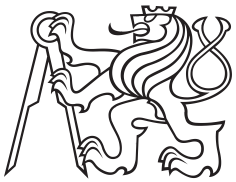
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Classical Planning Problems as Generalized TSP with Precedence

Bc. Petr Bergmann

Supervisor: Ing. Daniel Fišer
Field of study: Open Informatics
Subfield: Software Engineering
May 2020

MASTER'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 Computer Science**

Study program: **Open Informatics**

Specialisation: **Software Engineering**

II. Master's thesis details

Master's thesis title in English:

Classical Planning Problems as Generalized TSP with Precedence

Master's thesis title in Czech:

Problémy klasického plánování jako zobecněný problém obchodního cestujícího s přednostmi

Guidelines:

The goal of the thesis is to identify on the structural level and subsequently propose a solution of a subclass of classical planning problems that can be solved using techniques designed to tackle the Generalized Traveling Salesman Problem with Precedence Constraints (GTSPP).
The student should:
1) Study the literature in the area of classical planning and GTSPP, in particular a basic techniques of the heuristic search and ILP/SAT-based solvers.
2) Identify and formally describe a structure of classical planning problems that can be translated into GTSPP problems in a polynomial time.
3) Propose and implement a solver for the subclass of problems identified in 2).
4) Experimentally evaluate the implementation from 3) and compare the solution to the related state-of-the-art methods from classical planning

Bibliography / sources:

[1] I. Kara, H. Guden, O.N. Koc. 2012. New Formulations for the Generalized Traveling Salesman Problem. In Proc. ASM'12, p.60-65.
[2] K. Castelino, R. D'Souza, P.K. Wright. 2003. Toolpath optimization for minimizing airtime during machining. Journal of Manufacturing Systems, 22(3):173–180, 2003.
[3] N. Ascheuer, M. Jünger, G. Reinelt. 2000. A Branch & Cut Algorithm for the Asymmetric Traveling Salesman Problem with Precedence Constraints. Computational Optimization and Applications, 17(1):61-84, 2000.
[4] R. Huang, Y. Chen, W. Zhang. 2012. SAS+ Planning as Satisfiability. J. Artif. Intell. Res. 43:293-328, 2012.

Name and workplace of master's thesis supervisor:

Ing. Daniel Fišer, Department of Computer Science, FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **14.02.2020** Deadline for master's thesis submission: **22.05.2020**

Assignment valid until: **30.09.2021**

Ing. Daniel Fišer Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgements

I would like to thank my supervisor for his excellent guidance and feedback. I also wish to thank my family for their support throughout 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, 15. May 2020

Abstract

In this thesis, we have identified two classes of STRIPS problems that can be solved as variants of the traveling salesman problem (TSP). For problems from the first class, there exists an encoding into a TSP instance, and problems from the second class can be encoded as an instance of a generalized traveling salesman problem with precedence conditions (PC-GTSP). For both of the classes, we have proved that every solution to their encoding can be transformed into an optimal plan in the original problem and vice versa. Finally, we have implemented the encoding and experimentally evaluated it. The performance of these implementations was compared to several state-of-the-art planning methods of the Fast-Downward planner.

Keywords: Planning, Traveling salesman problem, Precedence constrained generalized traveling salesman problem

Supervisor: Ing. Daniel Fišer

Abstrakt

V této práci jsme identifikovali dvě třídy problémů STRIPS, které se dají vyřešit jako varianty problému obchodního cestujícího (TSP). Problémy z první třídy lze převést na instance TSP a problémy z druhé třídy na instance zobecněného problému obchodního cestujícího s přednostmi (PC-GTSP). Pro obě tyto třídy jsme dokázali, že řešení těchto instancí jde transformovat na optimální plán v původním problému. Stejně tak lze transformovat optimální plán problému na řešení instance TSP, respektivě PC-GTSP. Nakonec jsme převody obou tříd naimplementovali a experimentálně porovnali jejich výkonnost s řadou state-of-the-art plánovacích metod programu Fast-Downward.

Klíčová slova: Plánování, Problém obchodního cestujícího, Zobecněný problém obchodního cestujícího s přednostmi

Překlad názvu: Problémy klasického plánování jako zobecněný problém obchodního cestujícího s přednostmi

Contents

1 Introduction 1

2 Related Works 3

2.1 STRIPS Planning Language 3

2.2 Traveling Salesman Problem 5

2.3 TSP Transformations 6

2.4 TSP ILP Approaches 8

2.5 Non-ILP TSP Approaches 10

3 Background 11

3.1 Graph Background 11

3.2 Traveling Salesman Problem . . . 12

3.3 STRIPS Background 14

4 TSP-Reducible Problem 19

4.1 Problem Definition and Encoding 19

4.1.1 TSP-Reducible Problem Encoding 21

4.1.2 Properties of the Encoding . . 25

4.1.3 The Transformation of the TSP Solution into the Optimal Plan . . 28

4.1.4 Properties of Plans 29

4.1.5 The Transformation of the Optimal Plan into the TSP 31

4.1.6 Conclusion 32

4.2 Visit-All Is TSP-Reducible 33

4.2.1 Properties of the Visit-All problem 33

4.2.2 Proof that Visit-All is TSP-Reducible 35

5 PC-GTSP-Reducible Problem 37

5.1 Problem Definition 37

5.2 Problem Augmentation 39

5.3 Problem Structure 40

5.3.1 Clustering of Transitions 41

5.3.2 Start and End Positions of Transitions 43

5.4 Problem Encoding 44

5.5 Every Solution to the Encoding is an Optimal Plan 46

5.6 Every Optimal Plan Is a Solution to the Encoding 52

5.7 Conclusion 55

6 Experimental Results 57

6.1 Visit-All 58

6.1.1 Implementation 58

6.1.2 Results 59

6.2 OpenStacks 63

6.2.1 LKH-3 Version 64

6.2.2 ILP Version 65

6.3 Conclusion 66

7 Conclusion 69

A Bibliography 71

Bibliography 71

B CD Contents 77

Figures

4.1 An example of projections to three mutex groups in the TSP-reducible problem. The transition system $\mathcal{T}(\Pi, \mathcal{F}_P)$ is the positional projection, others are acyclic projections. The edges between states contain the possible labels of transitions between them. 22

4.2 The encoding of the TSP-reducible problem from Figure 4.1. Most of the edges e in the figure are marked with two items in the format $X; Y$, where X is the cost $c(e)$ and Y is the label $l(e)$. The rest of the edges are incident with v_A and are marked only with their cost. 25

4.3 (a) An example of a shortest path from I_{g_1} to G_{g_1} in the acyclic projection $\mathcal{T}(\Pi, \mathcal{F}_1)$ from Figure 4.1. (b) The splitting of the shortest path from the point (a). 27

5.1 An example of projections in PC-GTSP-reducible problem. If the transition is dependent on $\mathcal{F}_P$, it is drawn thicker. 39

5.2 An example of augmentation of a minimal problem. The operator o, independent on the cyclic projection, was replaced by o_1 and o_2 dependent on the cyclic projection. 40

5.3 Graph with vertices corresponding to transitions between distinct states in acyclic projections from the Figure 5.1. All transitions that are in the same blue rectangle has the same value of the *cluster* function. Every transition has the same index as its operator in the label. The clusters are marked as C_X, where X is the value of the *cluster* function. 41

5.4 Graph containing the clustering from Figure 5.3 with precedence conditions. If an edge leads from cluster C_i to C_j then C_i must be visited before C_j. 43

5.5 An example of the independent cheapest path in the positional projection between states. There exists two ICPs with non-zero costs. One between p_1 and I_P and one between p_2 and I_P. 45

5.6 Encoding $PCGTSP(\Pi)$ of the example problem shown in Figure 5.1. In Figure 5.6a, only edges with costs other than infinite, and edges between distinct clusters are shown. Most of the edges e in the figure are marked with two items in the format $X; Y$, where X is the cost $c(e)$ and Y is the label $l(e)$. The rest of the edges are incident with v_A and are marked only with their cost. In Figure 5.6b, an edge leads from a cluster C_i to a cluster C_j if $C_j \in PC_i$. The solution to this problem encoding is colored blue. 47

5.7 A solution to the PC-GTSP-reducible problem encoding from Figure 5.6. The edges with infinite costs and edges within clusters were intentionally left out. 51

6.1 Run time comparison of different Fast-Downward heuristics with our solver depending on the problem size. Each of the figures shows only successfully solved instances. 60

6.2 The direct comparison of runtimes of Fast-Downward heuristics and our solver. Every point in the graph is a single Visit-All instance. 61

6.3 The runtime dependency of TSP-reducible problem encoding on the size of Visit-All instances. The time is in seconds and plotted in logarithmic scale. The set of instances is taken from IPC 2011 and 2014 satisficing track. 62

6.4 Run time comparison of different Fast-Downward heuristics with our solver depending on the problem size. Each of the figures shows only successfully solved instances. 66

6.5 The scatter plot of the direct comparison of run times of the Fast-Downward planner and our solver. Every point is one problem instance. On the x-axis, there is a runtime of our solver. On the y-axis, there is a runtime of the Fast-Downward planner. 68

Tables

6.1 Comparison of the number of solved Visit-All instances. Our solver is denoted as TSP. 59

6.2 Comparison of the number of solved OpenStacks instances. 65

Chapter 1

Introduction

We recognize two fundamentally different approaches to planning. The first one is domain-specific planning, where we address each problem with representations and techniques tailored specifically for this problem. The second one is domain-independent planning. This is a general approach to planning that is capable of solving different problems. Its input is only a problem specification and knowledge about a domain, and it is using abstract generic representations and models of actions for finding a plan.

Domain-specific planning is an approach that is highly used because it is generally more efficient than using the domain-independent planning for solving the same problem. However, this approach has its drawbacks. Primarily, it is very costly to address every problem separately instead of reusing the same approach. Additionally, some commonalities of various problems may not be taken into account in the domain-specific planning. On the contrary, the domain-independent planning excels at its flexibility and operates on a more abstract level. Therefore, it can address various commonalities much better. However, all of this is at the cost of the lower overall efficiency of solving problems due to the generality of representations that are used.

For the domain-independent planning, it is beneficial if we can find classes of problems that can be planned using some existing domain-specific approach. This combination has benefits of both domain-independent and domain-specific planning. It has the same flexibility as domain-independent planning, and only problems with specific properties are planned domain-specifically. This is addressing the issue where various domain-specific approaches may omit the shared commonalities of their problems while the domain-specificity increases the planning efficiency.

Our goal was to find such classes of problems that can be solved using variants of the traveling salesman problem. The traveling salesman problem (TSP) is a problem of finding the cheapest Hamiltonian cycle in a graph. It is a thoroughly studied problem, and there exist various high-quality solvers for it. Besides the TSP, we have used a TSP variant called a generalized

traveling salesman problem with precedence conditions (PC-GTSP). This problem, on top of the TSP, introduces a clustering of vertices in the graph and partial ordering of these clusters. The solution to the PC-GTSP is the cheapest Hamiltonian path that visits every cluster exactly once and the clusters are visited in the correct order.

In this thesis, we have identified two distinct classes of problems defined in the STRIPS planning language. The first one is called TSP-reducible STRIPS problems, and we will show how to encode it as a TSP instance. Moreover, we will prove that every solution to this encoding can be transformed into an optimal plan of the original problem and vice versa. We will show that an actual problem from the International Planning Competition (IPC) is TSP-reducible.

The second identified class is called PC-GTSP-reducible STRIPS problems. Similarly to the TSP-reducible problems, we will show how to encode it as a PC-GTSP instance, and we will prove that every solution to this encoding is transformable into an optimal plan and vice versa. For both of the identified classes, we will show the relationship between the costs of solutions to their encodings and the costs of their optimal plans.

In the last part, our implementation of the TSP-reducible and the PC-GTSP reducible STRIPS problem encoding will be evaluated on the Visit-All TSP-reducible IPC problem and the OpenStacks PC-GTSP-reducible IPC problem. We will provide experimental results of our implementation and compare it to the state-of-the-art domain-independent planner Fast-Downward.

The thesis is organized as follows. In Chapter 2, a summary of related works is presented. It contains related works about planning and variants of the traveling salesman problem. Afterward, in Chapter 3, we will formally define all of the needed mathematical objects from the graph theory and the STRIPS planning. The definition of TSP-reducible and PC-GTSP-reducible problems, their encoding and related proofs and theorems will be laid out in Chapter 4 and Chapter 5, respectively. These two approaches will be experimentally evaluated and their performance compared to the Fast-Downward planner in Chapter 6. We will conclude our thesis in Chapter 7.

Chapter 2

Related Works

After the introduction, let us give a summary of the works related to our research. We will cover related works from the STRIPS planning, the automated planning, and the traveling salesman problem and its variations. For the traveling salesman problem (TSP), we will discuss its formulations and various heuristic approaches to solving the TSP. On top of that, some of the reformulations between its variants will be discussed.

2.1 STRIPS Planning Language

A planner called Stanford Research Institute Problem Solver (STRIPS) was created by Fikes and Nilsson (1971). This planner was using a new formal language called STRIPS for an input problem specification. This planning language has become popular for its simplicity yet expressiveness and is widely used up to this date. A STRIPS instance is defined by four components - facts, operators, an initial state, and a goal specification. Facts compose states, and we switch between states using the operators. Bylander (1994) showed that the STRIPS state space size and planning complexity is PSPACE-complete. As we are usually unable to recreate the whole search space, the planning of STRIPS problems requires the heuristic search to be even possible. An A* search algorithm is widely used as the search algorithm in the state space. There are various approaches in the current literature on how to create a heuristic function used in the search itself. The ones that are of the highest interest in this thesis are mostly related to abstractions and mutual exclusion state invariants.

Using abstractions is one of the heuristic approaches. The idea behind this approach is that, instead of finding a plan of the original problem, we consider its smaller version. When we create such an abstracted smaller version of the problem, we try to simplify states by dropping some of their distinctions. However, for a good abstraction, it is necessary to minimize alterations of

state transitions. This approach is mostly used in optimal planning. Helmert et al. (2007) proposed the general approach of deriving consistent heuristics. These heuristics were formed using the abstract state spaces created from the original transition system. They have also shown that experimental results of heuristics derived by this approach are comparable with other state-of-the-art heuristics.

Very useful for analyzing planning problems are mutual exclusion state invariants (mutex groups). A mutex group is a set of facts in the STRIPS problem such that at most one of the facts from the mutex group is present in every reachable state.

Mutex groups provide information about the structure of the problem. They are beneficial for various operations in planning. Edelkamp and Helmert (2000) and Helmert (2009) showed that mutex groups are valuable for a translation of planning problems to Finite Domain Representation. As shown by Haslum and Geffner (2000), mutexes are also useful for the creation of heuristic functions. Additionally, the benefits of using mutexes for a state space pruning were shown by Alcázar et al. (2013). However, there is a problem with finding mutex groups. The identification of maximum size mutex groups is a PSPACE-complete (Fišer and Komenda, 2018) problem, just as the complexity of the whole planning is.

The PSPACE-completeness is a drawback for efficient use in planning. It was shown by Fišer and Komenda (2018) that there exists a type of mutex groups called the fact-alternating mutex group (fam-group). This mutex group type depends on the initial state, and the fam-group inference is NP-complete. Additionally, Fišer and Komenda (2018) presented an ILP inference algorithm that is complete with respect to maximal fam-groups. These fam-groups can be used for the preprocessing of STRIPS problems.

