Bachelor Project

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Efficiency Improvement of Domain-Independent Planning by Integration of Solvers for Euclidean Sub-Problems

Petr Bergmann

Supervisor: Ing. Antonín Komenda, Ph.D.
Field of study: Open Informatics
Subfield: Computer and Information Science
May 2018

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Bergmann Petr**	Personal ID number:	**456914**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Open Informatics**		
Branch of study:	**Computer and Information Science**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Efficiency Improvement of Domain-Independent Planning by Integration of Solvers for Euclidean Sub-Problems

Bachelor’s thesis title in Czech:

Zvýšení efektivity doménově nezávislého plánování pomocí integrace solverů pro Euklidovské podproblémy

Guidelines:

The student will integrate efficient domain-specific solvers into a domain-independent planner in order to improve its efficiency. The integration will seamlessly allow to extend a language for domain-independent planning for the domain-specific fragments.
1) Study literature in the area of automated planning, both classical and domain specific for the Euclidean problems (esp. shortest and multi-goal paths in planar graphs).
2) Prepare and implement planning problems (incl. various parameterizations) compatible with both planning areas.
3) Theoretically analyze similarities and differences in the planning algorithms, specifically type of search and heuristic functions used.
4) Based on 2) and 3), implement integration of the domain-specific solvers into a domain-independent planner so that one unified problem specification language can be used.
5) Experimentally compare efficiency of the domain-independent planner without and with the domain-specific solvers used.

Bibliography / sources:

[1] Malik Ghallab, Dana S. Nau, Paolo Traverso: Automated planning - theory and practice. Elsevier 2004, ISBN 978-1-55860-856-6.
[2] Stefan Edelkamp, Stefan Schroedl: Heuristic Search: Theory and Applications. Morgan Kaufmann. 2012, ISBN 978-0-12372-512-7.

Name and workplace of bachelor’s thesis supervisor:

Ing. Antonín Komenda, Ph.D., Artificial Intelligence Center, FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **09.01.2018** Deadline for bachelor thesis submission: **25.05.2018**

Assignment valid until: **30.09.2019**

Ing. Antonín Komenda, Ph.D.	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Ing. Pavel Ripka, CSc.
Supervisor’s signature	Head of department’s signature	Dean’s signature

Acknowledgements

I would like to thank to my supervisor Ing. Antonín Komenda for sharing his knowledge and for his perfect guidance. Last but not least, my thanks go to my family for their support during my studies.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In Prague, 2. May 2018

Abstract

The purpose of this thesis is to increase efficiency of domain-independent planning. A solver for pathfinding on map background and solver for travelling salesman problem was integrated into a translator level. Thanks to that, huge runtime improvement was achieved. The results of the measurements and implementation are presented in the thesis. The result of the whole thesis is a much faster planner that accepts very simple extension of the input file format.

Keywords: Automated planning, Domain-independent planning, Domain-specific planning, Shortest path problem, Travelling salesman problem

Supervisor: Ing. Antonín Komenda, Ph.D.

Abstrakt

Tato práce se zabývá zvýšením efektivity doménově nezávislého plánování. Solver pro hledání tras na mapovém podkladu a solver na problém obchodního cestujícího byly integrovány do úrovně překladače. Tím bylo dosaženo výrazného urychlení celého plánování. V práci jsou předloženy výsledky měření a způsob implementace. Výsledkem je mnohanásobně rychlejší plánovač příjmající velmi jednoduché rozšíření formátu vstupního souboru.

Klíčová slova: Automatizované plánování, Doménově nezávislé plánování, Doménově specifické plánování, Problém nejkratší cesty, Problém obchodního cestujícího

Contents

Chapter 1

Introduction

A problem of planning is solved in this thesis. It is a part of a field of the artificial intelligence. The planning searches for complex plans in space with deterministic transitions between states. These plans are sequences of steps that are necessary to be made, in order to reach a goal state from some initial state. We recognize two types of planning, domain-independent and domain-specific planning.

Domain-independent planning is an approach that excels in its domain versatility. Any modifications of a planned problem are made in a very short time. There is no need to rewrite a code of a planner as well as there is no need for a new time-consuming debugging. However, this adaptability takes its toll on a running time. It would be great if this handicap could be suppressed while the versatility was preserved.

Then we have a domain-specific planning. Domain-specific solvers are able to solve problems much faster. On the other hand, these solvers are finely tuned for a specific task. For even a simple change in a domain input, large modifications in a planner's code are needed.

That being said, there is a space for a third approach. Some hybrid way that would simultaneously use both planning types. Domain-independent planning for large scope and domain-specific planning for certain subproblems. And that is the goal of this thesis. We want to integrate a domain-specific solver into a domain-independent planner. This should drastically improve running time of the planner. However, this approach will work when certain condition on a domain is met. The domain needs to be separable into two parts. Subproblems that are solvable in the domain-specific way and a main problem containing these subproblems. Needless to say, that this superior problem will be solved by the domain-independent planner. Luckily, most of the common domains meet this condition.

In this thesis, the logistics was selected as the primary domain. This planning domain offers multiple benefits for this type of research. Most importantly, it is a common real-life problem, that is solved by dozens of companies worldwide. Furthermore, this problem can be easily separated into those needed two parts. Specifically, those parts are the domain-specific one, route planning in Euclidean space, and the domain-independent one, finding the optimal path with respect to the smallest distance travelled or least time spent en route. Finally, a huge benefit is that this simple logistics problem can be easily extended by many additional conditions and restrictions. Some of these extensions are shown in next parts.

Last, but not least we would like to present an elegant way of extending a domain description language so it takes into account integrated solvers. This will guarantee ease of use with current planners and allow further implementation of this hybrid planning.

Now, let us describe the domain of the research more in-depth. More information about domain-specific planning are located in following Chapter 2 and about domain-independent planning in Chapter 3. Finally, in Chapter 4 starts a part about our implementation and measurements.

1.1 Domain description

As was stated, primary focus is on logistics. In the simplest form, a hypothetical company that have some trucks (let us call them universally vehicles) is considered. This company delivers packages between many depots. The depot itself can be either warehouse or the final customer. Every package has its starting location and can relate to some goal depot. Our task is to find optimal path of vehicle fleet for delivering all the packages to their final destinations. The issue is that every depot has its own GPS coordinate in the city, so, for solving the task, the shortest paths between all the depots must be found throughout the process. The preview of a domain in Planning Domain Definition Language (PDDL) is on Listing 1.1.

There are four predicates. *Road* predicate describes if there is a road

```
(define (domain transport)
 (:requirements :action-costs)

 (:predicates
  (road ?l1 ?l2)
  (at ?v ?at)
  (package-at ?p ?at)
  (package-in ?p ?in)
 )

 (:functions
  (road-length ?l1 ?l2)
  (total-cost)
 )

 (:action drive
  :parameters (?v ?l1 ?l2)
  :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 load
  :parameters (?v ?p ?where)
  :precondition
   (and (at ?v ?where)
    (package-at ?p ?where))
  :effect
   (and (not
    (package-at ?p ?where))
    (package-in ?p ?v))
 )

 (:action unload
  :parameters (?v ?p ?where)
  :precondition
   (and (at ?v ?where)
    (package-in ?p ?v))
  :effect
   (and (not
    (package-in ?p ?v))
    (package-at ?p ?where))
 )
)
```

Listing 1.1: PDDL logistics domain.

between two locations. So, for example, expression (*road house mall*) implies there is a road from the house to the mall. Predicate *at* defines the position of a vehicle on some location. Last predicates *package-at* and *package-in* defines location of package and if it is carried by vehicle, respectively.

The functions are needed for minimization of the road travelled. Function *road-length* defines length of road between two locations and function *total-cost* is the cost minimized by planner.

Action *drives* moves vehicle from one location to the other. It can be applied when the car is on the start location and road between these two location exists. Second action *load* transports a package from the ground to the vehicle. The precondition is that the vehicle and the package are at the same location. Third action *unload* works in the same manner as the action *load*, only it transports the package from the vehicle to the ground.

Additional information about PDDL are provided in Chapter 3.

1.2 OpenStreetMap

To be able to solve the task, it is necessary to use some map background. From this map intersections will be exported and added to our planner thereafter. OpenStreetMap has been chosen to provide this data. This project is most probably the biggest free map database. The maps are created on collaborative base by volunteers all over the world. The maps are formatted in so called OSM XML file format and its advantage is that it is widely supported by both planning libraries and map exporters. However, the map does not contain only routes, but also landscape and points of interest. Because of that, preprocessing of the whole map will be required to be able to plan the route with our domain-independent planner. This leads us to the next section.

