189 **0.87**	4189 **0.87**	5095 **0.71**	3636
p21	431 **1.00**	431 **1.00**	431 **1.00**	431
p22	675 **1.00**	710 **0.95**	677 **1.00**	675
p23	1140 **0.73**	1140 **0.73**	897 **0.93**	837
p24	1227 **1.00**	1227 **1.00**	1423 **0.86**	1227
p25	1943 **0.92**	1943 **0.92**	1881 **0.95**	1785
p26	2421 **0.74**	2421 **0.74**	1855 **0.97**	1797
p27	3255 **0.77**	3255 **0.77**	2634 **0.96**	2521
p28	2465 **1.00**	2465 **1.00**	2807 **0.88**	2465
p29	2817 **1.00**	2890 **0.97**	3238 **0.87**	2817
p30	4703 **0.76**	4703 **0.76**	4278 **0.84**	3595
total	**27.23**	**27.32**	**26.70**	

(a) Quality and score of our planners on the seq-sat-6 dataset.

#	MSFA5S.	RRAPNS.	TRRAPN	BEST
p01	52 **1.00**	52 **1.00**	52 **1.00**	52
p02	150 **0.82**	126 **0.98**	126 **0.98**	123
p03	uns.	198 **0.95**	198 **0.95**	189
p04	341 **0.76**	267 **0.97**	267 **0.97**	260
p05	285 **0.85**	277 **0.88**	275 **0.88**	243
p06	316 **0.80**	284 **0.89**	288 **0.88**	253
p07	uns.	uns.	uns.	367
p08	uns.	uns.	uns.	532
p09	uns.	464 **0.62**	436 **0.66**	286
p10	uns.	uns.	uns.	827
p11	332 **1.00**	332 **1.00**	332 **1.00**	332
p12	483 **0.90**	490 **0.88**	490 **0.88**	433
p13	572 **0.68**	486 **0.80**	484 **0.80**	389
p14	777 **0.77**	654 **0.91**	669 **0.89**	595
p15	1081 **0.76**	939 **0.88**	914 **0.90**	824
p16	1532 **0.49**	1088 **0.69**	1061 **0.70**	748
p17	1495 **0.53**	1170 **0.67**	1323 **0.60**	789
p18	1960 **0.62**	1822 **0.67**	1725 **0.71**	1217
p19	2226 **0.56**	1609 **0.78**	1692 **0.74**	1254
p20	2596 **0.42**	4688 **0.23**	2969 **0.37**	1084
p21	102 **0.62**	94 **0.67**	69 **0.91**	63
p22	192 **0.49**	114 **0.82**	114 **0.82**	94
p23	278 **0.44**	202 **0.61**	194 **0.63**	123
p24	350 **0.40**	202 **0.69**	184 **0.76**	140
p25	uns.	201 **0.78**	224 **0.70**	156
p26	uns.	286 **0.83**	288 **0.82**	236
p27	uns.	355 **0.72**	298 **0.85**	254
p28	uns.	340 **0.92**	468 **0.67**	312
p29	uns.	507 **0.62**	372 **0.84**	314
p30	uns.	692 **0.50**	517 **0.67**	348
total	**12.91**	**20.95**	**21.60**	

(b) Quality and rounded score of our planners on the tempo-sat-6 dataset.

#	MSFA3	MSFA5	RRAPN	BEST
p01	1053 **0.88**	1053 **0.88**	1018 **0.91**	925
p02	1027 **0.93**	1027 **0.93**	1081 **0.88**	955
p03	2817 **1.00**	2890 **0.97**	3267 **0.86**	2817
p04	2321 **1.00**	2321 **1.00**	2899 **0.80**	2321
p05	3209 **1.00**	3209 **1.00**	4434 **0.72**	3209
p06	3406 **0.86**	2936 **1.00**	3754 **0.78**	2936
p07	5051 **1.00**	5051 **1.00**	5187 **0.97**	5051
p08	1360 **0.78**	1360 **0.78**	1164 **0.91**	1059
p09	3636 **1.00**	4189 **0.87**	5095 **0.71**	3636
p10	4703 **0.76**	4703 **0.76**	4262 **0.84**	3595
p11	1426 **0.87**	1426 **0.87**	1341 **0.92**	1238
p12	1466 **0.92**	1466 **0.92**	1544 **0.87**	1349
p13	1630 **1.00**	1630 **1.00**	1982 **0.82**	1630
p14	uns.	uns.	6745 **0.88**	5919
p15	uns.	uns.	6175 **0.79**	4884
p16	uns.	uns.	6259 **0.89**	5567
p17	4838 **0.92**	4838 **0.92**	5440 **0.81**	4432
p18	3963 **0.92**	uns.	4145 **0.88**	3641
p19	4124 **1.00**	4124 **1.00**	4620 **0.89**	4124
p20	uns.	uns.	4662 **0.81**	3765
total	**14.84**	**13.90**	**16.97**	

(c) Quality and score of our planners on the seq-sat-7 dataset.

#	MSFA3	MSFA5	RRAPN	BEST
p01	1596 **0.82**	1596 **0.82**	1828 **0.72**	1309
p02	2109 **1.00**	2109 **1.00**	2543 **0.83**	2109
p03	1879 **0.82**	1879 **0.82**	1898 **0.81**	1539
p04	uns.	uns.	8065 **0.63**	5092
p05	uns.	uns.	7730 **0.70**	5394
p06	uns.	uns.	7978 **0.64**	5092
p07	uns.	uns.	5807 **0.72**	4202
p08	uns.	uns.	5848 **0.76**	4467
p09	uns.	4202 **1.00**	5839 **0.72**	4202
p10	uns.	uns.	6304 **0.71**	4473
p11	1395 **0.96**	1395 **0.96**	1806 **0.74**	1336
p12	1579 **1.00**	1579 **1.00**	2263 **0.70**	1579
p13	1683 **0.68**	1683 **0.68**	1698 **0.68**	1147
p14	uns.	uns.	7785 **0.77**	5974
p15	uns.	uns.	8136 **0.65**	5320
p16	uns.	5179 **0.91**	7249 **0.65**	4695
p17	uns.	uns.	5789 **0.78**	4540
p18	uns.	4585 **1.00**	6160 **0.74**	4585
p19	3812 **1.00**	3812 **1.00**	5364 **0.71**	3812
p20	4173 **0.92**	4173 **0.92**	5544 **0.69**	3853
total	**7.20**	**10.11**	**14.36**	

(d) Quality and score of our planners on the seq-sat-8 dataset.