There are numerous automated planners, and each of them differs in used types of search, implemented heuristics, and preprocessing methods. An International Planning Competition (IPC) evaluates these state-of-the-art planners on many benchmarking problems. This competition is held every two to four years during the International Conference on Planning and Scheduling (ICAPS). The competition consists of several tracks. We will be interested in the classical track. One of the most successful planners is the planner Fast-Downward (Helmert, 2009, 2006) and its extensions SymBA* (Álvaro Torralba et al., 2014) and Mercury (Katz and Hoffmann, 2014).

In this thesis, the STRIPS problems will be transformed into the traveling salesman problem and its variations. Huang et al. (2012) took a similar approach. In their approach, problems in the STRIPS formalism were encoded into SAT encoding scheme based on SAS+. The encoding was experimentally shown to provide a significant improvement over the state-of-the-art STRIPS encoding schemes.

2.2 Traveling Salesman Problem

The traveling salesman problem (TSP) is a widely studied problem. It is a problem of finding the cheapest Hamiltonian cycle in a graph. The very first formulation of the TSP was given by Dantzig et al. (1954), and they proposed an integer linear programming (ILP) based formalism for solving the problem. Using this ILP method, they have found a solution for an instance of 49 cities spread across 48 US states. The second very important ILP formulation was given by Miller et al. (1960). The importance of this formulation was in its polynomial number of conditions. On the contrary, the representation by Dantzig et al. (1954) has an exponential number of conditions.

These ILP formulations will be discussed more in-depth later on. Now, let us introduce various heuristic-based approximation algorithms that are used for solving the TSP.

The simplest approach is the nearest neighbor greedy algorithm (NN). This algorithm has polynomial $\theta(n^2)$ asymptotic time complexity. However, it gives us no guarantee for the quality of the solution. Moreover, it may not find a solution to the TSP at all, even if the solution exists. The likelihood of finding the solution is increased using a small variation called a repetitive nearest neighbor algorithm (RNN). In this algorithm, the nearest neighbor algorithm is started from all of the nodes in the graph. The best solution among them is selected.

Gutin et al. (2002) analyzed the greedy type heuristics in several experiments. The nearest neighbor, repetitive nearest neighbor, and greedy algorithm were shown to provide poor results on general instances of the symmetric TSP and the asymmetric TSP (ATSP). Additionally, they have shown that in the Euclidean space, the nearest neighbor algorithm provides sufficient quality of the results.

In practice, these greedy-type algorithms are used for finding the initial solution for other heuristic methods. However, even this use of greedy algorithms should be done with caution, as shown by Bang-Jensen et al. (2004). In their study, the greedy algorithm was producing the worst possible solutions for the traveling salesman problem.

Christofides (1976) proposed another widely used algorithm. This algorithm is used for the TSP defined in the metric space, and it provides a guarantee for the quality of the solution. The solution can not be worse than 3/2 of the cost of the optimal solution. The algorithm is based on finding a minimum spanning tree of the graph, a minimum weight-perfect matching on a subgraph induced by a set of vertices of an odd degree, and then finding an Eulerian circuit.

An extensive set of algorithms used for finding a solution of the TSP is

based on iterative improvement. These are local-search based heuristics using reordering of selected vertices in the cycle. The most notable is the 2-opt algorithm proposed by Croes (1958). This algorithm is iteratively finding two pairs of vertices such that the edges in Hamiltonian path cross each other. These vertices are reordered every step of the algorithm, so the cross-over does not happen. The 3-opt algorithm (Lin, 1965) operates in the same way. Only instead of two pairs, three pairs are reordered. These iterative algorithms were generalized by Lin and Kernighan (1973) as Lin–Kernighan heuristic. The Lin–Kernighan heuristic, or k-opt, reorders k different pairs of vertices in the cycle. The Lin–Kernighan heuristic is one of the best heuristics used for the symmetric TSP.

On top of these algorithms, the TSP can be solved by multiple probabilistic techniques. Notably, by genetic algorithms. Genetic algorithms are inspired by biological evolution and mostly depend on three aspects: a selection, an initial population, and a mutation function.

Various interesting works about genetic algorithms are done on the topic of the TSP. Among others, Razali et al. (2011) provides a comparison of various selection strategies used for solving the TSP. The genetic algorithm proposed by Deng et al. (2015) uses the k-means clustering for the generation of the initial population and provides a very significant error improvement compared to more simple methods of the initial population selection. On the topic of mutation, there is a notable article by Liu and Zeng (2009). They have used a reinforced mutation that outperformed various different solvers on TSPLIB instances in both the quality of the solution and the computation time.

On top of the described algorithms and approaches, there are two robust solvers—namely, a solver Concorde (Applegate et al., 2006) and a solver LKH2 (Helsgaun, 2000). The solver Concorde is an award-winning optimal solver for the TSP. This solver was able to find an optimal solution for all of the problems provided in the TSPLIB (Reinelt, 1995). The largest one of them contained more than 85000 vertices (Applegate et al., 2011). In contrast, the LKH2 solver is an approximate solver for solving the TSP. The solver LKH2 is an implementation of Lin-Kernighan heuristic. The heuristic is adaptive, so it is able to decide how many different paths should be swapped each round. In the year 2017, the LKH2 solver was given an extension called LKH3 that is capable of solving more TSP variants. This extension will be discussed more in-depth later on.

2.3 TSP Transformations

Speaking of different TSP variants, Ilavarasi and Joseph (2014) surveyed many different forms of the traveling salesman problem. However, we will focus only on three of them.

The first one is the generalized traveling salesman problem (GTSP). The GTSP contains several pairwise disjoint clusters of vertices. The goal is to find a Hamiltonian cycle going trough exactly one vertex in every cluster. The second one is the precedence constrained traveling salesman problem (PC-TSP), where on top of the normal TSP, we have constraints enforcing precedence of vertices in the Hamiltonian path. The Hamiltonian path is typically used instead of the Hamiltonian cycle because of the precedence. Because we can enforce constraints on top of any TSP variant, a similar problem to PC-TSP and GTSP is precedence constrained generalized traveling salesman problem (PC-GTSP). In the PC-GTSP, constraints are enforcing the precedence of clusters of vertices instead of the vertices themselves.

Even though the TSP is solved in numerous studies, the more complex the variant of the TSP is, the less it is explored. However, in the current literature, there are several proposed transformations between the variants of the TSP we are interested in.

Noon and Bean (1993) proposed a transformation of the GTSP to the ATSP. The main idea of this transformation is to create a zero cost directed cycle in every cluster, so it is assured that every vertex within a cluster is visited before the next cluster. This transformation does not add any new vertices, and it only creates at most $|V|$ new edges, where $|V|$ is the number of vertices of the original graph.

There are multiple transformations from the ATSP to the TSP. The original transformation was proposed by Karp (1972). It splits every vertex into three distinct vertices. In this split, one vertex acts as an input vertex, one as a connection vertex and one as an output vertex. Clearly, this transformation triples the size of the whole instance as the number of vertices of the transformed problem is $3|V|$, where $|V|$ is the number of vertices of the original graph.

Better transformations introduce less than three vertices. The first one was designed by Jonker and Volgenant (1983) and Kumar and Li (1996). These transformations improve the number of vertices added. Each vertex is split in half to an input and an output vertex. Between these two vertices, an edge with a sufficiently high cost M is placed. For an asymmetric instance, these transformations are capable of creating a symmetric TSP instance with size $2|V|$, where $|V|$ is the number of vertices of the original graph.

Additionally, based on the experimental testing conducted by Fischetti et al. (2007), the transformation by Jonker and Volgenant (1983) produces TSP instances that are generally faster to solve than the ones produced by the transformation by Karp (1972).

2.4 TSP ILP Approaches

Having discussed several transformations between some of the TSP variants, we would like to show some ways how the variants we are interested in are solved. First of all, we will discuss ILP approaches and then the non-ILP ones.

Dantzig et al. (1954) proposed the first ILP ATSP representation. This representation uses n^2 binary variables x_{ij} representing edges between vertices and n^2 variables c_{ij} representing the costs of edges between two vertices, where n is the number of vertices. A problem with this formulation is that it uses the exponential number of sub-tour elimination constraints.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} & \\
\text{subject to} \quad & \sum_{i=1}^{n} x_{ij} = 1, & j = 1,\dots,n \\
& \sum_{i=1}^{n} x_{ij} = 1, & i = 1,\dots,n \\
& \sum_{i\in S}\sum_{j\in S} x_{ij} \leq |S| - 1, & S \subset V, S \neq \emptyset \\
& x_{ij} \in \{0,1\}, & i,j = 1,\dots,n
\end{aligned}
$$

This representation is the foundation for all of the following representations. Despite having an exponential number of constraints, several LP relaxations are proposed by transforming some of the constraints. These transformations produce the LP relaxation of problems like a Min-Sum Assignment Problem or an r-SAP (Carpaneto and Toth, 1987; Edmonds, 1967). The asymptotic complexity of these relaxations ranges between $O(n^2)$ and $O(n^3)$.

Miller et al. (1960) introduced the first polynomial representation. This representation substituted sub-tour elimination constraints in Dantzig representation using $O(n^2)$ constraints and introduced $n-1$ new real variables u_i, $i \in \{2,\dots,n\}$, representing the order of cities in the tour.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} & \\
\text{subject to} \quad & \sum_{i=1}^{n} x_{ij} = 1, & j = 1,\dots,n \\
& \sum_{j=1}^{n} x_{ij} = 1, & i = 1,\dots,n \\
& u_i - u_j + (n-1)x_{ij} \leq n-2, & i,j = 2,\dots,n \\
& x_{ij} \in \{0,1\}, & i,j = 1,\dots,n
\end{aligned}
$$

This representation was further improved by Gouveia and Pires (1999, 2001), and it has increased the tightness of the LP relaxation.

Sarin et al. (2005) suggested an ILP ATSP representation with a polynomial number of constraints. Their representation works on a complete graph, and the system of their constraints is as much expressive as the sub-tour elimination constraint proposed by Dantzig. It contains two binary variables x and y. The total number of both variables is n^2. The variable x_{ij} represents whether the city i directly precedes the city j in a tour. The variable y_{ij} means whether the city i precedes the city j, but not necessarily directly. The variables c_{ij} are the same as in the representation proposed by Dantzig et al. (1954).

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} c_{ij} x_{ij} & \\
\text{subject to} \quad & \sum_{i=1, i\neq j}^{n} x_{ij} = 1, & j = 1, \ldots, n \\
& \sum_{j=1, j\neq i}^{n} x_{ij} = 1, & i = 1, \ldots, n \\
& y_{ij} \geq x_{ij}, & i, j = 2, \ldots, n, i \neq j \\
& y_{ij} + y_{ji} = 1, & i, j = 2, \ldots, n, i \neq j \\
& y_{ij} + y_{jk} + y_{ki} \leq 2, & i, j, k = 2, \ldots, n, \\
& & \quad i \neq j \neq k \\
& x_{ij} \in \{0, 1\}, & i, j = 1, \ldots, j \\
& y_{ij} = 0, & i, j = 1, \ldots, n, i \neq j
\end{aligned}
$$

This representation contains $O(n^3)$ constraints and provides a tighter relaxation than Gouveia and Pires (1999). Additionally, Sarin et al. (2005) has presented an extension of this representation for the PC-ATSP. This extension has been accomplished by introducing constraints on variables y_{ij} by setting them 0 if there is a precedence constraint between j and i.

The representation created by Sarin et al. (2005) was a used for the GTSP and the PC-GTSP ILP representation introduced by Salman (2015). Their formulation needs $O(n^2 + 2m^2)$ variables and $O(n^2 + m^3)$ constraints, where n is the number of cities, and m is the number of clusters. This is due to the introduction of new variables for direct movement between clusters and replacing existing y variables by variables describing if one cluster precedes the other in the tour. The largest GTSP instance solved by this formulation had size $n = 755$ and $m = 49$, as reported by Salman (2015).

2.5 Non-ILP TSP Approaches

So far, we have discussed only ILP solutions of TSP variants. In this section, various approaches are shown.

Multi-purpose approximate solver LKH2 (Helsgaun, 2000) is a solver that is able to solve TSP and ATSP instances. This solver is an implementation of the Lin-Kernighan heuristic. This heuristic is a generalization of 2-opt and 3-opt local search algorithms. The heuristic is adaptive, so it is able to decide how many different paths should be swapped each round. In 2017, the original solver was extended (Helsgaun, 2017), so it accepts more than thirty different TSP variants. The PC-ATSP is one of the supported problems. Unfortunately, no description of the solving mechanisms of this LKH-3 solver is provided.

Charikar et al. (1997) studied the constrained version of the TSP. In this work, multiple approximation heuristics are compared. The constrained TSP has shown to be difficult for approximation, as none of the theoretically motivated heuristics has been proven to be better than a simple nearest neighbor greedy algorithm.

Ascheuer et al. (2000) proposed a branch and bound algorithm for solving the PC-ATSP. Their algorithm works on the ILP definition by Dantzig. Optimal solutions to instances ranging between 100 and 200 cities were obtained using this algorithm.

Castelino et al. (2003) examined GTSP with and without precedence constraints. Their work is on the domain of the laser cutting problem that has several simplifications of the GTSP. First of all, all of the distances between vertices are non-zero positive numbers. Additionally, all of the vertices are in the Euclidean space, and thus the distance matrix is symmetric. The more specific transformation procedure from the GTSP to the TSP was provided utilizing these attributes. After this transformation, the precedence constraints were introduced. For solving the TSP, the LKH2 solver was used, and for the constrained version, the heuristics algorithms from Ascheuer et al. (2000) were used. TSP instances of size up to 1000 vertices were solved with difference lower than one percent from optimum and PC-ATSP instances of size up to 200 vertices were solved to their optima using these methods.

Shobaki and Jamal (2015) studied an exact algorithm for minimization of the energy switching problem in compilers. They proposed a branch and bound algorithm. Based on their experimental tests, this algorithm gives better results than ILP formulations. It optimally solves instances with size up to 598 vertices.

Chapter 3

Background

In the previous chapter, we have laid out a summary of works related to our research. In this chapter, we will introduce formal definitions of the discussed problems. First, we will establish definitions of terms from the graph theory and terms related to the traveling salesman problem (TSP) and its variants. Afterward, we will provide definitions related to STRIPS planning.

3.1 Graph Background

The whole graph theory is built upon the definition of a graph. In this thesis, we will use directed graphs. They will be necessary for defining TSP instances transformed from STRIPS problems.

A directed graph is a tuple of a set of vertices and a set of edges, where edges are pairs of vertices. For the graph, we define a cost function that assigns costs to graph edges. Similarly, every edge is assigned a label. This labeling of edges will be used for mapping sequences of operators from the STRIPS problem to edges in the transformed TSP instances.

Definition 3.1 (Directed Graph)**.** A directed graph G is a tuple $G = \langle V, E \rangle$, where V is a finite set of vertices and E is a finite set of edges. Then the edge $e \in E$ is a tuple $e = \langle v, v' \rangle$, where $v, v' \in V$. The costs of edges are defined by a function $c : E \to \mathbb{R}$. An edge $e \in E$ has a label $l(e)$.

An edge $e = \langle v_1, v_2 \rangle \in E$ is incident with the vertex $v \in V$ if either $v_1 = v$ or $v_2 = v$.

A special type of a directed graph is a complete directed graph. This is a directed graph that contains an edge between every pair of its vertices.

Definition 3.2 (Complete Directed Graph)**.** Given a directed graph $G = \langle V, E \rangle$, then G is a complete directed graph if $E = \{\langle v, v' \rangle \mid v, v' \in V, v \neq v'\}$.