1.3 Chosen domain-specific solver

For route planning, GraphHopper solver [8] was selected. This domain-specific solver is an open source route planning Java library. It uses OpenStreetMap as map resource and therefore it perfectly suits our needs. Furthermore, this library is core of paid product GraphHopper Directions API that is widely used for route optimization and planning by companies, such as Deutsche Bahn, FlixBus or Norwegian post Posten [9]. In default settings, GraphHopper is able to plan either vehicle or bike route with respect to total time travelled or total distance travelled. Additionally, it supports multiple routing algorithms like Dijkstra, A* or its bidirectional variant. More information about these routing algorithms is in next section.

1.4 Chosen domain-independent planner

From the field of domain-independent planners, we have chosen SymBA* planner [1]. This optimal planner has won several awards, among others, International Planning Competition 2014 [12] and it has proven its capability to solve numerous domains. Being an optimal planner, SymBA* satisfies our need for minimization of total distance or time spent en route, as input file SymBA* accepts Planning Domain Definition Language (PDDL format). As a result, all of the domain-independent problems and domain descriptions will be in this language through the whole thesis.

Chapter 2

Domain-specific planning

As stated in the Introduction, the focus will be put on domain specific planning in Euclidean space. From now on, let us restrict ourselves only to the planning in Euclidean space and let us offer more in-depth analysis.

Into mentioned subset belong many real-life path planning problems. Including logistics, domain described in Chapter 1. Other very popular problems are Vehicle Routing Problem and Travelling Salesman Problem. Both of them are strongly connected to the logistics. Although Vehicle Routing Problem will not be solved in this thesis, Travelling Salesman Problem solver will be integrated into the domain-independent planner in Chapter 8.

2.1 Uninformed search

In the area of search, two types, informed and uninformed search, are recognized. As the name suggests, uninformed search algorithms do not use any information about the problem. They work in brute force manner. The whole search is reduced into plainly generating successor states of visited nodes. Therefore, it is not as efficient as it can be. However, this does not mean that all of the uninformed search algorithms have to be primitive. An example of more sophisticated algorithm is Dijkstra's algorithm. [5]

2.1.1 Dijkstra's algorithm

Dijkstra's algorithm is one of the algorithms used by GraphHopper library. Named after its inventor Edsger W. Dijkstra, the algorithm uses a greedy search strategy for weighted graphs. Given a source node and a graph, it is used for finding the minimal cost path from the source node to every other node in the graph. Obviously, after a small modification, this algorithm can also be used for finding the shortest path just between two nodes. Not every node in the whole graph. It is important that Dijkstra's algorithm is optimal in contrast to some of the other uninformed search algorithms like Breadth first search. Another advantage of this algorithm is its time complexity $\mathcal{O}(|E| + |V| \log |V|)$, where $|E|$ is a number of edges and $|V|$ is a number of vertices in given graph.

The procedure is quite straightforward. Initially, the distance to every node except the source one is initialized to infinity. Then in loop the node with the smallest distance is selected and for every neighboring unvisited node new distance is calculated. If new shorter distance was found, the node's value is updated. The formula for the calculation is $distance(u) = distance(v) + edge\_length(v, u)$, if the node v is the current one and the node u is a successor node [5]. This repeats until all of the nodes in the graph are visited. When the algorithm is finished, the shortest distance to every node is calculated.

To conclude, this algorithm expands search space based on the distance from the start. Choosing the closest nodes ensures inability to select node with path worse than optimal. Although this approach is optimal and quite fast, it might be more effective to expand search space using approximate distance from the goal. And this leads us to the second type of search.

2.2 Informed search

The second type of search is informed search. Search strategy that uses knowledge about the domain. It uses heuristic function to determine the shortest path to the goal. Thanks to this approximate value, the search space can be expanded more effectively. The expansion of nodes that are closer to the goal will be prioritized and the ones that are further will be left intact. It is important to emphasize that the heuristic gives us only the approximation of the smallest distance. It is not necessarily the real value itself. In this context, concept of an admissible heuristic is recognized. It has the following definition: "An estimate h is an admissible heuristic if it is a lower bound for the optimal solution costs; that is, $h(u) \leq \delta(u, T)$, for all $u \in V$." [5, p.18], where δ is a distance function.

2.2.1 A* algorithm

The very best example of an algorithm that uses this type of search is A*. Algorithm that is also used by the GraphHopper library. A* is an algorithm that was first described in 1968. [10] As said, it belongs to the informed search class and it is an extension of Dijkstra's algorithm with time complexity $\mathcal{O}(|E|)$, where $|E|$ is the number of edges in the graph. For the search itself it uses both cost to reach goal and the estimated cost to get to the goal. Formula $f(u) = g(u) + h(u)$, where $f(u)$ is estimated cost of the cheapest solution through node u and $h(u)$ is heuristic value, the cost to get from node u to the goal. The value $g(u)$ is the cost to reach the node u and it is the same value as the value $distance(u)$ in the formula for Dijkstra's algorithm in the previous section. [18, p.93] Except that, the algorithm works similarly to the Dijkstra. The only difference is that instead of $distance(u)$, $f(u)$ is calculated. The node with the lowest f is always selected and f is recalculated for all of the neighboring nodes. This process is repeated until the goal state is found.

Chapter 3

Domain-independent planning

In this chapter, short summary of the domain-independent planning is given. As in the chapter about the domain-specific planning, this summary will be primarily focused on the planning needed in the following research and techniques used by the chosen planner SymBA*. And now let us introduce the representation of the problem in the domain-independent planning.

3.1 Classical representation

To represent a planning problem in more sophisticated way than enumerating all possible states, few equally expressive representations were invented. Classical representation, state-variable representation and set-theoretic representation are the most prominent ones. In this text, the classical representation will be discussed as this is the representation SymBA* planner uses.

The classical representation contains states and actions. Both of them are represented as a set of predicates and logical connectives. The planning problem is represented by three items: initial and goal states and a domain. The classical planning domain contains set of all possible states, set of actions and a transition function that changes current state by applying an action. For an action to be applicable, its preconditions must be fulfilled. If an action is applied, the current state is modified by its positive and negative effects [5].

3.2 PDDL

To describe classical representation for planner, Planning Domain Definition Language (PDDL) was developed in 1998. The PDDL is a standardized language for describing classical planning problems in the classical representation. It uses two separated files, one for the domain and one for the problem encoding. The domain file consists of descriptions of all possible actions and standard predicates known from first-order logic. Where all of the actions have their own parameters, preconditions and effects. On the other hand, in the problem file there are actual objects, and the initial state and the goal state. Both states use predicates from the domain file.

All in all, use of the PDDL for planning is quite straightforward and simple. Due to this simplicity of the PDDL structure, the language can be easily extended. There are several successful extensions of the PDDL. For example, PPDDL used for probabilistic planning [19] or MA-PDDL for multi-agent planning [14]. This simplicity of the PDDL is the primal reason why the PDDL was selected as the primary domain description language for this thesis. Our own extensions of the PDDL will be introduced in Chapters 6 and 8.

3.3 State-space planning

So far, the representation of the domain and its description for planners was introduced. Let us explain how do those planners actually work.

Planning algorithms in the classical representation fall into one of two big classes. The first one is a plan-space planning where a search space consist of partially specified plans and arcs are plan refinement operations that completes the plan. This representation will not be described any further as SymBA* planner uses the second planning class, state-space planning. For this type of planning, the search space is a set of all states and arcs are possible actions. The required plan is a path in a search space from initial state to the goal state.

For finding this plan, among others, a forward search is used. The procedure of this search strategy is fairly simple. Three steps are repeated. At first, it is checked if the goal state was not already found. Then all possible actions applicable in this state are enumerated. And finally, all possible successors are computed. The only real problem is with choosing the correct action. This is the place where heuristics come to play. Without heuristics in the worst case the whole state space could be searched before finding the goal [5]. As the planning itself is at least PSPACE-complete [6], research of heuristic functions is ongoing.

Most of the optimal planners uses A*, algorithm described in previous chapter, for exploring states. As in Euclidean space, admissible heuristic is necessary for optimal planner. Specificaly SymBA* planner use symbolic bidirectional variant of the A*. That is an A* that progresses from initial and from goal state at once, as the heuristic functions it uses several types of abstraction heuristics. These heuristics use mapping from state space into an abstract space. This mapping must preserve costs between states and improves the search time [2] [13].

Chapter 4

Comparison of speed

In the last two chapters, theory about domain-specific and domain-independent planning was discussed. Let us now dive more deeply into the comparison of these types of planning towards our goal, integrating the domain-specific solver into the domain-independent planner.

First of all, it would be useful to show the difference between these types of planning in terms of running time. It was stated in Chapter 1 that domain-specific planner dominates domain-independent planner when they solve the same task. This difference is due to different asymptotic complexities. The domain-independent planning is PSPACE-complete, yet algorithms like Dijkstra or A* has polynomial complexity.