Table 6.2: Results of our planners when run for 3 seconds on all datasets.

Conclusion

Domain-specific planning has historically been neglected as not general enough and, therefore, theoretically unimportant. Our work shows that there are more than enough problems left to solve when planning with prior domain knowledge, both theoretically and in practical approaches.

To support this statement, we have discussed the theoretical challenges underlying planning in general and how the challenges change when domain knowledge is added. We have analyzed variants of the Transport domain from the International Planning Competition and designed various planners specific to the domain. Using results of the experimental evaluation of several discussed approaches on datasets of the Transport domain, we have shown the ability to achieve comparable results to state-of-the-art domain-independent planners. The performance of domain-independent planners is generally very impressive, given the difficulty of the problem they are solving. Despite the broad misconception that they are not useful in practice, we have not managed to beat all of them even when leveraging domain-specific knowledge acquired by our analysis.

There remain more promising approaches to apply to planning for the Transport domain and to domain-specific planning in general. We list a few, in our opinion, perspective methods, which were not evaluated in this work:

- Hierarchical Task Networks: HTN planning uses *tasks*, a higher level and usually domain-specific description of sequences of operators to carry out some goal (Ghallab et al., 2004, Chapter 11). An HTN planner decomposes these tasks and embeds them in a classical plan. This approach has been thoroughly studied and is arguably one of the most used in practice today. The ad-hoc planners we designed are conceptually similar to HTN planning.

- Pointer Networks and Reinforcement Learning: A recent attempt at training special architectures of neural networks to solve TSP problem instances using reinforcement learning by Bello et al. (2016) shows reasonable promise for the future. While this technique is highly experimental at the moment, neural networks have successfully helped in pushing the limits of other fields before.

- Learning a domain-specific heuristic function: Another neural network approach aims to help solve the problem of coming up with a good heuristic for a domain. Chen and Wei (2011) train a neural network to use as a heuristic for state space search, which may help when creating a heuristic is simply too challenging or time-consuming. A similar approach is also used in DeepStack (Moravčík et al., 2017), which recently succeeded in beating human players in poker (a good example of a problem with a very large search space).

To make the analysis and planner design easier, we have developed TransportEditor (see Attachment 3), an intuitive graphical desktop application for transportation planning. TransportEditor was recently accepted to the System Demonstrations and Exhibits track at the 27th International Conference on Automated Planning and Scheduling (Škopek and Barták, 2017).

Bibliography

David L. Applegate, Robert E. Bixby, William J. Cook, and Vašek Chvátal. On the Solution of Travelling Salesman Problems. *Documenta Mathematica, Journal der Deutschen Mathematiker Vereinigung*, pages 645–656, 1998. Extra volume, ICM Berlin 1998.

David L. Applegate, Robert E. Bixby, Vašek Chvátal, and William J. Cook. *The Traveling Salesman Problem: A Computational Study.* Princeton University Press, September 2011. ISBN 978-0691129938.

Roman Barták and Jindřich Vodrážka. Domain Modeling for Planning as Logic Programming. In *Proceedings of the 29th International Florida Artificial Intelligence Research Society Conference*, 2016.

Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural Combinatorial Optimization with Reinforcement Learning. ArXiv preprint, 2016. URL `http://arxiv.org/abs/1611.09940`.

Avrim L. Blum and Merrick L. Furst. Fast Planning Through Planning Graph Analysis. *Artificial Intelligence*, 90(1-2):281–300, February 1997. doi: 10.1016/s0004-3702(96)00047-1.

Kris Braekers, Katrien Ramaekers, and Inneke Van Nieuwenhuyse. The Vehicle Routing Problem: State of the Art Classification and Review. *Computers & Industrial Engineering*, 99:300–313, 2016. doi: 10.1016/j.cie.2015.12.007.

Tim Bray. The JavaScript Object Notation (JSON) Data Interchange Format. RFC 7159, Internet Engineering Task Force, March 2014. URL `https://tools.ietf.org/rfc/rfc7159.txt`.

Hung-Che Chen and Jyh-Da Wei. Using Neural Networks for Evaluation in Heuristic Search Algorithm. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence*, 2011.

George B. Dantzig and John H. Ramser. The Truck Dispatching Problem. *Management Science*, 6(1):80–91, 1959.

Geoffrey De Smet et al. *OptaPlanner User Guide.* Red Hat and the community, 7.0.0.CR1 edition, April 2017. URL `https://www.optaplanner.org`. OptaPlanner is an open source constraint satisfaction solver in Java.

Burak Eksioglu, Arif V. Vural, and Arnold Reisman. The Vehicle Routing Problem: A Taxonomic Review. *Computers & Industrial Engineering*, 57(4):1472–1483, November 2009. doi: 10.1016/j.cie.2009.05.009.

Richard E. Fikes and Nils J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2(3-4):189–208, 1971.

Maria Fox and Derek Long. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, 20: 61–124, 2003. doi: 10.1613/jair.1129.

Henry L. Gantt. Work, Wages, and Profits: Their Influence on the Cost of Living. *Engineering Magazine*, 1910.

Ángel García-Olaya, Sergio Jiménez, and Carlos Linares López. The 2011 International Planning Competition: Description of Participating Planners of the Deterministic Track, June 2011.

Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated Planning: Theory and Practice.* Elsevier/Morgan Kaufmann, Burlington, 2004. ISBN 1558608567.

Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, July 1968. ISSN 0536-1567. doi: 10.1109/TSSC.1968.300136.

Malte Helmert. On the Complexity of Planning in Transportation Domains. In *Proceedings of the 6th European Conference on Planning*, pages 349–360, 2001a.

Malte Helmert. On the Complexity of Planning in Transportation and Manipulation Domains. Master's thesis, Albert-Ludwigs-Universität Freiburg, 2001b.

Richard Howey and Derek Long. VAL's Progress: The Automatic Validation Tool for PDDL2.1 used in the International Planning Competition. In *Proceedings of the ICAPS Workshop on The Competition: Impact, Organization, Evaluation, Benchmarks*, pages 28–37, 2003.