If a graph does not contain an edge that starts and ends in the same vertex, then the graph is simple. In our transformations from STRIPS problems into TSP instances, we will encounter only graphs that are complete and simple.

Definition 3.3 (Simple graph)**.** Given a directed graph $G = \langle V, E \rangle$, then G is a simple graph if there is not any edge $e = \langle v, v' \rangle \in E$ such that $v = v'$.

With the graphs defined, let us define various types of sequences of edges in a graph. Sequences of incident edges in the graph are called walks. We distinguish types of walks based on additional conditions that the walk meets. These conditions are enforced on the number of times every vertex, or every edge is visited. A path is a walk that visits every vertex at most once, and a tour is a walk that visits every edge at most once. The formal definition follows.

Definition 3.4 (Walk, path and tour)**.** Given a graph $G = \langle V, E \rangle$, then a walk between two vertices v_1, $v_k \in V$ is a sequence of vertices and edges $p = (v_1, e_1, v_2, e_2, \ldots, e_{k-1}, v_k)$, where $v_i \in V$ and $e_i = \langle v_i, v_{i+1} \rangle \in E$. The cost of the walk is $c(p) = \sum_{i=1}^{k-1} c(e_i)$. The walk is called a path if every vertex is visited at most once. If there are no repeated edges in the walk, it is called a tour.

The cheapest path between two vertices v, v' is a path $p = (v, e_1, \ldots, e_{k-1}, v')$ such that $c(p)$ is minimal.

A special type of walk is a cycle, which is a walk that starts and ends in the same vertex.

Definition 3.5 (Cycle)**.** Given the walk $p = (v_1, e_1, v_2, e_2, \ldots, e_{k-1}, v_k)$ in the graph G. The walk p is a cycle if $v_1 = v_k$ for $k > 1$.

3.2 Traveling Salesman Problem

Now, we can start with definitions of terms related to the traveling salesman problem. In the previous chapter, we have already outlined that the TSP is a problem of finding the cheapest Hamiltonian cycle. A Hamiltonian cycle is a cycle that visits every vertex of a graph exactly once. The definition of the Hamiltonian cycle follows as well as the definition of the closely related Hamiltonian path.

Definition 3.6 (Hamiltonian path)**.** Given a directed graph $G = \langle V, E \rangle$ and a path $p = (v_1, e_1, \ldots, e_{k-1}, v_k)$, then the path p is Hamiltonian if $\{v_1, \ldots, v_k\} = V$ and $k = |V|$.

Definition 3.7 (Hamiltonian cycle)**.** Given a directed graph $G = \langle V, E \rangle$ and a cycle $p = (v_1, e_1, \ldots, e_{k-1}, v_k, e_k, v_1)$, then p is Hamiltonian cycle if $\{v_1, \ldots, v_k\} = V$ and $k = |V|$.

It is obvious that the Hamiltonian cycle is a Hamiltonian path with an

extra edge to the first vertex. Note that the Hamiltonian path in the graph $G = \langle V, E \rangle$ contains exactly $|V| - 1$ edges, and the Hamiltonian cycle contains exactly $|V|$ edges. Karp (1972) proposed the transformation from the problem of finding the Hamiltonian cycle into the problem of finding the Hamiltonian path and vice versa.

Now, we can formally define the traveling salesman problem.

Definition 3.8 (Traveling salesman problem)**.** Given a complete directed graph $G = \langle V, E \rangle$, where V is a set of vertices and E is a set of edges. Then the traveling salesman problem is a problem of finding a Hamiltonian cycle of the graph G such that its cost is minimal.

With the TSP defined, it is possible to show how the other TSP variants change the original definition. If for every pair $v, v' \in V$, the cost function c is defined such that $c(\langle v, v' \rangle) = c(\langle v', v \rangle)$, then the traveling salesman problem is called as symmetric. If this condition is not enforced, the traveling salesman problem is called asymmetric (ATSP).

If we define a partitioning of vertices V into m clusters $C = \{C_1, \ldots, C_m\}$ such that $C_1 \cup \cdots \cup C_m = V$, $C_i \cap C_j = \emptyset$ for every i, j, $i \neq j$, and $C_i \neq \emptyset$ for every $C_i \in C$. The generalized traveling salesman problem (GTSP) is a problem of finding minimum cost cycle that visits exactly one vertex from every cluster $C_i \in C$.

For the precedence constrained traveling salesman problem (PC-TSP), we define precedence constraints of every vertex v_i as $PV_i = \{v_j \mid v_j \in V, v_j$ has to precede v_i in the cycle$\}$. The PC-TSP is a problem of finding the Hamiltonian path that visits every vertex and the precedence conditions are satisfied.

In the generalized version (PC-GTSP), the precedence constraints are defined on a cluster level. Precedence constraints of every cluster C_i are a set $PC_i = \{C_j \mid C_j \in C,$ cluster C_j precedes cluster C_i in the cycle$\}$. The PC-GTSP is a problem of finding the Hamiltonian path that visits exactly one vertex from every cluster $C_i \in C$ and satisfies all of the precedence conditions.

The TSP is the NP-hard problem, and so are the other TSP variants (Karp, 1972). From the definitions, it is apparent that the TSP is a special case of the ATSP. Furthermore, the ATSP is a special case of the GTSP with partitioning $C = \{C_1, \ldots, C_m\}$ such that $m = |V|$ and $|C_i| = 1$ for every $i \in \{1, \ldots, m\}$. Additionally, the precedence constrained version of each problem is a more general version of the original problem. This applies as the original problem is constrained problem with constraints $PV_i = \emptyset$ and $PG_i = \emptyset$ for every $i \in \{1, \ldots, m\}$. Sometimes, the precedence constrained version of ATSP is also called the sequential ordering problem (SOP).

The formal definition of the PC-GTSP is as follows. We will use the

definition of the PC-GTSP using the Hamiltonian path rather than using the Hamiltonian cycle as it is easier to work with.

Definition 3.9 (Precedence constrained generalized traveling salesman problem)**.** Given a complete directed graph $G = \langle V, E \rangle$ with set of vertex clusters C and their precedence conditions $PC_i \subseteq C$ for every $C_i \in C$, and given the cheapest Hamiltonian path $p = \langle v_1, e_1, \ldots, e_m, v_{m+1} \rangle$ in the graph G. Then p is a solution to the PC-GTSP instance defined by G if the following conditions are satisfied:

1. $C = \{C_1, \ldots, C_n\}$, $C_1 \cup \cdots \cup C_n = V$, and $C_i \cap C_j = \emptyset$ for every $i, j \in \{1, \ldots, n\}, i \neq j$.
2. For every cluster $C_i \in C$, there exists exactly one vertex $v \in p$ such that $v \in C_i$.
3. For every two vertices $v_l, v_k \in p$ such that $v_l \in C_i$ and $v_k \in C_j$ it holds that $l < k$ if $C_i \in PC_j$.

3.3 STRIPS Background

Now, let us move from the graph theory to the STRIPS planning. In this section, we will provide definitions related to the STRIPS planning.

We will start with the definition of the STRIPS planning problem.

Definition 3.10 (STRIPS problem)**.** An instance of a STRIPS planning problem Π is a tuple $\Pi = \langle \mathcal{F}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$, where $\mathcal{F}$ is a finite set of facts, $\mathcal{O}$ is a finite set of operators, $\mathcal{I} \subseteq \mathcal{F}$ is the initial state, and $\mathcal{G} \subseteq \mathcal{F}$ is a goal specification.

A state s is a set of facts $s \subseteq \mathcal{F}$. A state s is called a goal state if $\mathcal{G} \subseteq s$.

An operator o is a triple $o = \langle pre(o), add(o), del(o) \rangle$, where:

- $pre(o) \subseteq \mathcal{F}$ is a set of preconditions of the operator o.
- $add(o) \subseteq \mathcal{F}$ is a set of add effects of the operator o.
- $del(o) \subseteq \mathcal{F}$ is a set of delete effects of the operator o.

An operator o is applicable in a state s if $pre(o) \subseteq s$. The resulting state of the operator application is a state $o[s] = (s \setminus del(o)) \cup add(o)$.

In the whole thesis, we assume that every operator o is well-formed, i.e., $add(o) \cap pre(o) = \emptyset$ and $add(o) \cap del(o) = \emptyset$.

An operator sequence $\pi = \langle o_1, o_2, \ldots, o_n \rangle$ is applicable in a state $s_0 \subseteq \mathcal{F}$ if there are states $s_1, \ldots, s_n$ such that $s_i = o_i[s_{i-1}]$ for every $i \in \{1, \ldots, n\}$.

The resulting state of the operator sequence π applied in a state s_0 is denoted as a state $s_n = \pi[s_0]$.

Additionally, if an operator sequence π is empty, then it is applicable in every state s and $s = \pi[s]$.

A plan of a STRIPS problem is an operator sequence π such that $\mathcal{G} \subseteq \pi[\mathcal{I}]$.

A state s is a reachable state if there exists an operator sequence π such that $s = \pi[\mathcal{I}]$.

Multiple operator sequences can be concatenated. This operation preserves the ordering of the operator sequences and all of the operators inside them. The result of the concatenation is also an operator sequence.

Definition 3.11 (Concatenation of operator sequences)**.** Given operator sequences $\pi_1, \ldots, \pi_n$, where $\pi_i = \langle o^i_1, \ldots, o^i_{k_i} \rangle$ for $i \in \{1, \ldots, n\}$. Their concatenation is defined as $seq(\pi_1, \ldots, \pi_n) = \langle o^1_1, \ldots, o^1_{k_1}, o^2_1, \ldots, o^2_{k_2}, \ldots, o^n_1, \ldots, o^n_{k_n} \rangle$.

Some sets of facts possess special properties. Especially, mutex groups are very useful for the STRIPS problem analysis and decomposition. A mutex group is a set of facts such that every reachable state contains at most one of them.

Definition 3.12 (Mutex Group)**.** A set of facts $S \subseteq \mathcal{F}$ is a mutex group if it holds that $|S \cap s| \leq 1$ for every reachable state $s \subseteq \mathcal{F}$.

Fišer and Komenda (2018) examined the mutual exclusion state invariants. They have shown that the inference of mutex groups is PSPACE-complete, i.e., the inference is as hard as deciding whether the STRIPS problem has a plan. Furthermore, they proposed state invariants with lower asymptotic complexity. These invariants are called fact-alternating mutex groups (fam-groups). A fam-group is a set of facts such that the initial state contains at most one of its facts, and every operator adds less or as many of these facts as it deletes. Fišer and Komenda (2018) also proved that the inference of fam-groups is NP-complete and that every fam-group is a mutex group.

Definition 3.13 (Fact-Alternating Mutex Group)**.** A set of facts $S \subseteq \mathcal{F}$ is a fact-alternating mutex group if $|S \cap \mathcal{I}| \leq 1$ and if for every operator $o \in \mathcal{O}$ it holds that $|S \cap add(o)| \leq |S \cap pre(o) \cap del(o)|$.

From the state invariants, let us move to transition systems and projections.

A transition system is a structure containing a set of states and labeled transitions between them. Exactly one of the states is the initial state and some of the states are goal states. Transition systems are structures used in various fields. In this thesis, they will be used for modeling transitions between states in the state space of the STRIPS problem.

Definition 3.14 (Transition system)**.** A transition system $\mathcal{T}$ is a tuple $\mathcal{T} = \langle S, L, T, I, G \rangle$. The set S defines states of the system, L is a set of labels and

T is a transition relation $T \subseteq S \times L \times S$. I is an initial state $I \in S$ and G is a set of goal states $G \subseteq S$. Each label is given a cost by a function $c : L \to \mathbb{R}$.

If a transition $t = (s, o, s')$ then we call the transition t as a transition between states s and s'. The transition t starts in the state s and ends in the state s'. Additionally, we call the state s as a start state of t, and the state s' as an end state of t.

Two transition systems with the same set of labels can be further composed to form a new transition system. This composition is done using a synchronized product.

Definition 3.15 (Synchronized product)**.** Given two transition systems $\mathcal{T}_1 = \langle S_1, L, T_1, I_1, G_1 \rangle$ and $\mathcal{T}_2 = \langle S_2, L, T_2, I_2, G_2 \rangle$, the synchronized product $\mathcal{T}_1 \otimes \mathcal{T}_2 = \mathcal{T}$, where $\mathcal{T}$ is a transition system $\mathcal{T} = \langle S, L, T, I, G \rangle$ such that $S = S_1 \times S_2$, $\mathcal{T} = \{((s_1, s_2), l, (s'_1, s'_2)) \mid (s_1, l, s'_1) \in T_1, (s_2, l, s'_2) \in T_2\}$, $I = I_1 \times I_2$ and $G = G_1 \times G_2$.

For every transition system, we distinguish whether the transition system is cyclic or acyclic. This property is similar to the cyclic and acyclic graphs. In the cyclic graph, there exists a vertex and a nonempty path that both starts and finishes in this vertex. The transition system is cyclic if there exist a state s and a nonempty sequence of transitions between states such that s is the start state of the first transition in this sequence, and s is the end state of the last transition in this sequence. If no such state exists, then the transition system is acyclic. These properties are formally defined as follows.

Definition 3.16 (Cyclic and acyclic transition system)**.** Given a transition system $\mathcal{T} = \langle S, L, T, I, G \rangle$, $\mathcal{T}$ is called cyclic if there exists a state $s_0 \in S$, a sequence of states $(s_0, s_1, \ldots, s_n) \in S^n$, where $s_n = s_0$, and a sequence of labels $(l_0, \ldots, l_{n-1}) \in L^n$ such that $(s_i, l_i, s_{i+1}) \in T$ and $s_i \neq s_{i+1}$ for all $i \in \{0, \ldots, n-1\}$. Otherwise, we say that the transition system is acyclic.

The cyclicity and the acyclicity of transition systems will be used to define the form of our STRIPS problems. Most of the transition systems that will be described will need to be acyclic. Their acyclicity will be a sufficient condition for the STRIPS problem to be solvable as an instance of a TSP variant.

For every transition system, it is possible to create its abstractions. Abstractions are transition systems that omit some of the properties of its original transition system. The abstraction usually has a lower number of states and the mapping between states of the original transition system to the abstraction is defined using abstraction function. This function must be defined such that it preserves transitions between the original states and their abstracted states.

Definition 3.17 (Abstraction and abstraction function)**.** Given two transition systems $\mathcal{T}_1 = \langle S_1, L, T_1, I_1, G_1 \rangle$ and $\mathcal{T}_2 = \langle S_2, L, T_2, I_2, G_2 \rangle$ and function $\alpha : S_1 \to S_2$, we denote $\mathcal{T}_2$ as an abstraction of $\mathcal{T}_1$ with an abstraction function α if, for every transition $(s_a, l, s_b) \in T_1$, it holds that $(\alpha(s_a), l, \alpha(s_b)) \in T_2$,

$I_2 = \{\alpha(s) \mid s \in I_1\}$ and $G_2 = \{\alpha(s) \mid s \in G_1\}$.

In the following chapters, abstractions will be used for creating projections to sets of facts of the STRIPS problem.

So far, the definitions considered the general transition system. The following part will specifically focus on the transition systems related to STRIPS planning problems.

The first definition specifies the transition system of a STRIPS problem. This transition system has operators as labels and states are sets of facts. The initial state is the same as the initial state of the STRIPS problem. Every state that contains all of goal facts is considered as a goal state. A transition labeled by the operator o leading from state s to s' if $o[s] = s'$.