This hypothesis should be proven before moving any further. In case of refutation of this claim, integrating domain-specific solver into the domain-independent planner would not make any sense as it would not be an improvement. On the other hand, confirmation would give us first glimpse of scale of possible improvement and it would open the way for further investigation.

4.1 Measurement

For our measurement, six locations in Prague were randomly selected. Planner will search for the shortest route between three pairs of those locations.

Before the planning itself it is required to crop the map so it contains only intersections in neighborhood of starting location. These intersections and distance between them will be then added into PDDL file describing the problem. Those distances are easily obtained from difference of GPS coordinates from .osm file using simple relationship [7]. This neighborhood export can be done only with breadth first search with depth limit. With neighborhood exported we can rebuild .osm file for GraphHopper and export PDDL containing routes between intersections and their corresponding length for SymBA*.

4.1.1 Measurement 1

First route is from Resslova crossroad with Václavská to Spálená crossroad with Národní. Goal is in depth 69 and the total length of road is 1591 m. The results for this measurement are in Tables 4.1 and 4.2.

		SymBA* planning			**Total**
Depth	**To PDDL**	**Translator**	**Preprocessor**	**Search**	**[s]**
100	2.13	25	14	1	**42.13**
150	2.15	200	126	2	**330.15**
200	2.65	299	261	3	**565.65**

Table 4.1: SymBA* results for Measurement 1.

Depth	**Init**	**Search**	**Total [s]**
100	0.56	0.03	**0.59**
150	1.16	0.04	**1.20**
200	1.16	0.02	**1.18**

Table 4.2: GraphHopper results for Measurement 1.

4.1.2 Measurement 2

In the Tables 4.3 and 4.4, are results for route number two. This route is from Strakonická crossroad with Hořejší nábřeží to Jindřicha Plachty crossroad with Nádražní. Goal is in depth 23 and the total length of road is 1188 m.

Depth	To PDDL	SymBA* planning Translator	Preprocessor	Search	Total [s]
50	2.11	1	0	0	**3.11**
100	2.18	21	3	0	**26.18**
150	2.11	107	37	0	**146.11**
200	2.20	-	-	-	**-**

Table 4.3: SymBA* results for Measurement 2.

Depth	Init	Search	Total [s]
50	0.32	0.03	**0.35**
100	0.50	0.03	**0.53**
150	0.72	0.03	**0.75**
200	1.03	0.03	**1.06**

Table 4.4: GraphHopper results for Measurement 2.

4.1.3 Measurement 3

Last route is from Ječná crossroad with Karlovo náměstí to Ječná crossroad with V Tůních. Goal is in depth 16 and the total length of road is 482 m. The results are shown in Tables 4.5 and 4.6.

Depth	To PDDL	SymBA* planning Translator	Preprocessor	Search	Total [s]
25	2.07	0	0	0	**2.07**
50	2.14	2	0	0	**4.14**
100	2.18	55	10	0	**67.18**
150	2.23	230	88	0	**320.23**
200	2.22	295	150	0	**447.22**

Table 4.5: SymBA* results for Measurement 3.

Depth	Init	Search	Total [s]
25	0.27	0.03	**0.30**
50	0.35	0.03	**0.38**
100	0.59	0.03	**0.62**
150	0.87	0.03	**0.90**
200	1.34	0.05	**1.39**

Table 4.6: GraphHopper results for Measurement 3.

4.2 Results

From Tables 4.1-4.6 and Figure 4.1, it is clearly visible that the hypothesis was correct. The difference in running times in respect to number of nodes is especially seen in the Figure 4.1. It shows this dependence of running time in logarithmic scale. GraphHopper increases polynomialy and in the graph is nearly constant. On the other hand, SymBA* grows exponentially and much faster than the GraphHopper. This is exactly the behaviour expected as the domain-independent planning was supposed to be PSPACE-complete and domain-specific solver has polynomial asymptotic complexity.

GraphHopper is by far faster than the domain-independent planner. It will be shown in Chapter 6 how this problem can be tackled, but first, allow us to check how fast domain dependent planner really is.

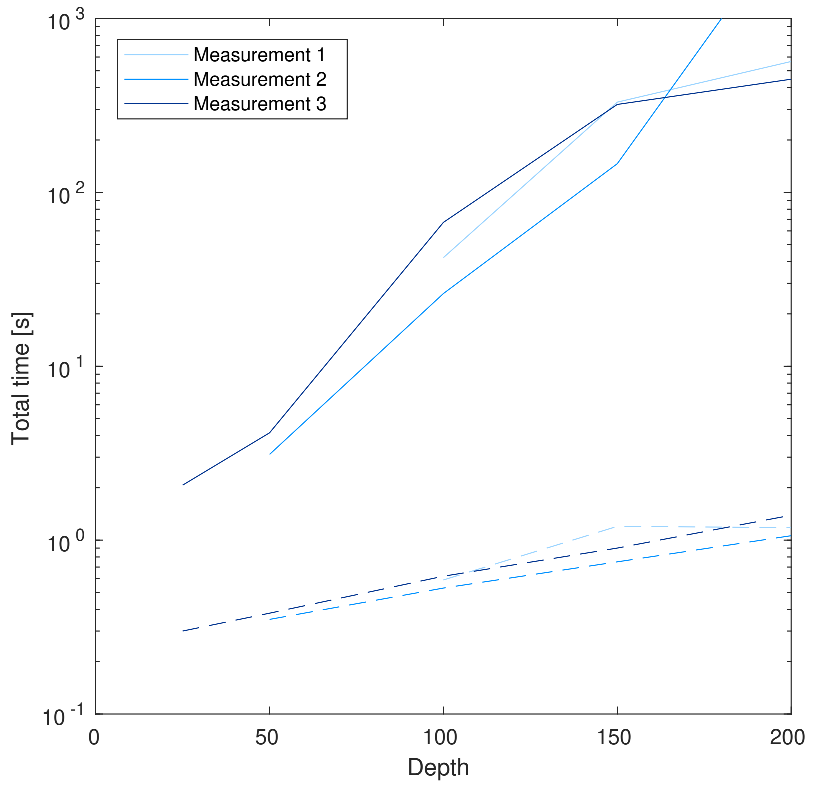


Figure 4.1: Graph of measurements run time. On the vertical axis there is total time on a logarithmic scale.

Chapter 5

Test of domain-specific planner on larger data sets

In the previous chapter, running times of SymBA* and GraphHopper were compared. Results were decisive, GraphHopper was faster than the domain-independent planner. This has confirmed the hypothesis stated in the introduction that domain specific solver will solve problems with polynomial complexity and domain-independent planner with exponential complexity.

However, the decisiveness of the last test did not provide us any data about possible upper bound of GraphHopper planning capabilities. Knowing this data would be useful for following measurements as they would provide useful insight of possible size of planned problems.

5.1 Measurement

In this new measurement we are looking for a boundary on size of input problem. Boundary, when the GraphHopper starts to return data in approximately four hundred seconds as SymBA* in the last measurement did. For this experiment, multiple data sets highly differing in size were prepared:

On every map, several locations were selected to determine speed of the planner. The selected paths are shown in Table 5.1.

- Prague center
- Whole Prague
- Central Bohemian Region
- Czech Republic

In this measurement, it is no longer needed to crop the map to neighborhood of start. The whole map is used because of an expectation that runtime won't be affected much by just few crossroads. Additionally, different map sizes will act as neighborhood size did in previous experiment.

5.2 Results

After conducting measurement, results shown in Table 5.2 were obtained.

Test No.	Map	Road
1	Prague center	Ječná - Ječná
2	Prague center	Strakonická - Jindřicha Plachty
3	Prague center	Reslova - Spálená
4	Whole Prague	Vyšehradská - Rohanské nábřeží
5	Central Bohemia	Rohanské nábřeží - Mladá Boleslav
6	Czech Republic	Praha - Brno
7	Czech Republic	Praha - Krnov
8	Czech Republic	Aš - Jablunkov

Table 5.1: Paths corresponding to measurements.

Test No.	Map	Size	Search time [s]
1	Prague center	<55 MB	**1.382**
2	Prague center	<55 MB	**1.059**
3	Prague center	<55 MB	**1.179**
4	Whole Prague	55 MB	**8.570**
5	Central Bohemia	976 MB	**56.051**
6	Czech Republic	8 GB	**418.023**
7	Czech Republic	8 GB	**432.979**
8	Czech Republic	8 GB	**441.022**

Table 5.2: Results of test of GraphHopper on larger data sets.

Right now, it is clearly visible how significant time improvement can be achieved using integrated planner like GraphHopper into domain-independent planner for solving subproblems. Planning over the whole Czech Republic took approximately the same time as when planning with SymBA* over an area with radius of not even few thousand meters.