Arthur B. Kahn. Topological Sorting of Large Networks. *Communications of the ACM*, 5(11):558–562, November 1962. ISSN 0001-0782. doi: 10.1145/368996.369025.

Joseph B. Kruskal. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. In *Proceedings of the American Mathematical Society*, volume 7, pages 48–48. American Mathematical Society (AMS), January 1956. doi: 10.1090/S0002-9939-1956-0078686-7.

Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL – The Planning Domain Definition Language. Technical report, Yale Center for Computational Vision and Control, October 1998.

Jairo R. Montoya-Torres, Julián López Franco, Santiago Nieto Isaza, Heriberto Felizzola Jiménez, and Nilson Herazo-Padilla. A Literature Review on the Vehicle Routing Problem with Multiple Depots. *Computers & Industrial Engineering*, 79:115–129, 2015. doi: 10.1016/j.cie.2014.10.029.

Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisý, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling.

DeepStack: Expert-Level Artificial Intelligence in Heads-Up No-Limit Poker. *Science*, 2017. ISSN 0036-8075. doi: 10.1126/science.aam6960.

Dana S. Nau. Current Trends in Automated Planning. *AI Magazine*, 28(4):43, 2007. ISSN 0738-4602.

NEO Research Group. Vehicle Routing Problem, 2013. URL `http://neo.lcc.uma.es/vrp/`. Accessed: 26 January 2017.

Ira Pohl. Heuristic Search Viewed as Path Finding in a Graph. *Artificial Intelligence*, 1(3-4):193–204, 1970.

Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995. ISBN 9780131038059.

Ondrej Škopek and Roman Barták. TransportEditor – Creating and Visualising Transportation Problems and Plans. Accepted to the System Demonstrations and Exhibits track of ICAPS 2017, 2017.

Mauro Vallati, Lukáš Chrpa, Marek Grześ, Thomas L. McCluskey, Mark Roberts, Scott Sanner, et al. The 2014 International Planning Competition: Progress and Trends, 2015.

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
BFS	Breadth-First Search
CVRP	Capacitated Vehicle Routing Problem
DAG	Directed Acyclic Graph
DFS	Depth-First Search
GCD	Greatest Common Divisor
ICAPS	International Conference on Automated Planning and Scheduling
IPC	International Planning Competition
MDVRP	Multiple Depot Vehicle Routing Problem
MSFA	Meta-heuristically Weighted Sequential Forward A*
MST	Minimum Spanning Tree
PDDL	Planning Domain Definition Language
RBAPN	Randomized Backtrack Around Path Nearby Planner
RRAPD	Randomized Restart Around Path Distribution Planner
RRAPN	Randomized Restart Around Path Nearby Planner
RRFL	Randomized Restart From Location Planner
RROP	Randomized Restart On Path Planner
RROPN	Randomized Restart On Path Nearby Planner
SFA	Sequential Forward A*
STRIPS	Stanford Research Institute Problem Solver
TFD	Temporal Fast Downward
TRRAPN	Temporal Randomized Restart Around Path Nearby Planner
TSP	Traveling Salesman Problem
TQE	Temporally Qualified Expression
VRP	Vehicle Routing Problem
WSFA	Weighted Sequential Forward A*

Attachments

1. CD contents

The compact disc attached to this thesis contains the following:

- `thesis.pdf`, the PDF version of this thesis;
- `TransportEditor-0.9.2/`, the `0.9.2` release of the TransportEditor software, also obtainable at (Git tag `v0.9.2`):
  `https://github.com/oskopek/TransportEditor/releases`. Contains:
  - `bin/`, built executable JAR files (with dependencies included);
  - `datasets/`, the datasets used at IPC;
  - `docs/`, the user, developer, and JavaDoc documentation and a specification document for TransportEditor;
  - `sources/`, sources of TransportEditor along with source of planners, benchmarks and the report generator; and
  - `tools/`, the benchmarker execution scripts and configuration files, along with other tools used. The `tools/benchmarks/results-long/` directory contains the benchmark results from Sections 6.2.1 and 6.3.1.

To compile and run executables of this project, you need Java 1.8 update 40+ and Maven 3+. To run:

- TransportEditor, execute:

  ```
  java -jar bin/TransportEditor-0.9.2-jar-with-dependencies.jar
  ```

  from the `TransportEditor-0.9.2/` directory; or
- the experiments from this work, execute:

  ```
  ./benchmark.sh configs/<config_name>.json
  ```

  from the `TransportEditor-0.9.2/tools/benchmarks/` directory.

The VAL validator (needed for validation of plans in experiments) can be obtained at: `https://github.com/KCL-Planning/VAL`.

TransportEditor User & Developer Manuals

A user manual explaining use-cases, the user interface, saving and loading files, together with a developer manual explaining the architecture, technical choices, and program flow is also available on the attached CD, in the directory:

`TransportEditor-0.9.2/docs/manuals/`.

Generated API documentation using JavaDoc is also available on the attached CD, in the directory:

`TransportEditor-0.9.2/docs/javadoc/`.

2. PDDL representations of Transport

```
(:action drive
  :parameters (?v - vehicle ?l1 ?l2 - location)
  :precondition (and
      (at ?v ?l1)
      (road ?l1 ?l2))
  :effect (and
      (not (at ?v ?l1))
      (at ?v ?l2)
      (increase (total-cost) (road-length ?l1 ?l2))))
(:action pick-up
  :parameters (?v - vehicle ?l - location ?p - package
               ?s1 ?s2 - capacity-number)
  :precondition (and
      (at ?v ?l)
      (at ?p ?l)
      (capacity-predecessor ?s1 ?s2)
      (capacity ?v ?s2))
  :effect (and
      (not (at ?p ?l))
      (in ?p ?v)
      (capacity ?v ?s1)
      (not (capacity ?v ?s2))
      (increase (total-cost) 1)))
(:action drop
  :parameters (?v - vehicle ?l - location ?p - package
               ?s1 ?s2 - capacity-number)
  :precondition (and
      (at ?v ?l)
      (in ?p ?v)
      (capacity-predecessor ?s1 ?s2)
      (capacity ?v ?s1))
  :effect (and
      (not (in ?p ?v))
      (at ?p ?l)
      (capacity ?v ?s2)
      (not (capacity ?v ?s1))
      (increase (total-cost) 1)))
```