Definition 3.18 (Transition system of a STRIPS problem)**.** Given a STRIPS problem $\Pi = \langle \mathcal{F}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$, the transition system of the problem Π is the transition system $\mathcal{T}(\Pi) = \langle S, L, T, I, G \rangle$ such that $S = 2^{\mathcal{F}}$, $L = \mathcal{O}$, $I = \mathcal{I}$, $G = \{s \in S \mid \mathcal{G} \subseteq s\}$, and for all states $s_i \in S$, it holds that $(s_i, o, s_j) \in T$ iff $o[s_i] = s_j$.

If a transition t is labeled by an operator o then we call the operator o as an operator of the transition t.

A special type of abstraction is a projection. A projection to a set of facts is an abstraction of the original transition system with states containing only facts from that set. Projections have a specific form of their abstraction function. It is defined by an intersection with the set of facts we are projecting into.

Definition 3.19 (Projection)**.** Given a STRIPS problem $\Pi = \langle \mathcal{F}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ and its transition system $\mathcal{T}(\Pi)$, a projection to the set of facts $\mathcal{F}' \subseteq \mathcal{F}$ is an abstraction $\mathcal{T}(\Pi, \mathcal{F}')$ of $\mathcal{T}(\Pi)$ with an abstraction function $\alpha(s) = s \cap \mathcal{F}'$.

A transition system of a STRIPS problem will be used as a primary tool to define the desired structure of the problems that can be solved using the TSP variants. The structure will be defined using projections of this transition system to mutex groups. Additionally, all of the problem properties will be defined using these projections.

In the same way, as we define a path in the graph, we can define a path in a transition system of a STRIPS problem. Informally, we can consider states as vertices and transitions as edges. A more formal definition is given next.

Definition 3.20 (Path in transition system of STRIPS problem)**.** Given a transition system $\mathcal{T}(\Pi) = \langle S, \mathcal{O}, T, I, G \rangle$, the path in $\mathcal{T}$ from $s_0 \in S$ to $s_n \in S$ is a sequence of transitions $p = \langle t_1, \ldots, t_n \rangle$ such that $t_i = (s_{i-1}, o_i, s_i) \in T$ for $i \in \{1, \ldots, n\}$. We say that the path is the cheapest if the cost $c(p) = \sum_{i=1}^{n} c(t_i)$ is minimal. The cheapest path from s_0 to s_n is denoted by $SP(s_0, s_n)$. The operators of the path p is the operator sequence $\langle o_1, \ldots, o_n \rangle$.

In the transition system of a STRIPS problem, we recognize dependency and independency of operators and transitions. We call an operator as an operator dependent on a set of facts if it contains a fact from this set either in its preconditions, add effects, or delete effects. A transition is dependent on a set of facts if it is labeled by the operator that is dependent on this set.

Additionally, we recognize operators partially dependent on a set of facts. These operators contain a fact from the set only in their precondition.

The transition in a transition system $\mathcal{T}(\Pi, \mathcal{F}')$ is independent if there is no set of facts disjoint with $\mathcal{F}'$ that the transition would be dependent on.

Definition 3.21 (Dependent and independent operator and transition)**.** Given a STRIPS problem $\Pi = \langle \mathcal{F}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$, we call an operator $o \in \mathcal{O}$ as dependent on a subset of facts $F' \subseteq \mathcal{F}$ if $add(o) \cap F' \neq \emptyset$ or $pre(o) \cap F' \neq \emptyset$ or $del(o) \cap F' \neq \emptyset$. The operator o is partially dependent on F' if $add(o) \cap F' = del(o) \cap F' = \emptyset$ and $pre(o) \cap F' \neq \emptyset$.
Given a projection $\mathcal{T}(\Pi, \mathcal{F}_1) = \langle S_1, \mathcal{O}, T_1, I_1, G_1 \rangle$, a transition $t = (s, o, s') \in T_1, s \neq s'$, and a subset of facts $\mathcal{F}_2 \subseteq \mathcal{F}, \mathcal{F}_1 \cap \mathcal{F}_2 = \emptyset$, then the transition t is (partially) dependent on $\mathcal{F}_2$ if the operator o is (partially) dependent on $\mathcal{F}_2$. If no such set of facts $\mathcal{F}_2$ exists, we denote the transition t as an independent transition.

Lemma 3.22. *Every operator $o \in \mathcal{O}$ partially dependent on $\mathcal{F}_2 \subseteq \mathcal{F}$ is dependent on $\mathcal{F}_2$.*

Proof. The lemma obviously holds according to Definition 3.21. □

We have shown all of the definitions needed for the next chapters. The background of the graph theory and the STRIPS planning was covered. Additionally, we have shown multiple variants of the TSP that will be discussed in the rest of the thesis. In the next chapter, we will show the first set of STRIPS problems called TSP-reducible problems. We will prove that these problems can be solved using the TSP.

Chapter 4

TSP-Reducible Problem

In the previous chapter, we have covered the graph theory and STRIPS background. In this chapter, we will define a TSP-reducible STRIPS problem. We will show that every TSP-reducible problem can be encoded into a TSP instance, which is a graph. Additionally, the solution to this TSP instance can be transformed into an optimal plan and vice versa.

An outline of the chapter is following. First of all, we will define the TSP-reducible STRIPS problem. Afterward, we will define the encoding for the TSP-reducible problem into the TSP instance. We will show that a solution to the TSP instance can be transformed into an optimal plan in the original STRIPS problem. We will also prove the transformation correctness. A similar transformation will be shown in the other direction as well. Last but not least, we will prove that an IPC problem called Visit-All is TSP-reducible.

4.1 Problem Definition and Encoding

A TSP-reducible problem is a STRIPS problem with several properties. Its set of facts can be split into pairwise disjoint mutex groups. One of these mutex groups, denoted as $\mathcal{F}_P$, is called positional, and the rest of them are called acyclic. The positional mutex group does not contain any goal facts. On the contrary, every acyclic mutex group contains exactly one goal fact.

The projection to an acyclic mutex group is an acyclic transition system and is called acyclic projection. The acyclic projection contains exactly one pair of states such that transitions between them are dependent on the positional mutex group and have the same add effects. These two states are called central states. Additionally, a transition between the central states must be in every path from the initial to the goal state in the acyclic projection. Every other transition between two distinct states in the acyclic projection is independent.

The projection to the positional mutex group is called positional projection. Operators of transitions leading to the same state have the same add effects.

The way the TSP-reducible problem is defined creates an important property. The transitions in the positional projection only affect the transition between central states in the acyclic projections. This property ensures that we can freely change between states in the positional projection while enabling the transition between central states in every acyclic projection.

The idea behind solving the TSP-reducible problem is as follows. All of the goal facts are in the acyclic mutex groups, and to reach them, we need to enable the transition between the central states. As this transition is dependent on the positional mutex group, it is necessary to find a path in the positional projection such that this transition is enabled.

As a result, solving the TSP-reducible problem can be split into two tasks. The first one is fulfilling the path between initial state and goal state in every acyclic projection. The second one is finding a path in the positional projection such that the transition between central states in every acyclic projection is enabled. These two tasks are interwoven in the whole section. Some of the lemmas that we will prove might be related only to one of the tasks. Therefore, whenever we will relate to these tasks we will call them as Acyclic task and Positional task, respectively.

Now, we have informally described the TSP-reducible problem, the formal definition of the TSP-reducible problem follows.

Definition 4.1 (TSP-reducible problem)**.** A STRIPS problem $\Pi = \langle \mathcal{F}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ is called as TSP-reducible problem if all of the following holds:

(i) There exists a set of mutex groups $\{\mathcal{F}_P, \mathcal{F}_1, \ldots, \mathcal{F}_k\}$ such that $\mathcal{F}_i \cap \mathcal{F}_j = \emptyset$ for $i, j \in \{P, 1, \ldots, k\}$, $i \neq j$, and $\mathcal{F} = \mathcal{F}_P \cup \mathcal{F}_1 \cup \cdots \cup \mathcal{F}_k$. Additionally, the mutex group $\mathcal{F}_P$ is also a fam-group.

(ii) The projection $\mathcal{T}(\Pi, \mathcal{F}_P) = \langle S_P, \mathcal{O}, T_P, I_P, G_P \rangle$ of a fam-group $\mathcal{F}_P$ is cyclic and the rest of the projections to mutex groups are acyclic and form a set of acyclic projections $Acyclic_\Pi = \{\mathcal{T}(\Pi, \mathcal{F}_1), \ldots, \mathcal{T}(\Pi, \mathcal{F}_k)\}$, where each $\mathcal{T}(\Pi, \mathcal{F}_i) = \langle S_{g_i}, \mathcal{O}, T_{g_i}, I_{g_i}, G_{g_i} \rangle$.

(iii) For every $\mathcal{T}(\Pi, \mathcal{F}_i) \in Acyclic_\Pi$, it holds that $|\mathcal{F}_i \cap \mathcal{G}| = |G_{g_i}| = 1$. For the set of facts $\mathcal{F}_P$, it holds that $\mathcal{F}_P \cap \mathcal{G} = \emptyset$.

(iv) There is no transition $t = (s, o, s') \in T_P$, where $s \neq s'$, partially dependent on acyclic mutex group.

(v) For every state $s \in S_P$, and every two operators $o, o' \in \mathcal{O}$ such that $(s', o, s), (s'', o', s) \in T_P$ where $s \neq s'$ and $s \neq s''$, it holds that $add(o) = add(o')$.

(vi) There exists an injective function $\gamma : Acyclic_\Pi \to \mathcal{F}_P$. For every projection $\mathcal{T}(\Pi, \mathcal{F}_i)$, there exist exactly two states $s, s' \in S_{g_i}, s \neq s'$ (called

central states) such that every transition $t = (s, o, s') \in T_{g_i}$ is dependent on $\mathcal{F}_P$ and $\gamma(\mathcal{T}(\Pi, \mathcal{F}_i)) \in add(o)$. Every other transition between two different states in S_{g_i} is independent. Additionally, it holds that every path from I_{g_i} to G_{g_i} contains a transition between s and s'.

As we stated earlier, the positional mutex group is denoted as $\mathcal{F}_P$ and the positional projection as $\mathcal{T}(\Pi, \mathcal{F}_P)$. Additionally, the acyclic mutex group will be always denoted as $\mathcal{F}_i$ with index $i \in \{1, \ldots, k\}$ and the acyclic projection to this mutex group will be denoted as $\mathcal{T}(\Pi, \mathcal{F}_i)$.

We call the facts from the positional mutex group as positional facts. An important observation is that, as all of the used projections are projections to mutex groups, every reachable state in these projections contains at most one fact.

From the definition, it is possible to make the following observation. For the TSP-reducible problem Π, it holds that $\mathcal{T}(\Pi, \mathcal{F}_P) \otimes \mathcal{T}(\Pi, \mathcal{F}_1) \otimes \cdots \otimes \mathcal{T}(\Pi, \mathcal{F}_k) = \mathcal{T}(\Pi)$. This statement holds because $\mathcal{F} = \mathcal{F}_P \cup \mathcal{F}_1 \cup \cdots \cup \mathcal{F}_k$ and the mutex groups are pairwise disjoint (Helmert et al., 2007).

An example of a TSP-reducible problem is shown in Figure 4.1. The problem contains three mutex groups and the figure contains the projections to them. The figure contains a positional projection $\mathcal{T}(\Pi, \mathcal{F}_P)$ and two acyclic projections $\mathcal{T}(\Pi, \mathcal{F}_1)$ and $\mathcal{T}(\Pi, \mathcal{F}_2)$. The positional projection models three positions. The position a and b are enabling positions of operators o_a and o_b, respectively. The positional projection contains an initial state I_P and does not contain any state with a goal fact.

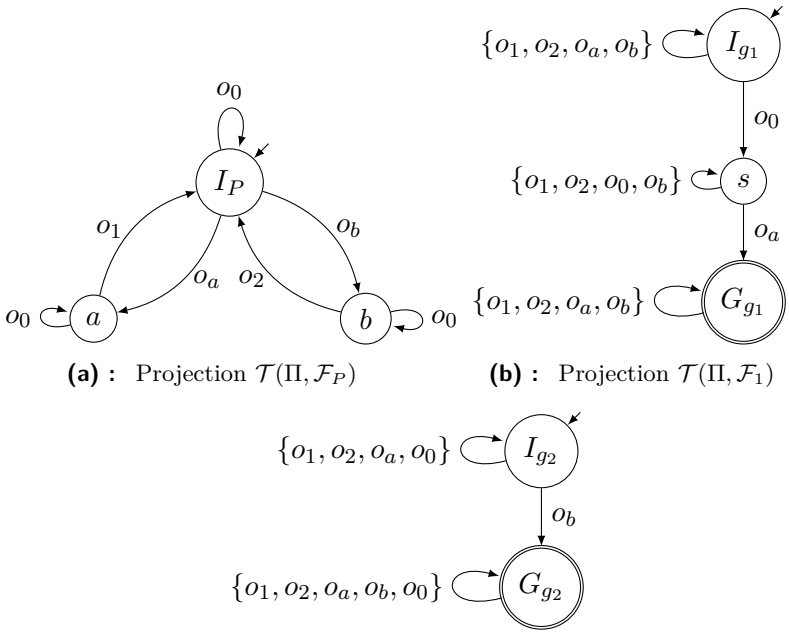
In the acyclic projection $\mathcal{T}(\Pi, \mathcal{F}_1)$, the central states are s and G_{g_1}. The transition between different states that is dependent on the positional mutex group is (s, o_a, G_{g_1}). The operator o_0 is dependent only on $\mathcal{F}_1$.

The second acyclic projection contains the central states I_{g_2} and G_{g_2}. The transition dependent on the positional mutex group is (I_{g_2}, o_b, G_{g_2}).

This example problem will be used as a running example. For the simplicity of the notation in these examples, we will assume that every state in the projections contains the fact of the same name, e.g., the state a in the positional projection is the state $\{a\}$.

4.1.1 TSP-Reducible Problem Encoding

With the TSP-reducible problem specified, let us show its encoding. We will encode the TSP-reducible problem as a traveling salesman problem instance, which is a graph. The solution to this TSP instance can be converted back into an optimal plan of the original TSP-reducible problem and vice versa.



(a) : Projection $\mathcal{T}(\Pi, \mathcal{F}_P)$ **(b) :** Projection $\mathcal{T}(\Pi, \mathcal{F}_1)$

(c) : Projection $\mathcal{T}(\Pi, \mathcal{F}_2)$

Figure 4.1: An example of projections to three mutex groups in the TSP-reducible problem. The transition system $\mathcal{T}(\Pi, \mathcal{F}_P)$ is the positional projection, others are acyclic projections. The edges between states contain the possible labels of transitions between them.

First of all, we will outline how the encoding works. Then we formally define the encoding itself. Afterward, we will prove that our assumptions about the transformation of a solution into an encoding to an optimal plan were correct.

We have already outlined that solving the TSP-reducible problem can be split into two tasks. The Positional task is finding a path in the positional projection such that the transition between central states in every acyclic projection is enabled. The encoding exactly copies this idea.

To create an encoding of the TSP-reducible problem, we need to know which positional facts are needed to enable these transitions between central states. According to Definition 4.1, these positional facts are specified by the function γ. This function assigns the positional fact to every acyclic projection. And the transition between central states of the acyclic projection has this positional fact in the add effect.

The encoding is a TSP instance defined as a complete graph with these positional facts as vertices. For the simplicity of notation, from now on, we will refer to a positional fact as of a vertex and vice versa.