Chapter 6

Integrating GraphHopper results into SymBA* planner

In the previous chapter, we have measured the difference in running time of the planning with SymBA* and GraphHopper. Additionally, we have found the approximation of the maximum problem size using GraphHopper. And now, it is time for our main task. The process of integrating the domain-specific solver into the domain-independent planner.

This integration will significantly speedup the whole planning process. The integration itself will be conducted in several steps. First of all, the SymBA*'s preprocessor and translator will be completely bypassed using separate preprocessor with integrated GraphHopper. Secondly, GraphHopper will be called to parse the input file and create the PDDL for the planner. And finally, the GraphHopper will be integrated into the core of the translator of SymBA*.

These separate steps will give us bigger insight on possible approaches and their total speedup. These steps are from the least robust to the most robust implementation (integration int the translator) as no heuristic function in these steps are skipped.

6.1 Preprocessor and translator

In Chapter 4, initial measurement of domain-independent planner was conducted. In the measured data, it is clearly visible that the computational time is primarily spent in translator and preprocessor part of planning. The search itself does not take much time. This trend is due to numerous heuristics and functions, outlined in Chapter 3, that SymBA* tries to apply on the data in those first two parts.

However, this whole process can be omitted. It is possible to make standalone program that will output the same file as SymBA* preprocessor does, yet no additional functions will be applied. This approach has one big advantage. Our knowledge about the domain can be used. Thanks to it the whole process will be much faster.

SymBA* preprocessor is taken over from Fast Downward planner. The first part of the format is the same as a format of translator output. This translator format is clearly specified by Fast Downward authors [16]. Additionally, there are three types of generated graphs that are used to speed up the search. They are Domain Transition Graph, Causal Graph and Successor Generator. All of these graphs are necessary to be created in order that the search is working.

6.2 Measurement with own preprocessor

Standalone preprocessor was implemented as an external Java program. Its output is the same as the original output format. The only difference is that there is no extra preprocessing like finding unreachable states etc. Although it speeds up the subproblem of path finding, it will become disadvantage if this approach on the whole logistics domain would be used. Thus, later on, better solution has to be found.

However, results for pathfinding are pleasant. Referring to the original measurement, search phase contributed in most cases with less than one second across all tests. Now, even though our preprocessor transfers computational time towards the search phase, satisfactory results were gained.

Depth	Previous result [s]	Conversion to PDDL	Search	Total [s]	Ratio [%]
Measurement 1					
100	**42.132**	2.349	22	**24.349**	57.792
150	**330.152**	2.339	51	**53.339**	16.156
200	**565.646**	2.370	57	**59.370**	10.496
Measurement 2					
50	**3.105**	2.138	0	**2.138**	68.857
100	**26.175**	2.206	4	**6.206**	23.710
150	**146.107**	2.211	10	**12.211**	8.358
200	**417.203**	2.333	14	**16.333**	3.915
Measurement 3					
25	**2.071**	2.219	0	**2.219**	107.146
50	**4.138**	2.075	1	**3.075**	74.311
100	**67.177**	2.205	4	**6.205**	9.237
150	**320.23**	2.360	7	**9.36**	2.923
200	**447.221**	2.328	7	**9.328**	2.086

Table 6.1: SymBA* results for measurement with our own preprocessor.

From Table 6.1 it is clear that the search itself is indeed longer, but the planning as a whole is much shorter and this difference is getting better and better with larger search space.

6.3 Creating input PDDL from GraphHopper

To get rid of the disadvantage described above, a slightly different approach must be used. Up to this point the search space concept was used. Meaning every intersection is saved in the PDDL and SymBA* planner finds both fastest paths and optimal logistics plan. Let us use not only our knowledge about domain but use the whole GraphHopper on the input of the SymBA*. This is the next step in integrating domain-specific and domain-independent planners.

As a first step, precomputed paths and corresponding distances can be simply saved from GraphHopper into the input PDDL. Thanks to this approach SymBA* will not need to look for optimal routes between depots. In this task, it has proven to be dominated by the domain-specific planner in all tests up to this point. Additionally, SymBA* will still be able to make complete preprocessing on the problem. This will be proven useful especially on more complicated domains.

6.3.1 Implementation

For this approach, the small notation language was created. It contains only three specific constructs and the filepath to the OSM file on the first line of the file. First of those constructs is a word *depot* that is followed by latitude and longitude, second construct is *package* that is followed by two numbers: number of starting depot and number of goal depot. The depots are numbered in order of appearance in the input file. The last construct is *vehicle*, that is followed by number of depots where vehicle starts. Again, number of depots are in order of appearance.

Having this notation language, a PDDL precompiler was implemented in Java. It contains parser that handles reading of the input file and creates formal encapsulation of this file for further use. This file encapsulation uses three additional classes, one per language construct, and they provide functions for easier handling of these constructs.

Second part of the precompiler is an implementation of the GraphHopper library. It operates over the library core and calls the appropriate functions needed for calculation of the shortest paths in map. All of this is done during the parsing of the input in previous part of the precompiler.

After all of these steps, the shortest routes between every pair of depots are calculated and the final appropriate PDDL is exported by simple routine. This PDDL is completely standard and SymBA* does not need any modifications to work with it.

6.3.2 Measurement

With this implementation of our own PDDL precompiler, results in Table 6.2 were measured. It is necessary to point out that from now on our measurements are conduct on the logistics domain as described in Chapter 1. This change is due to fact it is no longer needed to compare domain-independent and domain-specific planner. Domain-independent planner with integrated domain-specific planner and the one without are compared instead. All the previous measurements were just comparing abilities in finding the shortest paths on the map as was previously stated.

Depth	Export	Without GraphHopper	Export	With GraphHopper
Measurement 1				
20	0.298	**81.298**	0.398	**2.398**
30	0.390	**146.390**	0.488	**2.488**
40	0.352	**264.352**	0.599	**2.599**
50	0.392	**422.392**	0.703	**2.703**
60	0.402	**1628.402**	0.743	**2.743**
Measurement 2				
20	0.298	**1415.298**	0.398	**2.398**
30	0.390	**3655.390**	0.488	**2.488**
40	0.352	**4569.352**	0.599	**2.599**
50	0.392	**6702.392**	0.703	**2.703**

Table 6.2: SymBA* results for measurement with creating GraphHopper PDDL input.

From the results in Table 6.2, it is apparent that with integrated GraphHopper into SymBA* huge speed up was obtained. Let us check if the whole process can be neater.

6.4 Integration into translator

Although it has been proven that PDDL files with precomputed ways are effective the necessity of having external program is limiting. Thus, it would be huge benefit if GraphHopper was completely integrated into SymBA* translator.

The implementation works as follows. New requirement :osm is defined. It is recognized by SymBA* in a domain PDDL file. If this tag is present then problem file is expected to have :osm definition. This definition has following syntax. Whole block is bounded by brackets and contains :osm tag, path to program with GraphHopper and is followed by n components. For every two objects inside, one component path and path-length predicate will be created from GraphHopper. This component definition has syntax like this: at the first place, there is component keyword and then m triplets containing in this order object latitude and longitude. Example of definition is in Listing 6.1 and corresponding graph is in Figure 6.1.

Listing 6.1: Example of osm PDDL extension.

```
(define ...
 (:objects
  depot1 depot2 depot3 depot4 depot5 depot6
   ...
 )
 ...
 (:osm ../graphhopper/map.osm
    (component
        (depot1 50.078349 14.448512)
        (depot2 50.082220 14.449245)
        (depot3 50.080798 14.452959)
    )
    (component
        (depot3 50.080798 14.452959)
        (depot4 50.086275 14.434750)
    )
    (component
        (depot4 50.086275 14.434750)
        (depot5 50.084492 14.413531)
        (depot6 50.075970 14.435111)
    )
 )
)
```

This :osm segment is then parsed, objects and their locations are extracted. Afterwards the GraphHopper implementation on specified file path is called. It needs to be pointed out that GraphHopper is able to use contraction hierarchies. With this setting on, library precomputes data about map and can reuse it in every following search. This feature speeds up the process of pathfinding even more.

When paths are computed, from final distances new predicates path and path-length are constructed. These predicates are for SymBA* the same as if they were read from standard PDDL. Thus, SymBA* is still able to use all heuristic and pruning force to optimize resulting problem and preprocess it for the best performance on logistics problem.

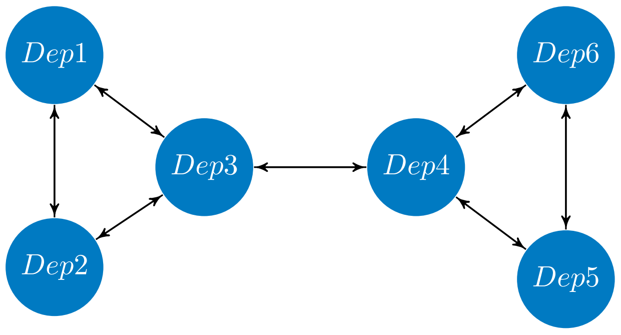


Figure 6.1: Graph corresponding to extended osm PDDL.