Formulation of actions in PDDL for `transport-strips`.

```
(:durative-action drive
  :parameters (?v - vehicle ?l1 ?l2 - location)
  :duration (= ?duration (road-length ?l1 ?l2))
  :condition (and
      (at start (at ?v ?l1))
      (at start (road ?l1 ?l2))
      (at start (>= (fuel-left ?v) (fuel-demand ?l1 ?l2))))
  :effect (and
      (at start (not (at ?v ?l1)))
      (at end (at ?v ?l2))
      (at start (decrease (fuel-left ?v) (fuel-demand ?l1 ?l2)))))
(:durative-action pick-up
  :parameters (?v - vehicle ?l - location ?p - package)
  :duration (= ?duration 1)
  :condition (and
      (at start (at ?v ?l))
      (over all (at ?v ?l))
      (at start (at ?p ?l))
      (at start (>= (capacity ?v) (package-size ?p)))
      (at start (ready-loading ?v)))
  :effect (and
      (at start (not (at ?p ?l)))
      (at end (in ?p ?v))
      (at start (decrease (capacity ?v) (package-size ?p)))
      (at start (not (ready-loading ?v))) ; lock vehicle
      (at end (ready-loading ?v)))) ; unlock vehicle
(:durative-action drop
  :parameters (?v - vehicle ?l - location ?p - package)
  :duration (= ?duration 1)
  :condition (and
      (at start (at ?v ?l))
      (over all (at ?v ?l))
      (at start (in ?p ?v))
      (at start (ready-loading ?v)))
  :effect (and (at start (not (in ?p ?v)))
      (at end (at ?p ?l))
      (at end (increase (capacity ?v) (package-size ?p)))
      (at start (not (ready-loading ?v))) ; lock vehicle
      (at end (ready-loading ?v)))) ; unlock vehicle
(:durative-action refuel
  :parameters (?v - vehicle ?l - location)
  :duration (= ?duration 10)
  :condition (and
      (at start (at ?v ?l))
      (over all (at ?v ?l))
      (at start (has-petrol-station ?l)))
  :effect
      (at end (assign (fuel-left ?v) (fuel-max ?v))))
```

Formulation of durative actions in PDDL for temporal Transport.

3. TransportEditor

To enable effective transportation planning, we have developed *TransportEditor*, a system for creating and visualizing transportation problems and plans. Specifically, TransportEditor aims to be a problem editor and plan visualizer for the Transport domain (and its variants). It is an intuitive and cross-platform graphical desktop application written in Java (see attached screenshot).

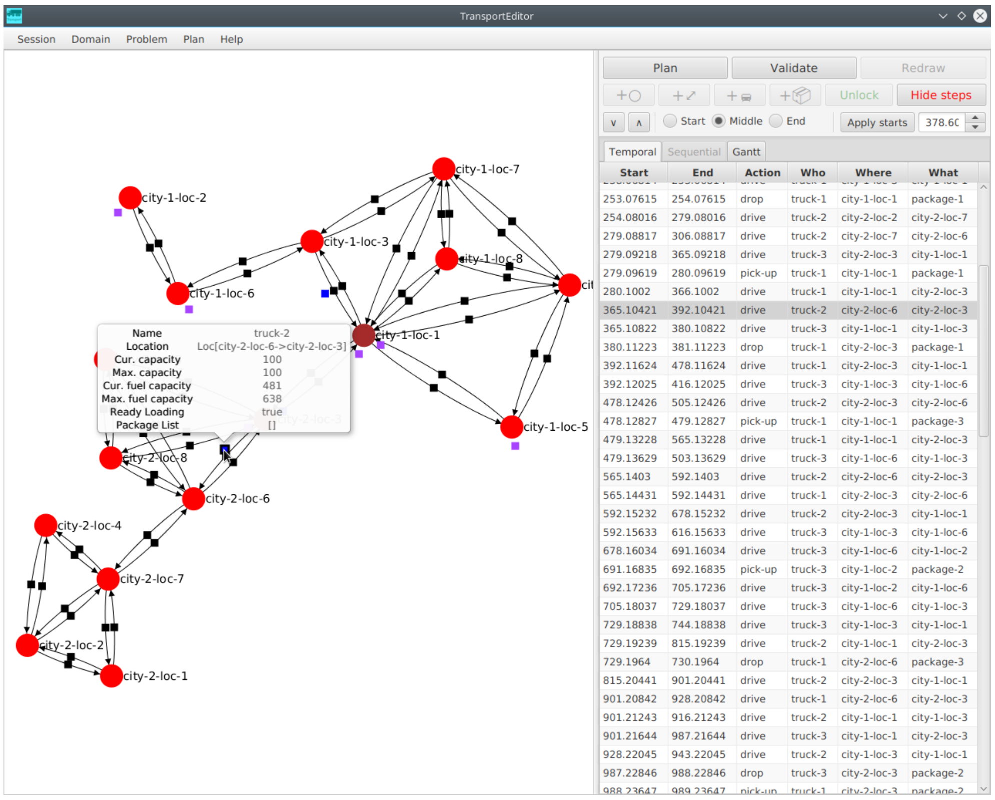
It allows the users to create a planning session, where they select a Transport domain variant, load a problem instance from PDDL (Section 1.4) or create a new one from scratch. The road network of the problem is automatically laid out and visualized for the users as a graph with locations as nodes and roads as edges. Users can then tweak the layout, make changes to vehicle and package properties and export the problem or domain back into PDDL.

They can also select an external planner referencing its executable file, or select one of the built-in planners and try to solve the loaded problem using the selected planner. Internal and external plan validators, like VAL (Howey and Long, 2003), can also be selected to verify that the plans are correct. Once plans are loaded and verified, it will let the users see a list of actions in the plan, or plot a Gantt chart (Gantt, 1910), which is useful for observing concurrent actions in temporal domain variants (Figure 6.5).

The best feature of TransportEditor is the option of tracing plans. We can select any action, specify an exact time point or just step through the actions in order and the road network on the left will display the current state of the problem, as if all actions before the current point were applied to the start state. It is possible to do all of this, and more, without ever leaving the TransportEditor user interface.