Recall that every reachable state in the positional projection contains exactly one fact. The edge between two facts f and f' has a label consisting

of the operators of the cheapest path between $\{f\}$ and $\{f'\}$ in the positional projection. The cost of the edge is equal to the cost of this path. Informally, every label of the edge from f to f' describes how to reach the state containing f' from state containing f, and the cost of this edge is equal to the cost of the needed operators.

Additionally, there is a vertex I_P for the positional fact from the initial state and a vertex v_A whose function will be described later on.

Definition 4.2 (TSP encoding of TSP-reducible problem)**.** Given the TSP-reducible problem $\Pi = \langle \mathcal{F}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ with a set of acyclic projections $Acyclic_\Pi$ and a function gamma $\gamma : Acyclic_\Pi \to \mathcal{F}_P$, let $Enabling\text{-}positions_\Pi = \{\gamma(\mathcal{T}) \mid \mathcal{T} \in Acyclic_\Pi\}$. The TSP encoding $TSP(\Pi) = \langle V, E \rangle$ is a complete graph with the cost function $c : E \to \mathbb{R}$ and labeling function l where:

- $V = Enabling\text{-}positions_\Pi \cup \{I_P\} \cup \{v_A\}$
- $E = \{\langle p, p' \rangle \mid p, p' \in V, p \neq p'\}$
- For every edge $e = \langle p, p' \rangle$, $p, p' \in V \setminus \{v_A\}$, the label $l(e)$ consists of the operators of the cheapest path between states $\{p\}$ and $\{p'\}$ in $\mathcal{T}(\Pi, \mathcal{F}_P)$ and $c(\langle p, p' \rangle) = c(l(\langle p, p' \rangle))$. If no such path exists, let $c(\langle p, p' \rangle) = \infty$. For other cases, $c(\langle v_A, I_P \rangle) = 0$ and for $v \in Enabling\text{-}positions_\Pi$ it holds that $c(\langle v_A, v \rangle) = c(\langle v, v_A \rangle) = |V| \max\{c(\langle v', v'' \rangle) \mid v', v'' \in V \setminus \{v_A\}, v' \neq v''\} + 1 = M$.

A solution to the encoding $TSP(\Pi)$ is the cheapest Hamiltonian cycle in the graph $TSP(\Pi)$, such that no edge e in a cycle has the cost $c(e) = \infty$.

From now on, we may sometimes call the TSP-reducible problem encoding simply as the encoding.

The solution to this TSP instance is a cycle that visits every positional fact needed to enable the transition between some central states. The labels of the edges in this cycle are operator sequences that were used for transitions in the positional projection.

Recall that the finding of the plan of TSP-reducible problem could be split into two tasks. The labels of edges in the solution to the TSP-reducible creates the path in the positional projection. This path enables the transition between central states in every acyclic projection. Therefore, it corresponds to the Positional task.

The Acyclic task is to fulfill the paths between initial state and goal state in every acyclic projection. Based on Definition 4.1, every such path in an acyclic projection contains only one transition dependent on the positional mutex group. All of the other transitions are independent.

As a result, the operators of transitions before the transition between central states can be applied anywhere in the plan before the enabling of the

transition between central states. Similarly, the operators of transitions after the transition between the central states can be applied anywhere in the plan after the enabling of the transition between central states.

In the STRIPS plan, we do not have to visit the initial state once again, when we have reached the goal state. If we did not require the solution to be a cycle, we would obtain the cheapest Hamiltonian path problem. However, since we want to solve the TSP, the vertex v_A is used for reduction from the cheapest Hamiltonian path problem to the cheapest Hamiltonian cycle problem, as proposed by Karp (1972).

This vertex v_A has the same cost of all the incident edges, except the edge $\langle v_A, I_P \rangle$. This cost is defined to be a constant M that is larger than $|V|$ multiplied by the maximum cost of the edge in the graph. Recall that the Hamiltonian cycle in a graph $\langle V, E \rangle$ has exactly $|V|$ edges. As a result, M is higher than the cost of any possible Hamiltonian cycle in the graph and cannot alter the Hamiltonian path. This leads us to the following property of all solutions to $TSP(\Pi)$.

Lemma 4.3. *Every solution to $TSP(\Pi)$ contains the edge $\langle v_A, I_P \rangle$.*

Proof by contradiction. Let us assume that the cheapest Hamiltonian cycle p is a solution to $TSP(\Pi)$ that does not contain the edge $\langle v_A, I_P \rangle$. For the cost of this cycle, it holds that $c(p) \geq 2M$ because every edge incident with v_A other than $\langle v_A, I_P \rangle$ has the cost $c(e) = M$.

As $TSP(\Pi)$ is a complete graph, we may construct a different Hamiltonian cycle p' that contains the edge $\langle v_A, I_P \rangle$. The cycle p' contains only one edge with a cost M. It holds that every Hamiltonian cycle contains exactly $|V|$ edges and $M = |V| \max\{c(\langle v', v'' \rangle) \mid v', v'' \in V \setminus \{v_A\}\} + 1$. Therefore, the cycle p' has the cost $c(p') < 2M \leq c(p)$ because $c(\langle v_A, I_P \rangle) < M$. This is the contradiction because the cycle p is not the cheapest. □

An example of the encoding of the problem from Figure 4.1 is given in Figure 4.2. Recall that the positions a and b are enabling positions. The encoding $TSP(\Pi) = \langle V, E \rangle$ has $V = \{I_P, v_A, a, b\}$. Most of the edges e in the figure are marked with two items in the format $X; Y$, where X is the cost $c(e)$ and Y is the label $l(e)$. The rest of the edges are incident with v_A and are marked only with their cost. For example, the edge $e = \langle a, b \rangle$ has the label $l(e) = \langle o_1, o_b \rangle$, because the shortest path between a and b in the positional projection is $\langle (a, o_1, I_P), (I_P, o_b, b) \rangle$.

The label is not present for edges incident with the vertex v_A. The constant M is for this example defined as $M = 9$ because $|V| = 4$ and the maximal cost of the edge in the encoding is equal to 2.

An example of a solution to this encoding is a path that is marked red and has a cost $c = 1 + 2 + M + 0 = 3 + M$.

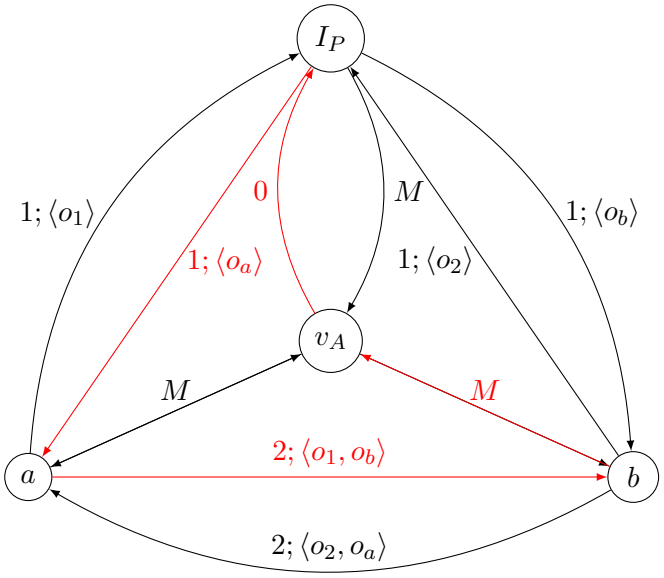


Figure 4.2: The encoding of the TSP-reducible problem from Figure 4.1. Most of the edges e in the figure are marked with two items in the format $X; Y$, where X is the cost $c(e)$ and Y is the label $l(e)$. The rest of the edges are incident with v_A and are marked only with their cost.

4.1.2 Properties of the Encoding

In the previous subsection, we have formulated the idea behind the TSP-reducible problem encoding. In this subsection, we will prove that all of our assumptions were correct. Recall that we have distinguished two tasks that are necessary for finding a plan of TSP-reducible problem.

The first lemma proven focuses on the Positional task. It states that the label of an edge from f to f' is an operator sequence applicable to any state containing the fact f. Additionally, the result of this application always contains the fact f'.

The solution of the encoding is a Hamiltonian cycle that visits every position enabling the transition between central states in acyclic projections. What we want to do in the transformation from the solution to the optimal plan is that we concatenate all of the labels of the consequent edges in the solution. This concatenation will be the core of the optimal plan that ensures the enabling of transitions between central states.

For this reason, it is necessary, that for every two consequent edges e, e' between positional facts such that $e = \langle v, v' \rangle$ and $e' = \langle v', v'' \rangle$ it holds that $l(e')$ is applicable in $l(e)[s]$ for a state s, $v \in s$.

Lemma 4.4. *Let e be an edge $e = \langle v, v' \rangle \in E$, where $v, v' \in V \setminus \{v_A\}$, and its label is $l(e) = \langle o_1, \ldots, o_n \rangle$. Then it holds $v' \in l(e)[s]$ for every $s \subseteq \mathcal{F}$ such that $v \in s$.*

Proof. Based on Definition 4.1, for every transition $(s_1, o, s_2) \in T_P$, $s_1 \neq s_2$, it holds that $pre(o) \subseteq \mathcal{F}_P$. Therefore, $s_2 \subseteq o[s'_1]$ for every $s'_1 \subseteq \mathcal{F}$ such that $s_1 \subseteq s'_1$.

Let $e = \langle v, v' \rangle$ be an edge in the encoding. The label of the edge $l(e) = \langle o_1, \ldots, o_n \rangle$ is the cheapest path from $\{v\}$ to $\{v'\}$ in the positional projection. Therefore, it must hold $v' \in l(e)[s]$ for every $s \subseteq \mathcal{F}$ such that $v \in s$. □

It is possible to see this property even in our example problem. For example, the edge $e = \langle I_P, a \rangle$ has a label $l(e) = \langle o_a \rangle$. From the positional projection, it is clearly visible that $b \in o_a[s]$ for every state s such that $a \in s$.

The second lemma concerns the Positional task as well. It shows that the edge leading to a positional fact $f \in \mathit{Enabling\text{-}positions}_\Pi$ really enables the transition between central states in an acyclic projection $\mathcal{T}(\Pi, \mathcal{F}_i)$ such that $\gamma(\mathcal{T}(\Pi, \mathcal{F}_i)) = v$. Obviously, if this property was not true, the whole idea behind the TSP-reducible problem encoding would be incorrect.

This lemma will be used later on to prove that transformed solution into the optimal plan made the transition between central states.

Lemma 4.5. *Let $v, v' \in V \setminus \{v_A\}$, $l(\langle v, v' \rangle) = \langle o_1, \ldots, o_n \rangle$ and $\gamma(\mathcal{T}(\Pi, \mathcal{F}_i)) = v'$. Then for some states $s, s' \in S_{g_i}$, $s \neq s'$, there exists a transition $(s, o_n, s') \in T_{g_i}$ dependent on the positional mutex group.*

Proof. From condition (v) of Definition 4.1, it follows that every transition leading to a given state $s \in S_P$ has the same add effect. The label $l(\langle v, v' \rangle)$ are the operators of the cheapest path in the positional projection between $\{v\}$ and $\{v'\}$. Let $(\{v''\}, o, \{v'\})$ denote the last transition of this path. Obviously, the operator o is dependent on the positional mutex group and $v' \in \mathit{add}(o)$. Because the function γ from condition (vi) of Definition 4.1 is the injective function and $\gamma(\mathcal{T}(\Pi, \mathcal{F}_i)) = v'$, this operator is dependent on $\mathcal{F}_i$. □

In contrast to the previous two lemmas, the next lemma is focused on the Acyclic task. Every plan must be a path from the initial state to the goal state in every acyclic projection. That is because every acyclic projection is a projection to a set of facts that contains exactly one goal fact.

We have already outlined a part of the transformation from the solution to the TSP encoding into the optimal path. The operators of transitions in the positional projections are reconstructed from the labels of edges in the solution to the encoding.

The second part of the transformation concerns the Acyclic task. We will split the cheapest path from the initial state to the goal state in every acyclic projection around the transition between central states.

The operators of transitions before will be placed in the plan before the transition between central states is enabled. This will ensure that the transition between central states can be done.

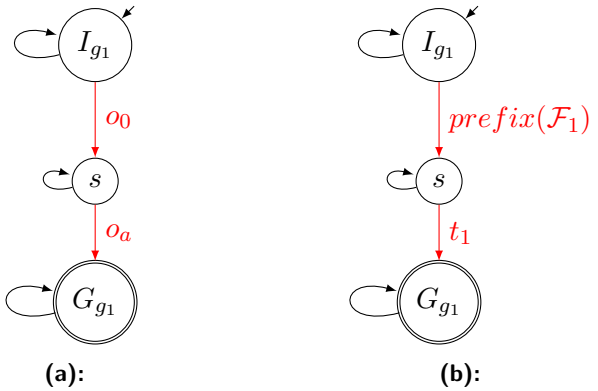


Figure 4.3: (a) An example of a shortest path from I_{g_1} to G_{g_1} in the acyclic projection $\mathcal{T}(\Pi, \mathcal{F}_1)$ from Figure 4.1.
(b) The splitting of the shortest path from the point (a).

The operators of transitions after will be placed in the plan after enabling the transition between central states.

The resulting plan will be a path from the initial state to the goal state in every acyclic projection.

In this lemma, it is shown that every cheapest path in an acyclic projection can be split into these three parts. Additionally, it is shown that there is exactly one way how the cheapest path can be split.

Lemma 4.6. *Let $\mathcal{T}(\Pi, \mathcal{F}_i)$ be an acyclic projection and π be the cheapest path from I_{g_i} to G_{g_i} such that $\pi = \langle t_1, \ldots, t_n \rangle$. Then there exist $prefix(\mathcal{F}_i) = \langle t_1, \ldots, t_{k-1} \rangle$ and $suffix(\mathcal{F}_i) = \langle t_{k+1}, \ldots, t_n \rangle$ such that every $t \in prefix(\mathcal{F}_i)$ and every $t' \in suffix(\mathcal{F}_i)$ are transitions independent on the positional mutex group, and t_k is dependent on the positional mutex group. Additionally, this splitting is unique.*

Proof. From Definition 4.1, it holds that every path from I_{g_i} to G_{g_i} contains a transition between central states that is dependent on the positional mutex group. This transition is t_k. As the path π is the cheapest, it does not contain any transition $(s, o, s) \in T_{g_i}$. This transition would be redundant and would only increase the cost of the path. Therefore, based on the definition of the TSP-reducible problem, every other transition in π has to be independent. Recall that the π is a path in an acyclic projection. Thus, it contains exactly one transition dependent on positional mutex group, the transition t_k. We can split the path π around t_k, so every $t \in prefix(\mathcal{F}_i)$ and every $t' \in suffix(\mathcal{F}_i)$ are transitions independent on the positional mutex group. □

We denote the splitting of the cheapest path π from I_{g_i} to G_{g_i} in $\mathcal{T}(\Pi, \mathcal{F}_i)$ as a triple $\langle prefix(\mathcal{F}_i), t_k, suffix(\mathcal{F}_i) \rangle$. From now on, we will always be interested only in the operators in the prefixes and suffixes.

An example of such splitting is shown in Figure 4.3. For the acyclic projec-

tion $\mathcal{T}(\Pi, \mathcal{F}_1)$, the shortest path from I_{g_1} to G_{g_1} is depicted. This path can be split according to Lemma 4.6. The splitting is a triple $\langle prefix(\mathcal{F}_1), t_1, suffix(\mathcal{F}_1)\rangle$ such that $t_1 = (s_1, o_a, s_1')$, and the preffix contains the independent transition before t_1. The suffix is empty in this case.