6.5 Measurement with integrated GraphHopper

Test No.	Number of Vehicles	Number of Depots	Number of Packages	Total time [s]
1	1	5	5	**5**
2	2	5	5	**5**
3	3	5	5	**6**
4	1	7	7	**5**
5	2	7	7	**8**
6	3	7	7	**13**
7	1	10	10	**9**
8	2	10	10	**29**
9	3	10	10	**272**
Maximum values				
10	1	14	14	**1645**
11	2	13	13	**1995**
12	3	11	11	**935**

Table 6.3: SymBA* results for measurement with integrated GraphHopper.

Right now, we have experimentally shown, that SymBA* is able to plan quite extensive problems in short time. Much better performance than any other previous implementation. In fact, it is no longer possible to even compare results with pure SymBA* because their computation simply takes too long.

To sum up, full integration of the domain-specific planner into the domain-independent one was achieved. It uses the very best of both worlds. Speed of domain-specific planning and versatility of domain-independent one. Solution of logistics problem is quite neatly split into subproblem of finding the shortest path and the main problem of optimal distribution of packages between depots.

In the next chapter application of this enhancement on slightly different and more complicated domains will be presented.

Chapter 7

Extensions of logistics domain

Logistics was the primary field of previous chapters. In this part, other domains will be the subject of the research. It is important to stress out that none of these domains needed any additional change in planner code. The only modification is in adding :osm part to problem file instead of explicit enumeration of all possible roads. This approach strictly corresponds to the topic of this thesis. The integration is seamless and does not bring any complicated requirements on the user side. Especially, the capability of the PDDL extension for definition of multiple components in the graphs is very advantageous in reformulating original tasks and thus easing the work with the language extension.

Some of the benchmark domains used for the optimal planning at IPC were selected as example domains. These domains have publicly available PDDLs. From original PDDL, after few changes and integrating :osm requirement, usable domains with GraphHopper were obtained. All of them are extensions of basic logistics problem thoroughly described in Chapter 1.

In next sections, every domain will be described with according measured data comparing performance with and without integrated GraphHopper.

Two of the domains, Depots and DriverLog, are taken from the Third International Planning Competition [15].

7.1 Capacity

This small extension of logistics adds capacity limit for every vehicle. In previous domain truck was able to carry infinite number of packages. Up to this moment it was not possible to specify any limit. Results are in Table 7.1.

Test No.	Translator	Preprocessor	Search	Total [s]
Pure SymBA*				
1	0	1	161	**162**
2	1	0	2022	**2023**
3	1	0	191	**192**
SymBA* with GraphHopper				
1	2	0	0	**2**
2	0	0	0	**<1**
3	1	0	0	**1**

Table 7.1: SymBA* results for measurement of capacity extension.

7.2 Depots

Depots problem is an extension of logistics towards the domain-independent planning. The previous task is expanded with modelling of crates containing packages and hoists that lifts and drops the crates and packages in every depot. Therefore, this domain is less route planning based and more focused on operations needed in the whole supply chain. Final results of this measurement are in Table 7.2.

Test No.	Translator	Preprocessor	Search	Total [s]
Pure SymBA*				
1	0	0	15	**15**
2	0	0	67	**67**
3	0	1	1586	**1587**
SymBA* with GraphHopper				
1	0	0	1	**1**
2	1	0	1	**2**
3	0	0	5	**5**

Table 7.2: SymBA* results for measurement of depots extension.

7.3 Truck-plane

Truck-plane extension adds new type of transport, airplanes. Thanks to it more complexity of problem is achieved. Moreover, it is more common problem in practice where multiple types of transportation is used in delivery chain. The problem itself is often divided in small enclaves of depots in every city. These groups of depots are connected via airfields.

In our OSM extension the division into enclaves is achieved thanks to component predicates. Every component is detached from the others and no connection between them is constructed. Measurement's outcome is in Table 7.3.

Test No.	Translator	Preprocessor	Search	Total [s]
Pure SymBA*				
1	1	0	7	**8**
2	1	0	3123	**3126**
3	0	0	159	**159**
SymBA* with GraphHopper				
1	0	0	0	**<1**
2	0	0	0	**<1**
3	0	0	0	**<1**

Table 7.3: SymBA* results for measurement of truck - plane extension.

7.4 Intercity transport

This domain is created from truck plane domain by little twist. The objective is to plan inter city transportation. There are some cities. Every city contains its own distribution network. But the whole transport between cities is handled by second type of network.

The main difference from previous domain is that all networks are handled by GraphHopper (the airplanes could not use it). As was proven in Chapter 5, GraphHopper is able to successfully plan routes over big distances, thus the planning will benefit in huge manner in respect to pure SymBA*. Results are in Table 7.4.

Test No.	Translator	Preprocessor	Search	Total [s]
Pure SymBA*				
1	1	0	4	**5**
2	1	0	125	**126**
3	1	0	1752	**1753**
SymBA* with GraphHopper				
1	0	0	0	**<1**
2	0	0	0	**<1**
3	0	0	0	**<1**

Table 7.4: SymBA* results for measurement of intercity transport extension.

7.5 Driverlog

Driverlog domain is modelling logistics from the perspective of one driver and not the company itself. Driver has locations from and where to ride. Additionally, the driver is able to leave the truck and walk onto additional locations. For example, the situation like this can be modelled. Driver must deliver packages between three locations, he must go for lunch, toilet and go shopping. The sequence, in which these tasks will be completed, is only up to him.

Driverlog problem contains two separate types of GraphHopper planning. Primarily the plan of driving to deliver packages and secondly plan of walk in each location. Final results of this measurement are in Table 7.5.

Test No.	Translator	Preprocessor	Search	Total [s]
Pure SymBA*				
1	0	1	3	**4**
2	0	1	617	**618**
3	0	1	1586	**1587**
SymBA* with GraphHopper				
1	0	0	1	**1**
2	0	1	0	**1**
3	1	0	0	**1**

Table 7.5: SymBA* results for measurement of driverlog extension.

Chapter 8

Travelling Salesman Problem

Previous chapters were focused on logistics and the GraphHopper library was used as the extension of SymBA* planner. This approach was thoroughly explored. Initially on basic logistics domain and on the modified domains afterwards.

It would be beneficial to prove that it is possible to integrate different domain-specific solvers into domain-independent planner. This would show robustness and portability of the previous integration principle. And the goal of the current chapter is verification of this possibility.

8.1 Problem specification

As a next domain Travelling Salesman Problem (TSP) was selected. This, well-known and thoroughly surveyed, problem has following formal definition: 'Given a distance matrix between n cities, a tour with minimum length has to be found, such that each city is visited exactly once, and the tour return to the first city.' [5, p.25]

The TSP belongs to NP-complete problem class. Since formulation in the 1930s, heuristics and exact algorithms are still evolving. Despite superpolynomial increase of running time with the number of cities, current heuristics are capable of approximately solving large problems with very small deviation.

As an example one can take World TSP solution, where approximation of a path through 1.9 million cities had length at most 0.049% longer than the optimal path. [11]

8.2 Implementation

Domain-specific planner Concorde [4] with use of QSopt [3] linear programming solver has been chosen for the implementation. Concorde is a widely used TSP solver capable of solving all TSPLIB problems. Regarding the domain-independent planning. SymBA* planner, that was used even in previous parts of the work, will still be used as domain-independent planner.

As a domain for domain-independent planning following problem was selected. Imagine a company that delivers leaflets to all of the apartment houses in a neighborhood. Every household has specific product preferences, carefully determined by market research. All of the company's peddlers must deliver all of the leaflets to all of the specified flats. And achieving this by walking as short distance as possible. To promote the TSP nature of this domain, every flat can be visited only once, otherwise the customers would be upset.

The TSP solver is used to find the shortest path through all of the flats in every apartment house. Afterwards, flats in every house are substituted by one PDDL object representing interior. The distance needed to go from the house entrance through the interior and back again to the house entrance is the distance obtained from the TSP solver. Visualisation is on Figure 8.1 and Figure 8.2.

To integrate Concorde into SymBA* it is necessary to extend the original notation of PDDL problem files. We do so by introducing new TSP description $(: TSP\ (house1\ interior1\ ./h1)\ (house2\ interior2\ ./h2)\ ...)$. After :TSP tag there are n 3-tuples containing (in this order): house object, substituting object of interior and path to corresponding TSP file of house interior. For sake of simplicity, the TSP file is in TSPLIB file format [17] that is one of the supported input formats of Concorde solver. Crop of example PDDL is on Listing 8.1.