TransportEditor will help researchers working on this domain fine-tune their planners; they can visualize the various corner cases their planner fails to handle, step through the generated plan and find the points where their approach fails. A secondary motivation is to be able to test approaches for creating plans for the domain. The basic user workflow of TransportEditor consists of the following steps:

- Selecting which formulation of the Transport domain they want to work with or create their own variant;
- Loading the PDDL or creating their own problem of the given domain. TransportEditor then visualizes the given graph as good as it can;
- Iterating among the following options:
  - Loading a planner executable and letting TransportEditor run the planner on the loaded problem instance for a given time (the user can cancel anytime), then loading the resulting plan;
  - Possibly loading a pregenerated plan;
  - Stepping through the individual plan actions and letting TransportEditor visualize them. The user can step forward and backward in the plan and inspect each action result in great detail;



Name	truck-2
Location	Loc[city-2-loc-6->city-2-loc-3]
Cur. capacity	100
Max. capacity	100
Cur. fuel capacity	481
Max. fuel capacity	638
Ready Loading	true
Package List	[]

Start	End	Action	Who	Where	What
253.07615	254.07615	drop	truck-1	city-1-loc-1	package-1
254.08016	279.08016	drive	truck-2	city-2-loc-2	city-2-loc-7
279.08817	306.08817	drive	truck-2	city-2-loc-7	city-2-loc-6
279.09218	365.09218	drive	truck-3	city-2-loc-3	city-1-loc-1
279.09619	280.09619	pick-up	truck-1	city-1-loc-1	package-1
280.1002	366.1002	drive	truck-1	city-1-loc-1	city-2-loc-3
365.10421	392.10421	drive	truck-2	city-2-loc-6	city-2-loc-3
365.10822	380.10822	drive	truck-3	city-1-loc-1	city-1-loc-3
380.11223	381.11223	drop	truck-1	city-2-loc-3	package-1
392.11624	478.11624	drive	truck-1	city-2-loc-3	city-1-loc-1
392.12025	416.12025	drive	truck-3	city-1-loc-3	city-1-loc-6
478.12426	505.12426	drive	truck-2	city-2-loc-3	city-2-loc-6
478.12827	479.12827	pick-up	truck-1	city-1-loc-1	package-3
479.13228	565.13228	drive	truck-1	city-1-loc-1	city-2-loc-3
479.13629	503.13629	drive	truck-3	city-1-loc-6	city-1-loc-3
565.1403	592.1403	drive	truck-2	city-2-loc-6	city-2-loc-3
565.14431	592.14431	drive	truck-1	city-2-loc-3	city-2-loc-6
592.15232	678.15232	drive	truck-2	city-2-loc-3	city-1-loc-1
592.15633	616.15633	drive	truck-3	city-1-loc-3	city-1-loc-6
678.16034	691.16034	drive	truck-3	city-1-loc-6	city-1-loc-2
691.16835	692.16835	pick-up	truck-3	city-1-loc-2	package-2
692.17236	705.17236	drive	truck-3	city-1-loc-2	city-1-loc-6
705.18037	729.18037	drive	truck-3	city-1-loc-6	city-1-loc-3
729.18838	744.18838	drive	truck-3	city-1-loc-3	city-1-loc-1
729.19239	815.19239	drive	truck-2	city-1-loc-1	city-2-loc-3
729.1964	730.1964	drop	truck-1	city-2-loc-6	package-3
815.20441	901.20441	drive	truck-2	city-2-loc-3	city-1-loc-1
901.20842	928.20842	drive	truck-1	city-2-loc-6	city-2-loc-3
901.21243	916.21243	drive	truck-2	city-1-loc-1	city-1-loc-3
901.21644	987.21644	drive	truck-3	city-1-loc-1	city-2-loc-3
928.22045	943.22045	drive	truck-2	city-1-loc-3	city-1-loc-1
987.22846	988.22846	drop	truck-3	city-2-loc-3	package-2
988.23647	989.23647	pick-up	truck-1	city-2-loc-3	package-2

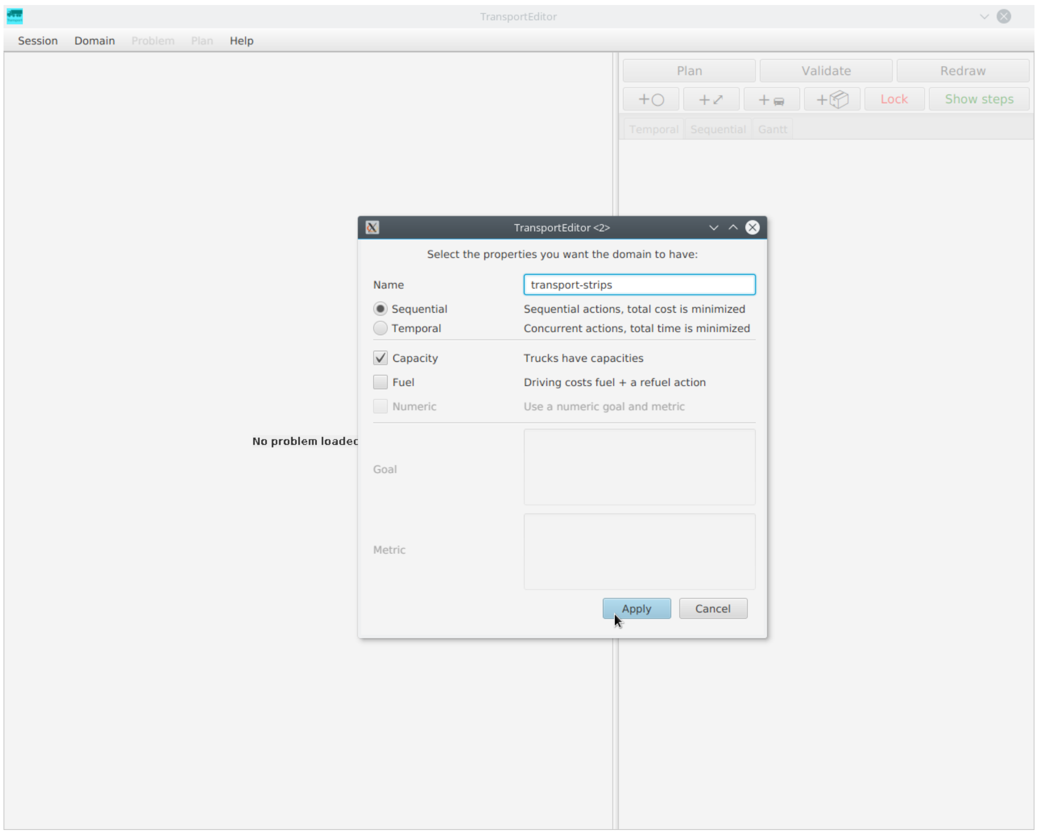
Screenshot of a user tracing actions of a plan for a small temporal problem in TransportEditor. The bubble in the middle shows details of a truck currently driving along a road between `city-2-loc-6` and `city-2-loc-3`. The `city-1-loc-1` location is plotted in a darker shade of red, which signifies that a petrol station is present at the location.