4.1.3 The Transformation of the TSP Solution into the Optimal Plan

Now, we will focus on the transformation itself. We will start with the transformation from the solution to $TSP(\Pi)$ to an optimal plan in Π. From now on, without loss of generality, let us assume that the solution p to $TSP(\Pi)$ is in the following form $p = \langle I_P, e_1, v_1, \dots e_n, v_A, \langle v_A, I_P\rangle, I_P\rangle$. We can consider the solution in this form because it is a cycle and the edge $\langle v_A, I_P\rangle$ is in every solution (Lemma 4.3).

We have described how the transformation works in the previous subsection. To summarize, the transformation itself is based on the splitting of the cheapest paths in the acyclic projections. Every cheapest path in an acyclic projection contains the transitions necessary to achieve a goal fact. Only one of those transitions is dependent on the positional mutex group. This transition is between central states.

First, we can apply all of the operators in prefixes. For every transition between central states, it is necessary to find the path in the positional projection such that this transition is enabled (Lemma 4.5). Once all of these operators of transitions are applied, it is possible to apply operators in suffixes. As all of the goal facts are in acyclic projections, the result of these applications will be a goal state.

Definition 4.7. Given some solution $p = \langle I_P, e_1, v_1, \dots, e_{n+1}, v_A, e_{n+2}, I_P\rangle$ to $TSP(\Pi)$, the transformation of the solution p is an operator sequence $plan(p) = seq(prefix(\mathcal{F}_1), \dots, prefix(\mathcal{F}_n), l(e_1), \dots, l(e_n), suffix(\mathcal{F}_1), \dots, suffix(\mathcal{F}_n))$ such that $\langle prefix(\mathcal{F}_i), t_i, suffix(\mathcal{F}_i)\rangle$ is a splitting of the cheapest path from I_{g_i} to G_{g_i} for every $\mathcal{T}(\Pi, \mathcal{F}_i) \in Acyclic_\Pi$.

Theorem 4.8. *Given a solution p to $TSP(\Pi)$, then the operator sequence $plan(p)$ is an optimal plan in Π and the cost $c(plan(p)) = c(p) - M + \sum_{i=1}^{n} c(prefix(\mathcal{F}_i)) + \sum_{i=1}^{n} c(suffix(\mathcal{F}_i))$.*

Proof. 1. Recall that every cheapest path from I_{g_i} to G_{g_i} in the projection $\mathcal{T}(\Pi, \mathcal{F}_i)$ can be split into $prefix(\mathcal{F}_i)$, one transition dependent on $\mathcal{F}_P$, and $suffix(\mathcal{F}_i)$. The parts $prefix(\mathcal{F}_i)$ and $suffix(\mathcal{F}_i)$ contain only independent transitions. In the plan π, we first apply all of the operators of the prefixes as they are independent and, therefore, applicable to any state s such that $\mathcal{I} \subseteq s$.

Afterward, we need to apply all of the operators of transitions dependent on the positional mutex group using the cheapest paths in the $\mathcal{T}(\Pi, \mathcal{F}_P)$.

The cycle p starts in the initial state $I_P = \mathcal{I} \cap \mathcal{F}_P$. Any of the prefixes did not alter this state. Since every two consequent edges e, e' in the cycle are incident with each other, $seq(l(e), l(e'))$ is applicable in every state s such that $l(e)$ is applicable in s, as shown in Lemma 4.4. Additionally, as proven in Lemma 4.5, given $v \in V$, $v = \gamma(\mathcal{T}(\Pi, \mathcal{F}_i))$, then every label of an edge leading to v contains the operator of the transition between central states in the acyclic projection $\mathcal{T}(\Pi, \mathcal{F}_i)$.

The cycle p visits every vertex in $Enabling\text{-}positions_\Pi$. Therefore, a transition between central states of every acyclic projection is used. As a result, $seq(suffix(\mathcal{F}_1), \ldots, suffix(\mathcal{F}_n))$ can be applied as every $suffix(\mathcal{F}_i)$ contains only independent transitions.

It holds that $\mathcal{F}_P \cap \mathcal{G} = \emptyset$ from the definition of the TSP-reducible problem. As the $suffix(\mathcal{F}_i)$ is the last part of the cheapest path leading to G_{g_i} and is used for every acyclic projection, $\mathcal{G} \subseteq plan(p)[\mathcal{I}]$. The operator sequence $plan(p)$ is a well-formed plan.

2. There is no cheaper plan in Π than $plan(p)$ because the path from I_{g_i} to G_{g_i} in projection $\mathcal{F}_i$ is the cheapest, all of the labels $l(e)$ are the operators of the cheapest paths, and the total cost of the cycle p is minimized. The operator sequence $plan(p)$ is an optimal plan.

3. Clearly, that $c(seq(l(e_1), \ldots, l(e_n))) = c(p) - M$, so the cost of the whole plan $c(plan(p)) = c(p) - M + \sum_{i=1}^{n} c(prefix(\mathcal{F}_i)) + \sum_{i=1}^{n} c(suffix(\mathcal{F}_i))$.

□

For our example case, the solution to the encoding was shown in the Figure 4.2. It is a cycle between vertices I_P, a, b, v_A, I_P, in this order. The only prefix is for the acyclic projection $\mathcal{T}(\Pi, \mathcal{F}_1)$ and contains $\langle o_0 \rangle$. All of the suffixes are empty. The edges $e = \langle I_P, a \rangle$ and $e' = \langle a, b \rangle$ has labels $l(e) = \langle o_a \rangle$ and $l(e') = \langle o_1, o_b \rangle$. Therefore the transformed solution into the optimal plan is an operator sequence $\langle o_0, o_a, o_1, o_b \rangle$.

We have established the transformation from a solution to a TSP-reducible problem encoding into an optimal plan. Now, we will provide the transformation for the other way.

4.1.4 Properties of Plans

An informal description of the transformation from the optimal plan into a solution to the TSP-reducible problem encoding is as follows.

Recall that the form of the optimal plan was created by the transformation of the solution to the encoding. It was an operator sequence starting with the prefixes. After these prefixes, there is the most important part. This part consists of the operators of transitions in the positional projection. The operator sequence ended with the suffixes.

First, we will show that every optimal plan can be reordered into this exact form. The transformation from the optimal plan into a solution to encoding will split the part consisting of the operators of transitions in the positional projection. This splitting will produce operator sequences. These will be the operators of the cheapest path in the positional projection between states of facts from $\mathit{Enabling\text{-}positions}_\Pi$. These operator sequences will correspond to edges in the encoding (both of them will be the cheapest paths between the same states).

Last but not least, we will show that these edges are part of the solution to the TSP-reducible problem.

In order to know that every optimal plan can be reordered, we need to prove that it contains the cheapest path from the initial to the goal state in every acyclic projection.

Lemma 4.9. *Let $\pi = \langle o_1, \ldots, o_n \rangle$ be an optimal plan of Π and $\mathcal{T}(\Pi, \mathcal{F}_i) \in \mathit{Acyclic}_\Pi$. Then π contains operators $o_{k_1}, \ldots, o_{k_m}$, in this order, for indexes $k_1, \ldots, k_m \in \{1, \ldots, n\}$ such that $\langle o_{k_1}, \ldots, o_{k_m} \rangle$ is the cheapest path from I_{g_i} to G_{g_i} in $\mathcal{T}(\Pi, \mathcal{F}_i)$.*

Proof. Based on Definition 4.1, for every $\mathcal{T}(\Pi, \mathcal{F}_i) \in \mathit{Acyclic}_\Pi$, $\mathcal{F}_i$ contains exactly one goal fact g. Therefore, it holds that $G_{g_i} = \{g\}$. Since π is a plan, it has to contain an operator o such that $g \in \mathit{add}(o)$. As $\mathcal{T}(\Pi, \mathcal{F}_i)$ is an acyclic projection, this state is reachable using the operators in the path of transitions of T_{g_i} from I_{g_i} to G_{g_i}. The plan π is optimal, so this path has to be the cheapest. □

Recall that a prefix and a suffix of every cheapest path in $\mathcal{T}(\Pi, \mathcal{F}_i) \in \mathit{Acyclic}_\Pi$ contains only independent transitions. Using the previous lemma, from now on, we will assume that every optimal plan is in the following ordering $\pi = \mathit{seq}(\mathit{prefix}(\mathcal{F}_1), \ldots, \mathit{prefix}(\mathcal{F}_n), o_1, \ldots, o_m,$
$\mathit{suffix}(\mathcal{F}_1), \ldots, \mathit{suffix}(\mathcal{F}_n))$.

Now, when it was shown that every optimal plan could be reordered. In the next lemma, we will show that the part containing the operators of transitions in the positional projection enables the transition between central states in every acyclic projection. Recall that this property is necessary to find the corresponding edges in the encoding to the splitting of this operator sequence.

Lemma 4.10. *Let $\pi = \mathit{seq}(\mathit{prefix}(\mathcal{F}_1), \ldots, \mathit{prefix}(\mathcal{F}_n), o_1, \ldots, o_m,$
$\mathit{suffix}(\mathcal{F}_1), \ldots, \mathit{suffix}(\mathcal{F}_n))$ be an optimal plan for Π. Then the sequence $\pi' = \langle o_1, \ldots, o_m \rangle$ are the operators of the cheapest path in the positional projection that visits every $f \in \mathit{Enabling\text{-}positions}_\Pi$ and $\pi'[I_P]$.*

Proof. The sequence π' is applicable in I_P because $I_P = \mathcal{I} \cap \mathcal{F}_P$ and every prefix contains only independent transitions in the acyclic projections. As we have shown in Lemma 4.9, an optimal plan contains the cheapest paths from I_{g_i} to G_{g_i} in every $\mathcal{T}(\Pi, \mathcal{F}_i)$. Therefore, π' must contain the transitions

between all of the central states. From the definition of the TSP-reducible problem, it holds that every transition between two states in positional projection has preconditions only from the positional mutex group. Therefore, π' visits every state $f \in \mathit{Enabling\text{-}positions}_\Pi$ one by one. As π is optimal, the cost of paths between every $f, f' \in \mathit{Enabling\text{-}positions}_\Pi$ is minimized. □

As we outlined at the beginning of this subsection, the last thing that needs to be proven is that we can find the operator sequences that are operators of the cheapest paths in the positional projections between states containing facts from $\mathit{Enabling\text{-}positions}_\Pi$. These operator sequences would correspond to edges in the encoding. Afterward, in the transformation, these edges (with the addition of two edges incident with v_A) will form the Hamiltonian cycle.

In the next lemma, we will show that such splitting exists. Each split is an operator sequence. This operator sequence consists of operators of the cheapest path between two enabling states. The cheapest path of every split ends in the enabling state where the cheapest path of the next split starts. Additionally, the cheapest path of the split can end only in the enabling state that no cheapest path of any previous split has visited.

If this last condition was not enforced, we could obtain more splits than is the number of enabling states. We, clearly, do not want this, as we want to have only one edge leading to every enabling state. Otherwise, the edges would not be part of the Hamiltonian cycle.

Lemma 4.11. *Let* $\pi = \mathit{seq}(\mathit{prefix}(\mathcal{F}_1), \ldots, \mathit{prefix}(\mathcal{F}_n), o_1, \ldots, o_m, \mathit{suffix}(\mathcal{F}_1), \ldots, \mathit{suffix}(\mathcal{F}_n))$ *be an optimal plan of* Π *with the encoding* $TSP(\Pi) = \langle V, E \rangle$. *Then* $\langle o_1, \ldots, o_m \rangle = \mathit{seq}(\pi_1, \ldots, \pi_n)$ *such that every* $\pi_i = \langle o^i_1, \ldots, o^i_{k_i} \rangle$ *is an operator sequence, where* $|\mathit{add}(o^i_{k_i}) \cap \mathit{Enabling\text{-}positions}_\Pi| = 1$, *and for every* $l < i$, *it holds* $(\mathit{add}(o^i_{k_i}) \cap \mathit{Enabling\text{-}positions}_\Pi) \notin \mathit{add}(o^l_{k_l})$. *Additionally, the edge* $e_i = \langle (\mathit{pre}(o^i_1) \cap \mathit{Enabling\text{-}positions}_\Pi), (\mathit{add}(o^i_{k_i}) \cap \mathit{Enabling\text{-}positions}_\Pi) \rangle \in E$ *has a cost* $c(e_i) = c(\pi_i)$.

Proof. According to Lemma 4.10, the path $\langle o_1, \ldots, o_m \rangle$ visits every fact in $\mathit{Enabling\text{-}positions}_\Pi$. Additionally, $\mathcal{F}_P$ is a fam-group (Definition 4.1). Therefore, there has to exist the specified splitting $\mathit{seq}(\pi_1, \ldots, \pi_n)$. The plan π is optimal. Hence, the costs of the paths are the cheapest. As the costs of the edges in the encoding are the costs of the cheapest paths between two facts, $c(e_i) = c(\pi_i)$ for every π_i. □

4.1.5 The Transformation of the Optimal Plan into the TSP

In the previous section, we have outlined how the transformation of the optimal plan into the solution to the TSP-reducible problem encoding works. To shortly summarize, the prefixes and suffixes of the plan are stripped off. The remaining part is the cheapest path in the positional projection (Lemma 4.10). This part can be further split into sections depicting transitions

between states containing facts in $\mathit{Enabling\text{-}positions}_\Pi$ (Lemma 4.11). As these sections are the cheapest paths between these states, they have the same cost as edges in the TSP-reducible problem encoding. The transformed path is a sequence of these edges, followed by the sequence leading back to I_P using the vertex v_A.

Definition 4.12. Given an optimal plan π and its reordering $\pi' = \mathit{seq}(\mathit{prefix}(\mathcal{F}_1), \ldots, \mathit{prefix}(\mathcal{F}_n), \pi_1, \ldots, \pi_n, \mathit{suffix}(\mathcal{F}_1), \ldots, \mathit{suffix}(\mathcal{F}_n))$, let us define its transformation to a sequence of edges and vertices in $TSP(\Pi)$ as $\mathit{cycle}(\pi) = \langle I_P, e_1, v_1, \ldots, e_n, v_n, \langle v_n, v_A\rangle, v_A, \langle v_A, I_P\rangle, I_P\rangle$, such that e_i is an edge between the beginning and end fact of π_i.

Theorem 4.13. *Let π be an optimal plan of Π with its reordering $\pi' = \mathit{seq}(\mathit{prefix}(\mathcal{F}_1), \ldots, \mathit{prefix}(\mathcal{F}_n), \pi_1, \ldots, \pi_n, \mathit{suffix}(\mathcal{F}_1), \ldots, \mathit{suffix}(\mathcal{F}_n))$, and let $TSP(\Pi) = \langle V, E\rangle$ be the TSP-reducible problem encoding of Π. Then its transformation $\mathit{cycle}(\pi)$ is a solution to $TSP(\Pi)$ with cost $c(\mathit{cycle}(\pi)) = c(\pi) + M - \sum_{i=1}^{n} c(\mathit{prefix}(\mathcal{F}_i)) - \sum_{i=1}^{n} c(\mathit{suffix}(\mathcal{F}_i))$.*

Proof. The sequence $\mathit{seq}(\pi_1, \ldots, \pi_n)$ is a splitting of $\langle o_1, \ldots, o_m\rangle$, as shown in Lemma 4.11. Therefore, every two edges e_i and e_{i+1} for $i \in \{1, \ldots, n-1\}$ are incident. The first edge e_1 starts from the vertex I_P as $\langle o_1, \ldots, o_m\rangle$ is applicable to $\{I_P\}$ (Lemma 4.10). The sequence $\mathit{cycle}(\pi)$ is, as a result, a cycle in $TSP(\Pi)$. Moreover, it visits every vertex in V (Lemma 4.10). As the plan π is optimal, the total cost $c(\pi)$ is minimal. As shown in Lemma 4.3, every Hamiltonian cycle has to contain the edge $\langle v_A, I_P\rangle$, and the cost of every edge leading to v_A is the same. Thus, $\mathit{cycle}(\pi)$ is Hamiltonian and it is minimal.