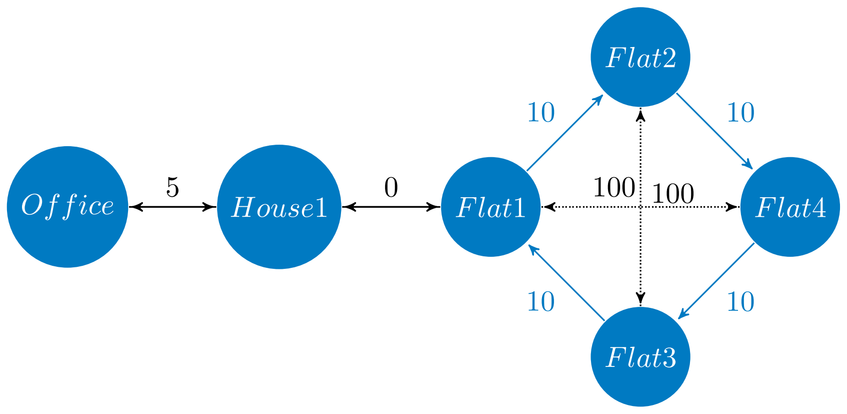


Figure 8.1: Example graph corresponding to original problem. TSP solution has blue color.

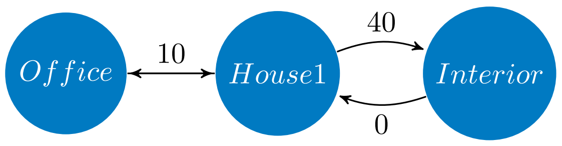


Figure 8.2: Converted example from Figure 8.1 using PDDL extension.

8.3 Measurement

To measure the efficiency of the planner enhancement, several testing problems were created. Most of them contain three houses with different number of flats and one peddler to deliver the leaflets. The measurement focus on path planning itself, so there are no tasks for the peddler to accomplish other than delivering all of the leaflets. But the possibilities are immense due to domain-independent planning. The whole measurement ended up very decisively, as SymBA* with integrated Concorde scored run-time lower than one second for every problem.

Test No.	Size of TSP subproblems	Pure SymBA* Total [s]	With Concorde Total [s]
1	5-4	**2**	**<1**
2	5-7	**11**	**<1**
3	6-5-5	**242**	**<1**
4	5-5-5	**392**	**<1**
5	6-5-6	**832**	**<1**
6	7-5-5	**1654**	**<1**
7	3-3-3-5	**195**	**<1**

Table 8.1: SymBA* results for TSP measurements.

Listing 8.1: Example of PDDL extension for SymBA* with Concorde.

```
(define ...
 (:domain ...)
 (:objects
  house1 house1interior
  house2 house2interior
  leaflet1 leaflet2
  ...
 )
 (:init
  ...
 )
 (:goal
  (leaflet-at leaflet1 house1interior)
  (leaflet-at leaflet2 house2interior)
  ...
 )
 ...
 (:TSP
  (house1 house1interior ../tsp_input/houseMap1.tsp)
  (house2 house2interior ../tsp_input/houseMap2.tsp)
 )
)
```

8.4 Upper bounds

The previous measurement in Table 8.1 did not provide us any upper bound on runtime dependence on number of rooms in a house. Thus, additional data about how long planning takes are needed. The data in Table 8.2 are average values of Concorde's running time. It is the average time of five measurements. These measurements were randomly generated as Euclidean problem of specified size. Their files are given as symmetric succession matrix with distance values between 1 and 100 in TSPLIB file format. These measurements are sent as an input directly to the Concorde solver with no middle layer like SymBA*.

Number of rooms	Time [s]
100	**0.078**
250	**0.734**
500	**3.446**
750	**10.232**
1000	**20.252**
1500	**28.966**

Table 8.2: Concorde running time for domains of different sizes.

Given this data we can estimate runtime needed for planning on domain containing houses of some size. As the Concorde planner is called from within the SymBA*, the translator time will increase by corresponding value from Table 8.2. To support this claim the additional measurement is provided in Table 8.3. Every measurement contains three houses of the same size and once again leaflet delivery is required. Comparing those two tables, the dependence is clearly visible. Time of the translator phase is every time roughly three times the number of Concorde's planning.

Rooms per house	**SymBA* planning**			**Total**
	Translator	**Preprocessor**	**Search**	**[s]**
100	0	0	0	**<1**
250	6	0	0	**6**
500	9	0	0	**9**
750	30	0	0	**30**
1000	57	0	0	**57**
1500	85	0	0	**85**

Table 8.3: Concorde running time for leaflet problem having three houses with the same number of rooms.

Having defined upper bounds, from now on it is known how big domain can be planned in reasonable time. From now on, we can approximately predetermine the time needed for solving specific domain. In the next chapter, the full connection between our original logistics domain and the TSP domain will be shown.

Chapter 9

Logistics with TSP

In previous parts, two PDDL extensions were successfully created. First one uses GraphHopper for finding the shortest route on a map. Second one uses TSP solver for finding optimal solution for delivering leaflets in a house. Up to this point, these extensions were used separately, but in this chapter, we will take a look on their simultaneous performance for solving more robust tasks.

9.1 The domain

The domain used in this chapter will be the same as the one used for the TSP. Again, leaflets will be delivered to neighboring houses. However, this time houses will be placed on a location on the OSM map. This change will result in much more complex planning problem.

The measurement will be conducted in two parts. In the first part, the measurement will focus on comparison of pure SymBA* with enhanced SymBA* with integrated Concorde and GraphHopper. For reasonable comparison, the problems will be small in size. However, in the second measurement results of extended SymBA* on large scale domains will be presented.

9.2 Comparison of planners

As was stated, this measurement compares original SymBA* with extended planner. For this measurement, small OSM extracts varying in size were selected. The OSM extracts are from part of Prague Horni Pocernice. This area was selected because no one-way streets are present in this district. One-way streets are limiting the planner on small area as some depots might have no escape. All of the extracts have the same origin and the starting location is always on the same place.

Results of the measurement are in Table 9.1. Every measurement has its corresponding depth of OSM extract and size of placed houses for TSP. Every measurement that have the same distribution of houses but differing in size of problem have all of the houses placed on the same place for better comparison.

	Size	**SymBA* planning**			**Total**
Depth	**of houses**	**Translator**	**Preprocessor**	**Search**	**[s]**
25	3	0	1	4	**5**
50	3	5	69	72	**146**
100	3	9	150	96	**255**
25	3-4	0	4	80	**84**
50	3-4	6	248	537	**791**
100	3-4	10	345	1377	**1732**
25	3-3-3	0	7	219	**226**
50	3-3-3	8	322	1409	**1739**
25	3-4-3	0	8	515	**523**

Table 9.1: Results of planning on small problems with TSP and OSM background.

Results are very decisive. Very large improvement has been made because every measurement was calculated in less than one second. The success is not only time-wise, but even in simplicity of the input problem PDDL. Just for outlining the difference, a number of lines of the largest PDDL problem file is 9160 lines. The corresponding file for extended SymBA* has 36 lines.

Having compared the original and extended planners, let us check how extended SymBA* plans on domains much bigger in size.

9.3 Large domains

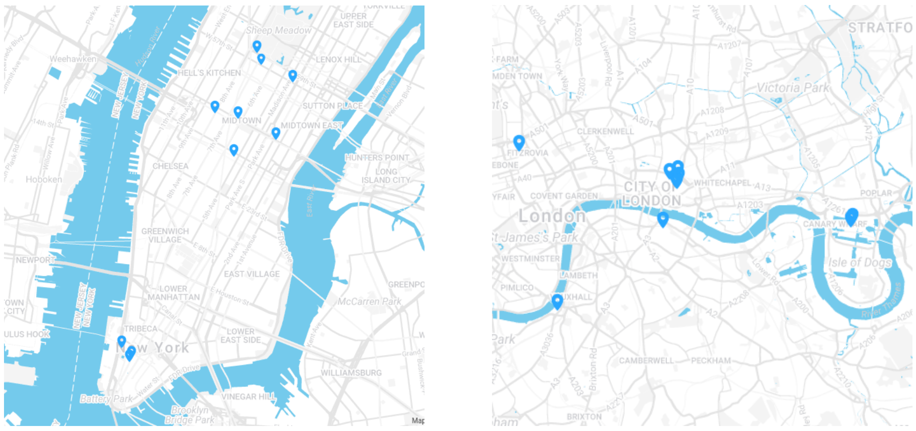
To determine the possible size of large domains, our previous measurements offers good upper bounds on reasonable planning time. Three of them are especially prominent. Primarily, Table 5.2 contains dependence of the GraphHopper's running time on the size of the input OSM file. Secondly, results of the measurement in Table 6.3 gives us a good estimate on maximum number of nodes in the whole problem. And finally, in Table 8.2 there are average running times of Concorde solver for given number of rooms.