  - Editing the graph: adding/removing/editing the location or properties of vehicles, packages, roads, locations and possibly petrol stations;
  - Saving the currently generated plan;
  - Saving the problem or domain (exporting to a PDDL file);

- Exit the application or go back to the first step.

TransportEditor is a part of this thesis and the reader can find it on the attached CD (see Attachment 1 for more information). Both the TransportEditor User & Developer Manuals are attached to this thesis in a digital format, offering guidance when using the program and providing an in-depth description.

Screenshots displaying typical TransportEditor usage follow.



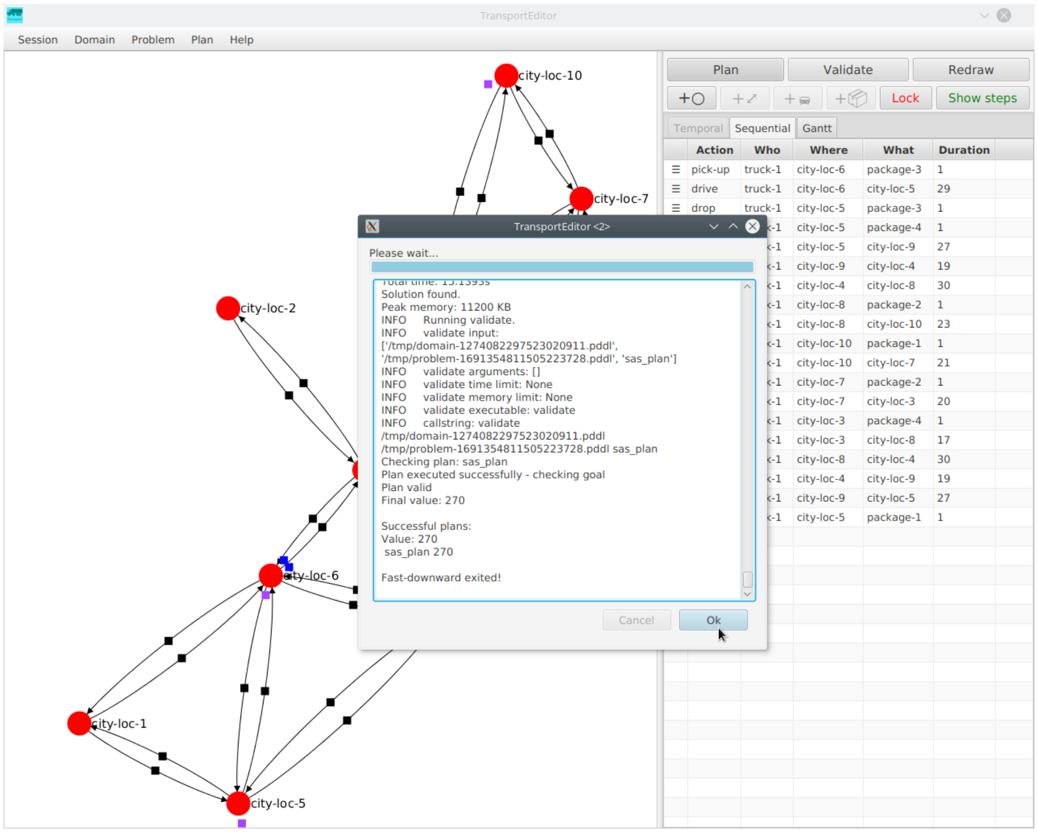
TransportEditor <2>

Select the properties you want the domain to have:

Name	transport-strips
(●) Sequential	Sequential actions, total cost is minimized
( ) Temporal	Concurrent actions, total time is minimized
☑ Capacity	Trucks have capacities
☐ Fuel	Driving costs fuel + a refuel action
☐ Numeric	Use a numeric goal and metric
Goal	
Metric	

Apply　Cancel

Selecting a domain variant.



	Action	Who	Where	What	Duration
☰	pick-up	truck-1	city-loc-6	package-3	1
☰	drive	truck-1	city-loc-6	city-loc-5	29
☰	drop	truck-1	city-loc-5	package-3	1
		k-1	city-loc-5	package-4	1
		k-1	city-loc-5	city-loc-9	27
		k-1	city-loc-9	city-loc-4	19
		k-1	city-loc-4	city-loc-8	30
		k-1	city-loc-8	package-2	1
		k-1	city-loc-8	city-loc-10	23
		k-1	city-loc-10	package-1	1
		k-1	city-loc-10	city-loc-7	21
		k-1	city-loc-7	package-2	1
		k-1	city-loc-7	city-loc-3	20
		k-1	city-loc-3	package-4	1
		k-1	city-loc-3	city-loc-8	17
		k-1	city-loc-8	city-loc-4	30
		k-1	city-loc-4	city-loc-9	19
		k-1	city-loc-9	city-loc-5	27
		k-1	city-loc-5	package-1	1

TransportEditor <2>

Please wait...