Recall that the cost of every edge between $v, v' \in \mathit{Enabling\text{-}positions}_\Pi$ is defined as $c(\langle v, v'\rangle) = c(l(\langle v, v'\rangle))$. The cost $c(\langle v_A, I_P\rangle) = 0$ and $c(\langle v_n, v_A\rangle) = M$. The transformation cost $c(\pi) = c(\mathit{seq}(\pi_1, \ldots, \pi_n)) + c(\mathit{seq}(\mathit{prefix}(\mathcal{F}_1), \mathit{suffix}(\mathcal{F}_1))) + \cdots + c(\mathit{seq}(\mathit{prefix}(\mathcal{F}_n), \mathit{suffix}(\mathcal{F}_n)))$. Because $c(\pi_i) = c(e_i)$ (Lemma 4.11), the cost $c(\mathit{cycle}(\pi)) = M + \sum_{i=1}^{n} c(\pi_i)$. As a result, the cost $c(\mathit{cycle}(\pi)) = c(\pi) + M - \sum_{i=1}^{n} c(\mathit{prefix}(\mathcal{F}_i)) - \sum_{i=1}^{n} c(\mathit{suffix}(\mathcal{F}_i))$. □

4.1.6 Conclusion

We have shown that every TSP-reducible STRIPS problem can be solved as the traveling salesman problem. We have established the relations between the costs of the solution to the TSP and the optimal plan of the original STRIPS problem. Moreover, every optimal plan of the STRIPS problem can be transformed into a Hamiltonian cycle in its TSP-reducible problem encoding.

The TSP-reducible problems form a set of STRIPS problems that are easier to solve. As we stated in Chapter 3, the TSP is an NP-complete problem (Karp, 1972). On the contrary, solving the STRIPS problem is PSCPACE-complete (Bylander, 1994).

4.2 Visit-All Is TSP-Reducible

In this section, we would like to provide an example of a TSP-reducible STRIPS problem. We will prove that the Visit-All problem is a TSP-reducible problem. The Visit-All problem was a part of the International planning competition 2011 and 2014. It models an agent in a grid that has to visit all of the specified positions.

Structurally, the problem contains two distinct subsets of facts. The first one is a set of positional facts that represent positions in a grid. The second one is a set of goal facts. These facts represent a position in the grid that was visited. The Visit-All problem has only one type of operator. Every operator describes a movement from one position to the other, and once it is applied, the position is marked as visited. A formal definition follows.

Definition 4.14 (Visit-All problem)**.** An instance of a STRIPS problem $\Pi = \langle \mathcal{F}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ is a Visit-All problem if all of the following conditions are satisfied:

- $\mathcal{F} = \mathcal{F}_{pos} \cup \mathcal{G}$, such that $\mathcal{F}_{pos} \cap \mathcal{G} = \emptyset$.
- There exists an injective function $\gamma : \mathcal{G} \to \mathcal{F}_{pos}$.
- $\mathcal{I} = \{p\}$, where $p \in \mathcal{F}_{pos}$.
- For every operator $o \in \mathcal{O}$, it holds that $pre(o) = del(o) = \{p_1\} \subseteq \mathcal{F}_{pos}$. If there exist $g \in \mathcal{G}$ such that $\gamma(g) = p_2 \in \mathcal{F}_{pos}$, then $add(o) = \{p_2, g\}$. Otherwise, $add(o) = \{p_2\} \subseteq \mathcal{F}_{pos}$. The cost of every operator $o \in \mathcal{O}$ is $c(o) = 1$.

In the Visit-All definition, $\mathcal{F}_{pos}$ denotes the positional facts, and $\mathcal{G}$ denotes the goal facts. The function γ maps every goal fact to a specific positional fact.

4.2.1 Properties of the Visit-All problem

With the problem defined, let us show that Visit-All is a TSP-reducible problem. First of all, we need to inspect its structure, find mutex groups, their projections, and their relations. As the first step of this process, we will prove that $\mathcal{F}_{pos}$ is a fam-group and that every goal fact is a mutex group. Recall that it is necessary to split all facts into pairwise disjoint mutex groups. Otherwise, the problem would not be TSP-reducible.

Lemma 4.15 ($\mathcal{F}_{pos}$ is a fam-group)**.** *Let Π_V be a Visit-All problem $\Pi = \langle \mathcal{F}_{pos} \cup \mathcal{G}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$. Then its set of positional facts $\mathcal{F}_{pos}$ is a fam-group.*

Proof. From the definition of the Visit-All problem, it holds that there exists $p \in \mathcal{F}_{pos}, g \in \mathcal{G}$ such that $\mathcal{I} = \{p, g\}$. Thus, $|\mathcal{F}_{pos} \cap \mathcal{I}| = 1$ and the first condition of the fam-group is satisfied.

For every $o \in \mathcal{O}$, there are two possible variants of add effects and the size of the intersection is $|\mathcal{F}_{pos} \cap add(o)| = 1$ for both of them. The variant of the delete effects and preconditions is only one. Hence, $|\mathcal{F}_{pos} \cap pre(o) \cap del(o)| = |\mathcal{F}_{pos} \cap \{p\} \cap \{p\}| = 1$, for $p \in \mathcal{F}_{pos}$, such that $pre(o) = del(o) = p$.

Therefore, it is true that $|\mathcal{F}_{pos} \cap add(o)| = 1 \leq |\mathcal{F}_{pos} \cap pre(o) \cap del(o)| = 1$ and the second condition of the fam-group is satisfied. □

It is trivial that every set $S = \{f\}$ of a single fact is a mutex group because for every reachable state s it holds $|s \cap S| \leq 1$. As a result, every goal state $g \in \mathcal{G}$ forms a mutex group $\{g\}$.

Now, we have the whole set of facts $\mathcal{F} = \mathcal{F}_{pos} \cup \mathcal{G}$ split into pairwise disjoint mutex groups. With the mutex groups specified, let us show how their projections look like. Their attributes are apparent from the definition of a projection (Definition 3.19).

The transition system $\mathcal{T}(\Pi, \mathcal{F}_{pos}) = \langle S_P, \mathcal{O}, T_P, I_P, G_P \rangle$ is a projection to $\mathcal{F}_{pos}$ with the abstraction function $\alpha : S \to S_P$, such that $\alpha(s) = s \cap \mathcal{F}_{pos}$. For the transition system $\mathcal{T}(\Pi, \mathcal{F}_{pos})$, it holds:

- $S_P = \{\{f\} \mid f \in \mathcal{F}_{pos}\} \cup \{\emptyset\}$
- $I_P = I \cap \mathcal{F}_{pos}$
- $G_P = S_P$
- $T_P = \{(\alpha(s), o, \alpha(s')) \mid (s, o, s') \in T\}$

A projection of the goal fact $g \in \mathcal{G}$ is a transition system $\mathcal{T}(\Pi, \{g\}) = \langle S_g, \mathcal{O}, T_g, I_g, G_g \rangle$, where:

- $S_g = \{\emptyset, \{g\}\}$
- $I_g = I \cap \{g\}$
- $G_g = \{\{g\}\}$
- $T_g = \{(\alpha(s), o, \alpha(s')) \mid (s, o, s') \in T\}$

In order to show that the Visit-All problem is TSP-reducible, it is necessary to determine which mutex group projections are acyclic. Every TSP-reducible problem contains only one mutex group whose projection can contain cycles. In the next lemma, we will show that a projection to every goal fact is acyclic. Therefore, the only cyclic projection will be $\mathcal{T}(\Pi, \mathcal{F}_{pos})$.

Lemma 4.16 (Projection $\mathcal{T}(\Pi, g)$ is an acyclic transition system)**.** *Let $\Pi = \langle \mathcal{F}_{pos} \cup \mathcal{G}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ be a Visit-All problem. Then for every $g \in G$, the projection $\mathcal{T}(\Pi, \{g\})$ is an acyclic transition system.*

Proof. Recall that every operator $o \in \mathcal{O}$ has delete effect $del(o) \subseteq \mathcal{F}_{pos}$ (Definition 4.14). Therefore, there is no transition $(\{g\}, o, \emptyset)$ because there is no operator such that $g \notin o[s]$ if $g \in s$. □

Closely related to the goal fact projections is the following lemma. This lemma states that every transition between two distinct states in the goal fact projection is dependent on $\mathcal{F}_{pos}$. This is one of the conditions needed to be true for a TSP-reducible problem.

Lemma 4.17. *Let Π be a Visit-All problem and let $\mathcal{T}(\Pi, g)$ be an acyclic projection of some $g \in \mathcal{G}$. Then every transition between states $\emptyset$ and $\{g\}$ is dependent on $\mathcal{F}_{pos}$.*

Proof. Based on Definition 4.14, if an operator $o \in \mathcal{O}$ contains $g \in add(o)$, then $|add(o) \cap \mathcal{F}_{pos}| = 1$. Therefore every transition that changes a state in $\mathcal{T}(\Pi, g)$ is a transition dependent on $\mathcal{F}_{pos}$. □

4.2.2 Proof that Visit-All is TSP-Reducible

With all of the previous lemmas proven, we can finally prove that a Visit-All problem is a TSP-reducible problem. The proof is split into several parts according to the conditions in the TSP-reducible problem definition (Definition 4.1).

Theorem 4.18. *Every Visit-All problem Π is TSP-reducible.*

Proof. Let Π be a Visit-All problem. We will show that every condition from Definition 4.1 holds.

(i) The set of facts is defined as $\mathcal{F} = \mathcal{F}_{pos} \cup \mathcal{G}$. The set of positional facts $\mathcal{F}_{pos}$ is a fam-group as shown in Lemma 4.15 and a set $\{g\}$ is a mutex group for every $g \in \mathcal{G}$.

(ii) The projection $\mathcal{T}(\Pi, \{g\})$ to a fact $g \in \mathcal{G}$ is acyclic (Lemma 4.16) and all of these goal projections creates a set $Acyclic_\Pi = \{\mathcal{T}(\Pi, g) \mid g \in \mathcal{G}\}$.

(iii) Obviously, for every two acyclic projections $\mathcal{T}(\Pi, g)$ and $\mathcal{T}(\Pi, g')$, it holds $\{g\} \cap \{g'\} = \emptyset$. For every $\mathcal{T}(\Pi, g)$, it holds $|\{g\} \cap \mathcal{G}| = 1$ and $\{g\} \cap \mathcal{F}_{pos} = \emptyset$. From Definition 4.14, it holds $\mathcal{F}_{pos} \cap \mathcal{G} = \emptyset$.

(iv) For every operator $o \in \mathcal{O}$, it holds that $pre(o) \subseteq \mathcal{F}_{pos}$. Therefore, even for every transition $t = (s, o, s')$ in a projection $\mathcal{T}(\Pi, \mathcal{F}_{pos})$, it holds $pre(o) \subseteq \mathcal{F}_{pos}$.

(v) For every operator $o \in \mathcal{O}$, it holds that if there exists $g \in \mathcal{G}$ such that $\gamma(g) = p_2 \in \mathcal{F}_{pos}$, then $add(o) = \{p_2, g\}$. Otherwise, $add(o) = \{p_2\} \subseteq \mathcal{F}_{pos}$. Therefore, every two operators o', o'' of transitions leading to the same state have $add(o') = add(o'')$.

(vi) Provided we have an injective function γ in the Visit-All definition. We can create a different injective function $\gamma' : Acyclic_{\Pi} \rightarrow \mathcal{F}_{pos}$ such that $\gamma'(\mathcal{T}(\Pi, g)) = \gamma(g)$. As shown in Lemma 4.17, every acyclic projection has exactly two states s, s' and every transition between them is dependent on $\mathcal{F}_{pos}$. As $s = I_{g_i}$ and $s' = G_{g_i}$, every path between them contains this transition.

□

The Visit-All problem is an example of a TSP-reducible problem. Later on, we will provide the run time comparison of TSP solver with the domain-independent planner employing heuristic search. However, in the next chapter, we will provide a more general definition of problems that can be solved using TSP variants.

Chapter 5

PC-GTSP-Reducible Problem

In this chapter, a different set of STRIPS problems is presented. This set is called PC-GTSP-reducible problems and these problems can be solved as the PC-GTSP.

In the first part, we will provide the PC-GTSP-reducible problem definition and discuss its structure. Afterward, the encoding of the graph will be shown. This encoding of the PC-GTPS-reducible problem creates an instance of PC-GTSP, which is a graph. In the last part, we will prove that the solution to this PC-GTSP instance can be transformed into an optimal plan in the original STRIPS problem and vice versa.

5.1 Problem Definition

A PC-GTSP-reducible problem is a STRIPS problems with a specific structure. The core of the structure is the same as TSP-reducible problems. There must exist a splitting of facts of the problem into pairwise disjoint mutex groups. With one exception, all of the projections to these mutex groups have to be acyclic. Therefore, these mutex groups are called acyclic mutex groups and projections to them are called acyclic projections. The remaining one mutex group is called the positional mutex group, and the projection to it is called the positional projection.

Just as in the TSP-reducible problem, the positional mutex group does not contain any goal facts. On the contrary, every acyclic mutex group must contain exactly one goal fact.

A transition between distinct states in the positional projection can be independent on acyclic projections. The second possibility is that it may be dependent on some acyclic mutex groups. The only condition is that the operator of this transitions is a label of a transition between distinct states in exactly one acyclic projection. In other words, every operator of a transition

between distinct states in the positional projection is either independent on acyclic projections or it is an operator of a transition between distinct states in a single acyclic projection.

In the acyclic projection, every transition between distinct states can be dependent on the positional mutex group or partially dependent on some other acyclic mutex group. If one transition between two distinct states in the acyclic projection is partially dependent on some other acyclic mutex group, then all of the transitions between these two states has to be dependent on this acyclic mutex group as well. Moreover, facts from this acyclic mutex group in preconditions of operators of these transitions have to be the same. If one of the transitions between two distinct states is dependent on the positional mutex group, then every transition between these states must be dependent on the positional mutex group. However, their operators can have different facts from the positional mutex group in preconditions.

Lastly, every path from the initial state to the goal state in an acyclic projection must visit every state in this projection. The whole structure is formally specified in the following definition. The positional mutex group is denoted as $\mathcal{F}_P$ and the positional projection as $\mathcal{T}(\Pi, \mathcal{F}_P)$. Additionally, the acyclic mutex group will always be denoted as $\mathcal{F}_i$ with index $i \in \{1, \ldots, k\}$ and the acyclic projection to this mutex group will be denoted as $\mathcal{T}(\Pi, \mathcal{F}_i)$.