Based on these limitations, the problem was created. The aim was to have approximately ten buildings and each one with up to 1500 rooms. Based on these restrictions, two problems were created.

9.3.1 Measurement

In the first one, the goal is to find a way between ten tallest buildings in the New York City. OSM Extract with size of 1.1 GB was selected as the map background and 10,000 rooms were evenly distributed between buildings according to the size of the building. The second problem has same goal. Only the location differs as this one is planned in the city of London. Map has size of 600 MB and thus is half the size of the first map. Again 10,000 rooms were distributed between ten tallest buildings in London. The maps of the selected buildings in both cases are in Figure 9.1.

The measurement was conducted in the SymBA* with both sub solvers, GraphHopper and Concorde. The data of each part of planning are in Table 9.2 for New York City and in Table 9.3 for London.



(a) : New York City **(b) :** City of London

Figure 9.1: Map of planned problems with highlighted locations of skyscrapers.

Concorde	GraphHopper	Preprocessor	Search	Total [s]
175	99	0	997	**1271**

Table 9.2: Result of the measurement for the New York City experiment with simultaneous use of GraphHopper and Concorde.

Concorde	GraphHopper	Preprocessor	Search	Total [s]
165	43	0	1370	**1578**

Table 9.3: Result of the measurement for the London experiment with simultaneous use of GraphHopper and Concorde.

9.3.2 Results

The experiment was very successful. It is important to stress out, that without the solvers just from the OSM the domain would contain 8.2 million and 3.6 million nodes respectively. Additionally, 10,000 nodes would be necessary for the TSP subproblem.

Chapter 10

Conclusion

Our goal was to find a way to improve the efficiency of the domain-independent planning. The primary domain chosen for testing possible improvements was logistics. The open-source path planning library GraphHopper was used and integrated into SymBA* planner. Tests using this implementation have proven themselves to be a huge success as an immense speedup in the planning was accomplished. This increase was observable across all of the domain variant, with practically no difference. The result was a planner that is still domain-independent, yet has fast subsolvers for specific tasks. This planner is therefore able to use different domains while not needing any changes in its implementation, like the standard domain-specific solver would.

To test our approach more thoroughly, new domain and corresponding solver were selected. The solver Concorde for Travelling salesman problem was integrated in the same manner. Its results were as good as the GraphHopper ones. Moreover, the domain that uses both of the solvers at the same time was presented. On this domain, huge tests, based on real world data, were conducted, and they provided a great insight of possible problem sizes.

Our primary goal have been successfully met and so our second task. The integration into standardized planning language called PDDL is seamless and offers comfort and large possibilities in defining various tasks.

To sum up, the goals have been met and hopefully this thesis will provide a good starting ground for any following research or implementations on their own.

Appendix A

Used PDDL domains

In this appendix, the summary of all main PDDL domains that was used throughout the thesis is given. Every section contains domain of the problem with corresponding name.

A.1 Capacity

```
(define (domain capacity)
  (:requirements :typing
     :action-costs :osm)
  (:types
     location target locatable - object
     vehicle package - locatable
     capacity-number - object
  )

  (:predicates
     (road ?l1 ?l2 - location)
     (at ?x - locatable ?v - location)
     (in ?x - package ?v - vehicle)
     (capacity ?v - vehicle
        ?s1 - capacity-number)
     (capacity-predecessor
        ?s1 ?s2 - capacity-number)
  )

  (:functions
     (road-length
        ?l1 ?l2 - location) - number
     (total-cost) - number
  )

  (: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)
        )
      )
    )
```

Listing A.1: PDDL capacity domain.

A.2 Depots

```
(define (domain depot)
(:requirements :typing
     :action-costs :osm)
(:predicates
         (at ?x ?y) (on ?x ?y)
         (in ?x ?y) (lifting ?x ?y)
         (available ?x) (clear ?x)
         (place ?x) (locatable ?x)
         (depot ?x) (distributor ?x)
         (truck ?x) (hoist ?x)
         (surface ?x) (pallet ?x)
         (crate ?x) (road ?y ?z))
(:functions
     (road-length ?l1 ?l2) - number
     (total-cost) - number
  )
  (:types
    location target locatable - object
    vehicle package - locatable
    capacity-number - object
  )
(:action drive
 :parameters (
    ?x ?y - location ?z - location)
 :precondition
        (and (truck ?x)
          (at ?x ?y) (road ?y ?z))
 :effect
        (and (at ?x ?z) (not (at ?x ?y))
         (increase (total-cost)
           (road-length ?y ?z))
         ))

(:action lift
 :parameters ( ?x ?y ?z
?p  - location)
 :precondition
        (and (hoist ?x) (crate ?y)
         (surface ?z) (at ?x ?p)
         (available ?x) (at ?y ?p)
         (on ?y ?z) (clear ?y))
 :effect
        (and (lifting ?x ?y) (clear ?z)
         (not (at ?y ?p))
         (not (clear ?y))
         (not (available ?x))
         (not (on ?y ?z))))

(:action drop
 :parameters ( ?x ?y ?z
    ?p - location)
 :precondition
         (and (hoist ?x) (crate ?y)
          (surface ?z)
          (at ?x ?p) (at ?z ?p)
          (clear ?z) (lifting ?x ?y))
 :effect
         (and (available ?x) (at ?y ?p)
          (clear ?y) (on ?y ?z)
          (not (lifting ?x ?y))
          (not (clear ?z))))

(:action load
 :parameters ( ?x ?y ?z
    ?p - location)
 :precondition
         (and (hoist ?x) (crate ?y)
          (truck ?z) (at ?x ?p)
          (at ?z ?p) (lifting ?x ?y))
 :effect
         (and (in ?y ?z) (available ?x)
          (not (lifting ?x ?y))))

(:action unload
 :parameters ( ?x ?y ?z
    ?p - location)
 :precondition
         (and (hoist ?x) (crate ?y)
          (truck ?z) (at ?x ?p)
          (at ?z ?p) (available ?x)
      (in ?y ?z))
 :effect
         (and (lifting ?x ?y)
          (not (in ?y ?z))
          (not (available ?x))))
)
```

Listing A.2: PDDL depots domain.

A.3 Truck-plane and Intercity transport

```
(define (domain truckPlane)
  (:requirements :action-costs :osm)

  (:predicates
      (package ?obj)
          (truck ?truck)
          (airplane ?airplane)
      (airport ?airport)
      (airway ?airport1 ?airport2)
      (road ?l1 ?l2)
          (at ?obj ?loc)
          (in ?obj ?obj))

      (:functions
       (road-length ?l1 ?l2)
       (total-cost)
      )

(:action load-truck
  :parameters
   (?obj
    ?truck
    ?loc)
  :precondition
```

```
    (and (package ?obj) (truck ?truck)
    (at ?truck ?loc) (at ?obj ?loc))
   :effect
    (and (not (at ?obj ?loc))
     (in ?obj ?truck)))

(:action load-airplane
   :parameters
    (?obj
     ?airplane
     ?loc)
   :precondition
    (and (package ?obj)
      (airplane ?airplane)
      (at ?obj ?loc)
      (at ?airplane ?loc))
   :effect
    (and (not (at ?obj ?loc))
      (in ?obj ?airplane)))

(:action unload-truck
   :parameters
    (?obj
     ?truck
     ?loc)
   :precondition
    (and (package ?obj)
      (truck ?truck)
      (at ?truck ?loc)
      (in ?obj ?truck))
   :effect
    (and (not (in ?obj ?truck))
      (at ?obj ?loc)))

(:action unload-airplane
   :parameters
    (?obj
     ?airplane
     ?loc)
   :precondition
    (and (package ?obj)
      (airplane ?airplane)
      (in ?obj ?airplane)
```

```
      (at ?airplane ?loc))
   :effect
    (and (not (in ?obj ?airplane))
      (at ?obj ?loc)))

(:action drive-truck
   :parameters
    (?truck
     ?loc-from
     ?loc-to)
   :precondition
    (and (truck ?truck)
      (road ?loc-from ?loc-to)
      (at ?truck ?loc-from)
    )
   :effect
    (and (not (at ?truck ?loc-from))
     (at ?truck ?loc-to)
     (increase (total-cost)
     (road-length ?loc-from ?loc-to))))

(:action fly-airplane
   :parameters
    (?airplane
     ?loc-from
     ?loc-to)
   :precondition
    (and (airplane ?airplane)
     (airport ?loc-from)
     (airport ?loc-to)
     (airway ?loc-from ?loc-to)
         (at ?airplane ?loc-from))
   :effect
    (and (not (at ?airplane ?loc-from))
     (at ?airplane ?loc-to)
     (increase (total-cost)
     (road-length ?loc-from ?loc-to))))
)
```

Listing A.3: PDDL truck-plane and intercity transport domain.