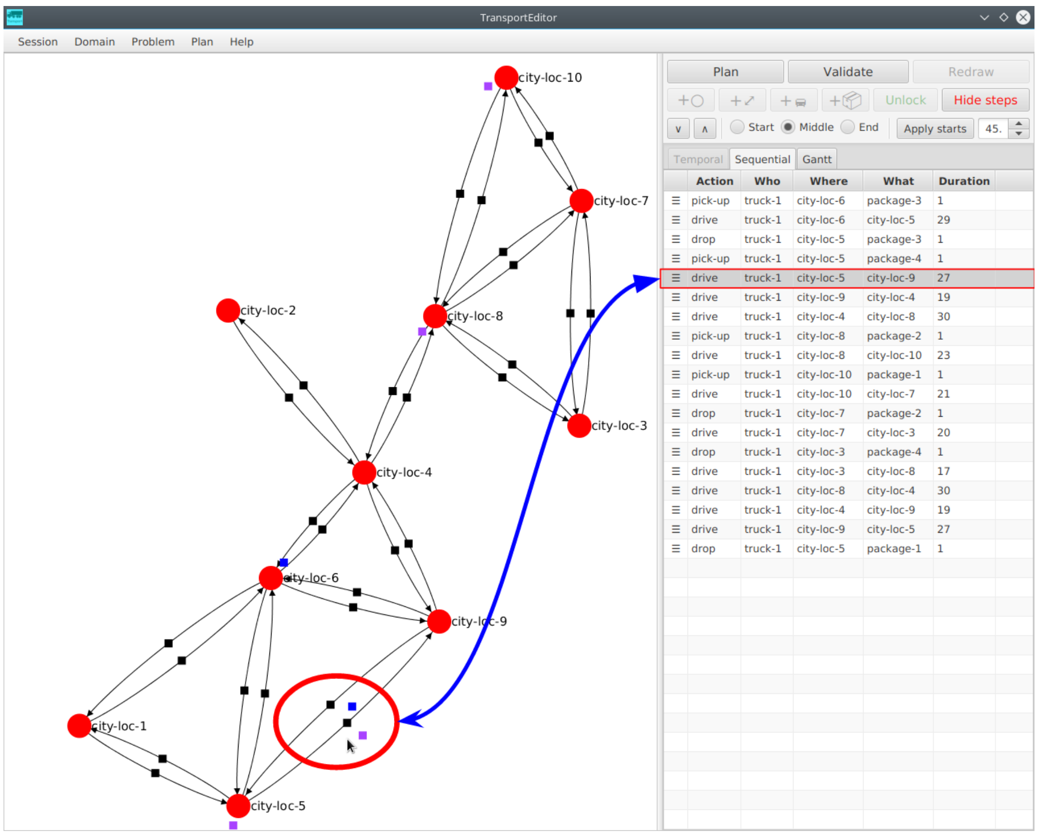
```
Solution found.
Peak memory: 11200 KB
INFO     Running validate.
INFO     validate input:
['/tmp/domain-1274082297523020911.pddl',
'/tmp/problem-1691354811505223728.pddl', 'sas_plan']
INFO     validate arguments: []
INFO     validate time limit: None
INFO     validate memory limit: None
INFO     validate executable: validate
INFO     callstring: validate
/tmp/domain-1274082297523020911.pddl
/tmp/problem-1691354811505223728.pddl sas_plan
Checking plan: sas_plan
Plan executed successfully - checking goal
Plan valid
Final value: 270

Successful plans:
Value: 270
 sas_plan 270

Fast-downward exited!
```

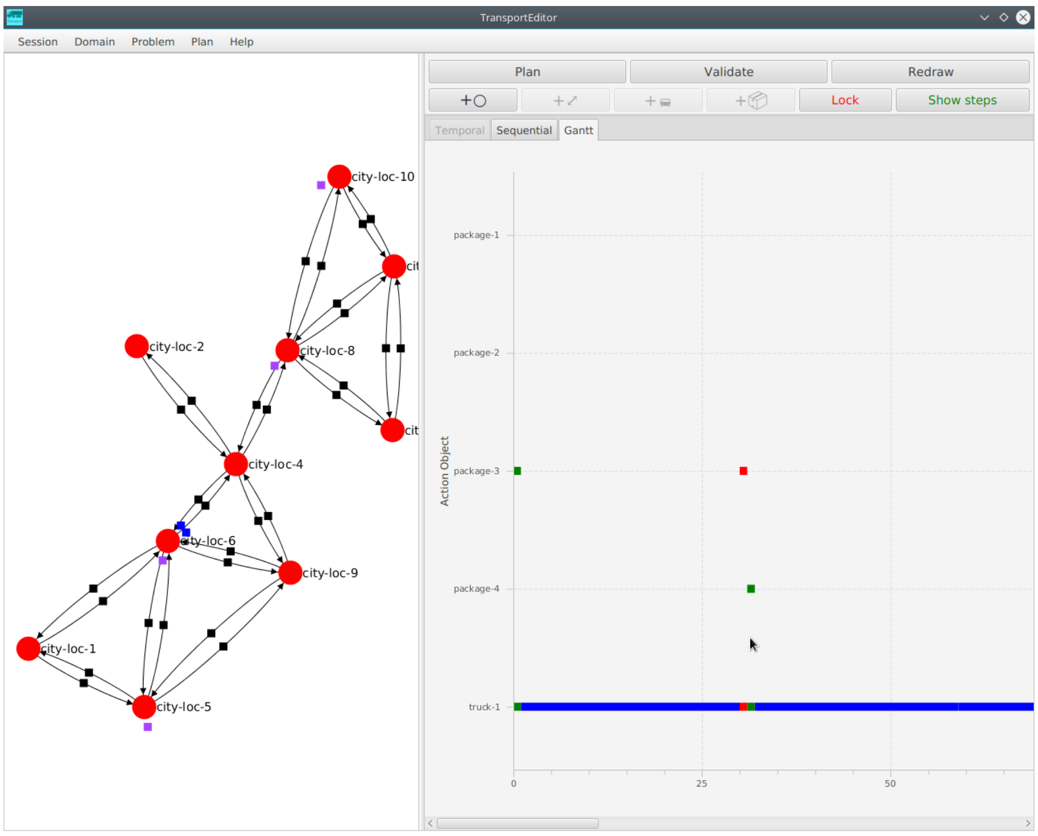
Cancel　Ok

Running an external planner on a sequential problem.