A.4 Driverlog

```
(define (domain driverlog)
  (:requirements :strips :osm
        :typing :action-costs)
  (:predicates  (OBJ ?obj)
            (TRUCK ?truck)
        (LOCATION ?loc)
                (driver ?d)
                (at ?obj ?loc)
                (in ?obj1 ?obj)
                (driving ?d ?v)
                (road ?x ?y)
                (path ?x ?y)
                (empty ?v)
)
(:types
     location target locatable - object
     vehicle package - locatable
     capacity-number - object
  )
(:functions
     (road-length ?l1 ?l2) - number
     (total-cost) - number
  )

(:action LOAD-TRUCK
  :parameters
   (?obj
    ?truck
    ?loc - location)
  :precondition
   (and (OBJ ?obj) (TRUCK ?truck)
   (at ?truck ?loc) (at ?obj ?loc))
  :effect
   (and (not (at ?obj ?loc))
```

```
      (in ?obj ?truck)))

(:action UNLOAD-TRUCK
  :parameters
   (?obj
    ?truck
    ?loc - location)
  :precondition
   (and (OBJ ?obj) (TRUCK ?truck)
        (at ?truck ?loc)
        (in ?obj ?truck))
  :effect
   (and (not (in ?obj ?truck))
     (at ?obj ?loc)))

(:action BOARD-TRUCK
  :parameters
   (?driver
    ?truck
    ?loc - location)
  :precondition
   (and (DRIVER ?driver) (TRUCK ?truck)
   (at ?truck ?loc) (at ?driver ?loc)
     (empty ?truck))
  :effect
   (and (not (at ?driver ?loc))
    (driving ?driver ?truck)
    (not (empty ?truck))))

(:action DISEMBARK-TRUCK
  :parameters
   (?driver
    ?truck
    ?loc - location)
    :precondition
     (and (DRIVER ?driver) (TRUCK ?truck)
          (at ?truck ?loc)
          (driving ?driver ?truck))
    :effect
     (and (not (driving ?driver ?truck))
      (at ?driver ?loc) (empty ?truck)))

(:action DRIVE-TRUCK
    :parameters
     (?truck
      ?loc-from - location
      ?loc-to - location
      ?driver)
    :precondition
     (and (TRUCK ?truck)
       (DRIVER ?driver)
       (at ?truck ?loc-from)
       (driving ?driver ?truck)
       (road ?loc-from ?loc-to))
    :effect
     (and (not (at ?truck ?loc-from))
           (at ?truck ?loc-to)
           (increase (total-cost)
           (road-length ?loc-from ?loc-to))))

(:action WALK
    :parameters
     (?driver
      ?loc-from - location
      ?loc-to - location)
    :precondition
     (and (DRIVER ?driver)
          (at ?driver ?loc-from)
          (path ?loc-from ?loc-to))
    :effect
     (and (not (at ?driver ?loc-from))
     (at ?driver ?loc-to)))

)
```

Listing A.4: PDDL driverlog domain.

A.5 Peddler

```
(define (domain peddler)
  (:requirements :typing :action-costs)
  (:types
    location
    leaflet
    postman
  )

  (:predicates
     (road ?l1 - location
        ?l2 - location)
     (at ?v - postman
        ?at - location)
     (leaflet-at ?p - leaflet
        ?at - location)
     (leaflet-in ?p - leaflet
        ?in - location)
     (visited ?l - location)
     (flat ?l - location)
  )

  (:functions
     (road-length ?l1 - location
        ?l2 - location) - number
     (total-cost) - number
  )

  (:action walk
    :parameters (?v - postman
        ?l1 - location ?l2 - location)
    :precondition (and
        (at ?v ?l1)
        (not (flat ?l2))
        (road ?l1 ?l2)
      )
    :effect (and
        (not (at ?v ?l1))
        (at ?v ?l2)
        (increase (total-cost)
            (road-length ?l1 ?l2))
      )
  )

  (:action walk-to-flat
    :parameters (?v - postman
        ?l1 - location ?l2 - location)
    :precondition (and
        (at ?v ?l1)
        (flat ?l2)
        (not (visited ?l2))
        (road ?l1 ?l2)
      )
    :effect (and
        (not (at ?v ?l1))
        (at ?v ?l2)
        (visited ?l2)
        (increase (total-cost)
            (road-length ?l1 ?l2))
      )
  )

  (:action take-leaflet
             :parameters
        (?v - postman ?p - leaflet
        ?where - location)
             :precondition
        (and (at ?v ?where)
        (leaflet-at ?p ?where))
             :effect (and (not
        (leaflet-at ?p ?where))
        (leaflet-in ?p ?v))
      )

      (:action deliver-leaflet
             :parameters
        (?v - postman ?p - leaflet
        ?where - location)
             :precondition
        (and (at ?v ?where)
        (leaflet-in ?p ?v))
             :effect (and (not
        (leaflet-in ?p ?v))
        (leaflet-at ?p ?where))
      )
)
```

Listing A.5: PDDL postman domain.

Appendix B

CD contents

- graphHopper.zip - Java implementation of GraphHopper library that is capable of communicating with SymBA*
- osmToPDDL.zip - A Java program that converts OSM files to PDDL problem file or to translator SAS file or to preprocessor output
- README.txt - Description of the usage of files on the CD
- tasks.py - Changed python code of original SymBA* translator

Appendix C

Bibliography

[1] Álvaro Torralba, Vidal Alcázar, Daniel Borrajo, Peter Kissmann and Stefan Edelkamp. 2014. SymBA*: A symbolic bidirectional A* planner. In International Planning Competition, 105–108.

[2] Álvaro Torralba, Carlos Linares López, Daniel Borrajo: Abstraction Heuristics for Symbolic Bidirectional Search. IJCAI 2016. 3272-3278.

[3] D. Applegate, W. Cook, S. Dash, and M. Mevenkamp. *QSopt - linear programming solver and library.*

[4] David Applegate, Ribert Bixby, Vasek Chvatal, and William Cook. *Concorde TSP solver.* 2006.

[5] Stefan Edelkamp and Stefan Schrödl. *Heuristic Search: theory and applications.* 1st ed. Morgan Kaufmann, July 2011. ISBN: 978-0-12-372512-7.

[6] Kutluhan Erol, Dana S Nau, and VS Subrahmanian. "On the complexity of domain-independent planning". In: *AAAI.* Vol. 1. 1992, pp. 381–386.

[7] Matin Ghaziani. "Improved Positioning By Distance-Based Differential Gps". Appendix A - Calculate Distance Between Two Set Of Coordinate. MA thesis. Middle East Technical University, Sept. 2013.

[8] GraphHopper GmbH. *GraphHopper repository.* `https://github.com/graphhopper/graphhopper`. [Online; accessed 20-December-2017].

[9] GraphHopper GmbH. *Selected Customers.* `https://www.graphhopper.com/`. [Online; accessed 21-December-2017].

[10] P. E. Hart, N. J. Nilsson, and B. Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (July 1968), pp. 100–107. ISSN: 0536-1567. DOI: `10.1109/TSSC.1968.300136`.

[11] Keld Helsgaun. "An Effective Implementation of K-opt Moves for the Lin-Kernighan TSP Heuristic". In: *DATALOGISKE SKRIFTER (Writings on Computer Science)* 109 (2006). Roskilde University.

[12] ICAPS. *International Planning Competition 2014 results*. `https://helios.hud.ac.uk/scommv/IPC-14/resDoc.html`. [Online; accessed 20-December-2017].

[13] Michael Katz and Carmel Domshlak. "Optimal admissible composition of abstraction heuristics". In: *Artificial Intelligence* 174.12-13 (2010), pp. 767–798.

[14] Daniel L. Kovacs. "A Multi-Agent Extension of PDDL 3.1". In: *Proceedings of the 3rd Workshop on the International Planning Competition* (2012). Atibaia, Brazil, 25-29 June 2012, pp. 19-27.

[15] Derek Long. *The Third International Planning Competition*. `https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume20/long03a-html/node37.html`. [Online; accessed 19-December-2017].

[16] MalteHelmert. *Output of the Fast Downward translator*. `http://www.fast-downward.org/TranslatorOutputFormat`. [Online; accessed 18-December-2017].

[17] Gerhard Reinelt. *TSPLIB 95*. `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp95.pdf`. [Online; accessed 25-Ferbruary-2018]. Im Neuenheimer Feld 294, D-69120 Heidelberg: Universität Heidelberg - Institut für Angewandte Mathematik, 1995.

[18] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd ed. Prentice Hall, 2009. ISBN: 978-0-13-604259-4.

[19] Hakan L. S. Younes and Michael L. Littman. "PPDDL1.0: An Extension to PDDL for Expressing Planning Domains with Probabilistic Effects". In: *DATALOGISKE SKRIFTER (Writings on Computer Science)* (2004). Carnegie Mellon University.