	Action	Who	Where	What	Duration
☰	pick-up	truck-1	city-loc-6	package-3	1
☰	drive	truck-1	city-loc-6	city-loc-5	29
☰	drop	truck-1	city-loc-5	package-3	1
☰	pick-up	truck-1	city-loc-5	package-4	1
☰	drive	truck-1	city-loc-5	city-loc-9	27
☰	drive	truck-1	city-loc-9	city-loc-4	19
☰	drive	truck-1	city-loc-4	city-loc-8	30
☰	pick-up	truck-1	city-loc-8	package-2	1
☰	drive	truck-1	city-loc-8	city-loc-10	23
☰	pick-up	truck-1	city-loc-10	package-1	1
☰	drive	truck-1	city-loc-10	city-loc-7	21
☰	drop	truck-1	city-loc-7	package-2	1
☰	drive	truck-1	city-loc-7	city-loc-3	20
☰	drop	truck-1	city-loc-3	package-4	1
☰	drive	truck-1	city-loc-3	city-loc-8	17
☰	drive	truck-1	city-loc-8	city-loc-4	30
☰	drive	truck-1	city-loc-4	city-loc-9	19
☰	drive	truck-1	city-loc-9	city-loc-5	27
☰	drop	truck-1	city-loc-5	package-1	1

Tracing a sequential plan. Highlighted is the corresponding `drive` action and the visualization of the vehicle (blue square) and the package it carries (purple square) on the road graph.



Visualizing the plan as a Gantt chart.

4. Accompanying software toolkit architecture

To run our experiments efficiently, we have implemented a software project, confusingly called TransportEditor, of which the editor described in the previous section is only a part. The "scaffolding" for running experiments, generating reports, and the planner implementations that we will describe in the next two chapters are all part of the project as well. The project is split into several modules, listed in approximate dependency order:

- `transport-thirdparty`, a set of 3rd party libraries that have not been published in a dependency management repository like Maven Central,[5] making their compile-time download and linking impossible;
- `transport-core`, a module that contains code modeling the Transport domain and persisting it to disk;
- `transport-planners`, containing sequential and temporal planners, both *internal* (implemented as a part of this project) and *external* (only calling an executable of a different project);
- `transport-benchmark`, a set of tools for running repeatable experiments on Transport datasets, possibly in parallel;
- `transport-report`, a collection of table generators and graph plotters, operating on the results of benchmarks; and
- `transport-editor`, the module containing the graphical user interface of the planning system TransportEditor. This module is only dependent on `transport-planners` and all its dependencies.

Independent on the other modules is `transport-docs`, the module containing this thesis, along with our other textual works relating to researching the Transport domain. Information about the model in `core` and the `editor` is available in the TransportEditor User & Developer Manuals. All the modules are written in Java and attached to this thesis (see 1. CD contents for details). Also included are a few shell scripts, usually for quick data conversion. The only important scripts reside in the `tools` directory; the `benchmark.sh` script, used for running experimental benchmarks, and `generate-reports.sh`, used for generating SVG and PDF plots or LaTeX tables of the scores and run times from benchmark results.

The benchmarker takes as input a JSON (Bray, 2014) configuration file, runs the specified benchmarks and returns a `results.json` output file in a different JSON format (see attached grammars). In both of the figures are BNF grammars. We write zero-or-more repeated statements using (and)*. Also, [and] are used for JSON lists, they do not mean an optional statement — we use (and) (without the star) for that. Under `<character>`, we assume any valid UTF-8 character, `<double>` means any non-negative IEEE compatible double precision floating-point number, and `<long>` is any non-negative whole number smaller than or equal to $2^{63} - 1$. Additionally, all file paths may use the `${transport.root}` variable, designating the root directory of the project.

[5] `http://central.sonatype.org/`

Below is the grammar of the input configuration JSON file in BNF. The token `<class>` is expected to be a valid Java class name (including the package) and present on the classpath. The `<executable_templ>` token is expected to be a valid executable planner with supplied parameters and {0} used to represent the domain file path, {1} the problem file path, and {2} the output plan file path.

```
<config> ::= { "domain" : "<filepath>",
               "problems" : { "<problem_name>" : <problem_config>
                          ( , "<problem_name>" : <problem_config> )* },
               "scoreFunctionType"  : "<score_function>",
               "planners": { "<planner_name>" : <planner_config>
                         ( , "<planner_name>" : <planner_config> )* },
               "threadCount": <integer>,
               "timeout": <integer> }
<filepath> ::= <string>
<problem_name> ::= <string>
<problem_config> ::= { "filePath" : "<filepath>",
                       "bestScore" : <score> }
<score> ::= null | <float>
<score_function> ::= TOTAL_TIME | ACTION_COUNT
<planner_name> ::= <string>
<planner_config> ::= { "className" : "<class>"
                   ( , "params" : "<executable_templ>" ) }
<class> ::= <string>
<executable_templ> ::= <string>
<string> ::= ( <character> )*
```

Finally, below is the grammar of the result configuration JSON file in BNF. The tokens `<action>` and `<tpa>` were defined in Section 1.4. Note that this grammar is only valid if concatenated with the previous grammar.

```
<results> ::= { "runs" : [ ( <run> ) ( , <run> )* ] }
<run> ::= { "actions" : [ ( <action> ) ( , <action> )* ],
            "temporalPlanActions" : [ ( <tpa> ) ( , <tpa> )* ],
            "domain" : "<string>",
            "planner" : "<string>",
            "problem" : "<string>",
            "results" : <run_results> }
<action> ::= "<string>"
<tpa> ::= "<string>"
<run_results> ::= { "score" : <score>,
                    "bestScore" : <score>,
                    "exitStatus" : "<exit_status>",
                    "startTimeMs" : <timestamp>,
                    "endTimeMs" : <timestamp>,
                    "durationMs" : <timestamp>,
                    "quality" : <double> }
<exit_status> ::= UNSOVLED | INVALID | VALID | NOTVALIDATED | SUBOPT
<timestamp> ::= <long>
```

Apart from the results file, a plain text log containing information about the planner runs is also produced. Both of these files are placed in the directory `results/config_file_name/YYYYmmdd-HHMMSS/`, under the directory where `benchmark.sh` is placed. We can then use the `generate-reports.sh` script on the results file, which will create a `reports` directory next to the results file, containing all the generated report files.