Definition 5.1 (PC-GTSP-reducible STRIPS problem)**.** A STRIPS problem $\Pi = \langle \mathcal{F}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ is called as PC-GTSP-reducible problem if all of the following holds:

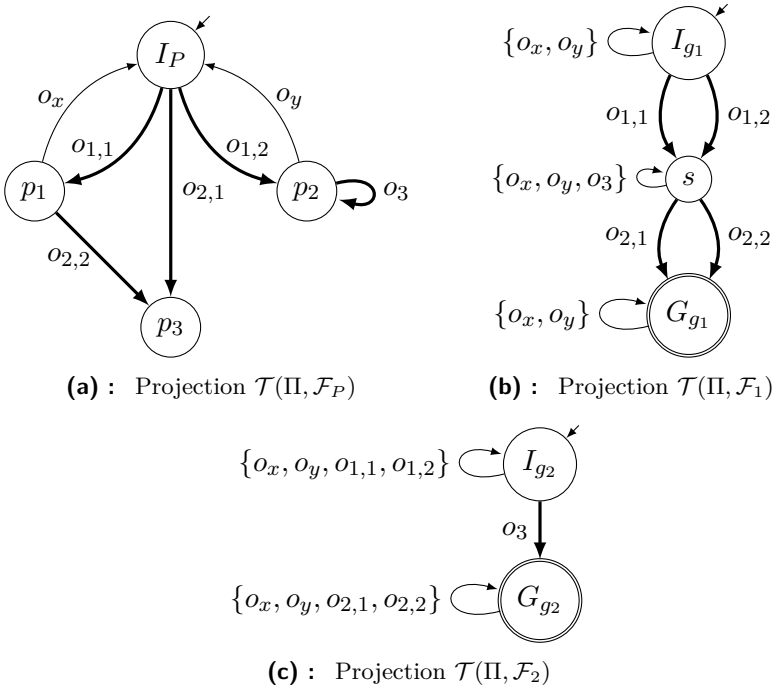
(i) There exists a set of mutex groups $\{\mathcal{F}_P, \mathcal{F}_1, \ldots, \mathcal{F}_k\}$ such that $\mathcal{F}_i \cap \mathcal{F}_j = \emptyset$ for every two $\mathcal{F}_i, \mathcal{F}_j$ and $\mathcal{F} = \mathcal{F}_P \cup \mathcal{F}_1 \cup \cdots \cup \mathcal{F}_k$. Additionally, the mutex group $\mathcal{F}_P$ is also a fam-group.

(ii) The projection $\mathcal{T}(\Pi, \mathcal{F}_P) = \langle S_P, \mathcal{O}, T_P, I_P, G_P \rangle$ of a fam-group $\mathcal{F}_P$ is cyclic and the rest of the projections to mutex groups are acyclic and form a set of acyclic projections $Acyclic_\Pi = \{\mathcal{T}(\Pi, \mathcal{F}_1), \ldots, \mathcal{T}(\Pi, \mathcal{F}_k)\}$, where each $\mathcal{T}(\Pi, \mathcal{F}_i) = \langle S_{g_i}, \mathcal{O}, T_{g_i}, I_{g_i}, G_{g_i} \rangle$.

(iii) For every $\mathcal{T}(\Pi, \mathcal{F}_i) \in Acyclic_\Pi$, it holds that $|\mathcal{F}_i \cap \mathcal{G}| = |G_{g_i}| = 1$. For the set of facts $\mathcal{F}_P$, it holds that $\mathcal{F}_P \cap \mathcal{G} = \emptyset$.

(iv) If the transition $t = (s, o, s') \in T_P, s \neq s'$ is partially dependent on acyclic mutex group $\mathcal{F}_i$, than there exists acyclic mutex group $\mathcal{F}_j$ such that t is dependent on $\mathcal{F}_j$ but not partially.

(v) For every $\mathcal{T}(\Pi, \mathcal{F}_i) \in Acyclic_\Pi$ and every pair of states $s, s' \in S_{g_i}, s \neq s'$, every transitions $t = (s, o, s'), t' = (s, o', s') \in T_{g_i}$ are dependent on $\mathcal{F}_P$ or partially dependent on some $A' \subseteq Acyclic_\Pi$. It holds that $(pre(o) \backslash \mathcal{F}_P) = (pre(o') \backslash \mathcal{F}_P)$. If t is dependent on $\mathcal{F}_P$ then $|pre(o) \cap \mathcal{F}_P| = |pre(o') \cap \mathcal{F}_P| = 1$. Additionally, every path from I_{g_i} to G_{g_i} visits every state in S_{g_i}.



(a) : Projection $\mathcal{T}(\Pi, \mathcal{F}_P)$ **(b) :** Projection $\mathcal{T}(\Pi, \mathcal{F}_1)$

(c) : Projection $\mathcal{T}(\Pi, \mathcal{F}_2)$

Figure 5.1: An example of projections in PC-GTSP-reducible problem. If the transition is dependent on $\mathcal{F}_P$, it is drawn thicker.

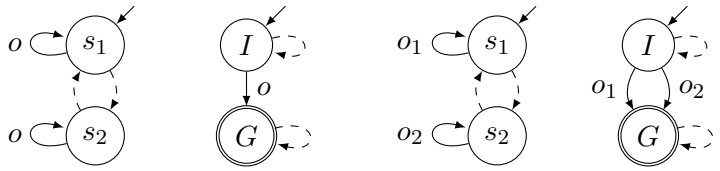
(vi) For every $o \in \mathcal{O}$, the cost $c(o) = 1$.

An example of a PC-GTSP-reducible problem is in Figure 5.1. It contains two acyclic projections $\mathcal{T}(\Pi, \mathcal{F}_1)$ and $\mathcal{T}(\Pi, \mathcal{F}_2)$. The projection $\mathcal{T}(\Pi, \mathcal{F}_P)$ is the positional projection. The acyclic projection $\mathcal{T}(\Pi, \mathcal{F}_2)$ contains a transition with the operator o_3. For o_3, it holds that $pre(o_3) \cap s \neq \emptyset$ and it is only applicable in the state $p_2 \in S_P$. For the operators $o_{2,1}$ and $o_{2,2}$, it holds $pre(o_{2,1}) \cap G_{g_2} = pre(o_{2,2}) \cap G_{g_2} \neq \emptyset$. The operators o_x, o_y are independent on acyclic projections.

In the following text, we will use this problem as a running example.

5.2 Problem Augmentation

For the encoding of the PC-GTSP-reducible problem that we will propose, it is necessary that every transition between distinct states in acyclic projections is dependent on the positional mutex group. There exists a problem augmentation, that can be used to transform every PC-GTSP-reducible problem to a problem satisficing this property. The augmentation takes an operator without the precondition and creates $|\mathcal{F}_P|$ new operators that have exactly one of the facts from the positional mutex group in their precondition.



(a) : Before augmentation **(b) :** After augmentation

Figure 5.2: An example of augmentation of a minimal problem. The operator o, independent on the cyclic projection, was replaced by o_1 and o_2 dependent on the cyclic projection.

Theorem 5.2 (PC-GTSP-reducible problem augmentation)**.** *Let $\Pi = \langle \mathcal{F}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ be a PC-GTSP-reducible problem and let $t = (s, o, s') \in T_i$ be a transition in $\mathcal{T}(\Pi, \mathcal{F}_i) \in Acyclic_\Pi$ that is not dependent on the positional mutex group. Additionally, let $O' = \{o' \mid f \in \mathcal{F}_P, pre(o') = pre(o) \cup \{f\}, del(o') = del(o), add(o') = add(o)\}$. Then, for every plan π in Π, we can swap o in π with $o' \in O'$ to obtain a plan π' in $\Pi' = \langle \mathcal{F}, \mathcal{O}', \mathcal{I}, \mathcal{G} \rangle$, where $\mathcal{O}' = (\mathcal{O} \setminus \{o\}) \cup O'$. This plan has cost $c(\pi') = c(\pi)$.*

Proof. For every reachable state s in Π, it holds that $s \cap \mathcal{F}_P \neq \emptyset$ as $\mathcal{F}_P$ is a fam-group. Therefore, the operator o is applied in the plan π in a state s' such that $s' \cap \mathcal{F}_P = \{f\}$. This operator can be swapped with $o' \in O'$ such that $pre(o') = \{f\} \cup pre(o)$. This operator o' always exists and does not alter the add-effects and delete-effects of o. This swap is made in one-for-one fashion, so $c(\pi) = c(\pi')$. □

We can do this augmentation for every operator of a transition that is not dependent on the positional mutex group in every acyclic projection. Hence, from now on, we will assume that every transition between two distinct states in an acyclic projection is dependent on the positional mutex group. Therefore, every operator in $\mathcal{O}$ is dependent on the positional mutex group.

In Figure 5.2, there is a minimal problem containing one operator in the acyclic projection that is not dependent on the positional projection. The augmented problem contains operators o_1 and o_2 instead of o. We assume, that state $s_1 = \{f_1\}$, $s_2 = \{f_2\}$, and that $f_1 \in pre(o_1)$ and $f_2 \in pre(o_2)$.

5.3 Problem Structure

Remember that we want to show an encoding of the PC-GTSP-reducible problem to a PC-GTSP instance, which is a graph. This encoding will have transitions between distinct states in acyclic projections as vertices. We want to be sure that these vertices appear in the solution in the correct order according to their precedences and their order in the acyclic projections.

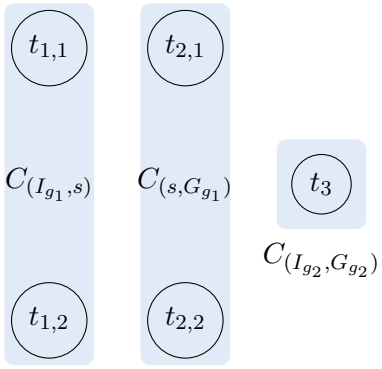


Figure 5.3: Graph with vertices corresponding to transitions between distinct states in acyclic projections from the Figure 5.1. All transitions that are in the same blue rectangle has the same value of the *cluster* function. Every transition has the same index as its operator in the label. The clusters are marked as C_X, where X is the value of the *cluster* function.

We can clearly see in our example problem that, for example, no transition with the operator $o_{2,1}$ cannot appear before the transitions with the operator $o_{1,1}$ or $o_{1,2}$.

To ensure this property, these transitions that are vertices will be split into clusters. Two transitions will be in the same cluster if they are between the same states. Recall that in the PC-GTSP, only one vertex from the same cluster can be visited. Therefore, this splitting will ensure that only one transition between two distinct states in the acyclic graph was selected.

5.3.1 Clustering of Transitions

The splitting of transitions into clusters will be done using the *cluster* function. This function creates a pair of states between which the transition is defined.

Definition 5.3. Given $\mathcal{T}(\Pi) = \langle S, \mathcal{O}, T, I, G \rangle$, let a function $cluster : T \to S \times S$ be defined by $cluster((s, o, s')) = (s, s')$.

In our example problem from Figure 5.1, the *cluster* function for transitions between distinct states in acyclic projections has three different values: $cluster((I_{g_1}, o_{1,1}, s)) = cluster((I_{g_1}, o_{1,2}, s)) = (I_{g_1}, s)$, $cluster((s, o_{2,1}, G_{g_1})) = cluster((s, o_{2,2}, G_{g_1})) = (s, G_{g_1})$ and $cluster((I_{g_2}, o_3, G_{g_2})) = (I_{g_2}, G_{g_2})$.

Figure 5.3 contains a graph with vertices representing these transitions between distinct states in acyclic projections from our example. Every transition has the same index as its operator in the label. E.g. the transition $t_{1,1} = (I_{g_1}, o_{1,1}, s)$ and $t_3 = (I_{g_2}, o_3, G_{g_2})$. Two transitions are in the same cluster if they are in the same rectangular area.

The states within every acyclic projection have a defined order in which they can appear in the path from the initial to the goal state. Additionally,

in the PC-GTSP-reducible problem, every path from the initial to the goal state in every acyclic projection leads through every state of this projection.

If we combine these two conditions, we obtain the ordering of states in every acyclic projection. In every path from the initial to the goal state, a state is visited after visiting all states of a lower order. This is a property that must be reflected even in the solution to the PC-GTSP-reducible problem encoding. It is necessary to make sure that the transitions in this solution are in the correct order.

Lemma 5.4. *Let $\mathcal{T}(\Pi, \mathcal{F}_i) = \langle S_{g_i}, \mathcal{O}, T_{g_i}, I_{g_i}, G_{g_i} \rangle$ be an acyclic projection. Then there exists an ordering $s_1 < s_2 < \cdots < s_m$ of states in $S_{g_i} = \{s_1, \ldots, s_m\}$ such that every path from I_{g_i} to G_{g_i} visits a state s_j before every s_k for $j \in \{1, \ldots, m-1\}$, $k > j$.*

Proof. Recall that every path from I_{g_i} to G_{g_i} visits the same states in S_{g_i}. Given that the projection is acyclic, there are no states s, s' for which there exists a path from s to s' and a path from s' to s at the same time. □

In our example problem, the orderings are $I_{g_1} < s < G_{g_1}$ in the acyclic projection $\mathcal{T}(\Pi, \mathcal{F}_1)$, and $I_{g_2} < G_{g_2}$ in $\mathcal{T}(\Pi, \mathcal{F}_2)$. It can be clearly seen that there exists no path in the acyclic projections that would break this order.

Using this ordering, we can define the precedence conditions between clusters in the encoding. A cluster of transitions starting in the state s must be after a cluster of transitions that start in the state s' such that $s' < s$.

This way, it is possible to create an ordering of clusters of transitions within a single acyclic projection. Recall that the transitions in the acyclic projections can also be partially dependent on some other acyclic mutex group.

In case of our example problem, the transition t_3 cannot be made before $t_{1,1}$ or $t_{1,2}$ because of the precondition on the fact from the state s.

However, the precedence conditions between clusters from different acyclic projections can be made using the state ordering as well. Recall that the acyclic projections are projections to mutex groups. Therefore, every reachable state contains at most one fact. Assume that a transition is partially dependent on an acyclic mutex group and its operator has a fact f from this acyclic mutex group in its precondition. Then the cluster of this transition must be after every cluster of transitions starting in the state s such that $s < \{f\}$.

In Figure 5.4, there is a clustering with precedence conditions for our example problem. The clustering contains clusters $C_{(I_{g_1},s)}$, $C_{(s,G_{g_1})}$ and $C_{(I_{g_2},G_{g_2})}$.

The cluster $C_{(I_{g_1},s)}$ must be before $C_{(s,G_{g_1})}$ because $I_{g_1} < s$. Additionally, $C_{(I_{g_1},s)}$ must be before $C_{(I_{g_2},G_{g_2})}$ because $pre(o_3) \cap s \neq \emptyset$. The cluster $C_{(I_{g_2},G_{g_2})}$ must be before $C_{(s,G_{g_1})}$ because $pre(o_{2,1}) \cap G_{g_2} = pre(o_{2,2}) \cap G_{g_2} \neq \emptyset$.

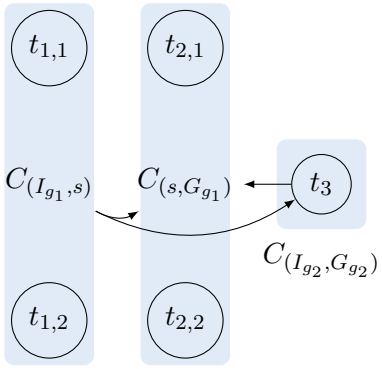


Figure 5.4: Graph containing the clustering from Figure 5.3 with precedence conditions. If an edge leads from cluster C_i to C_j then C_i must be visited before C_j.

5.3.2 Start and End Positions of Transitions

The last thing that needs to be discussed is how the edges in the encoding will be defined. Recall that every transition in every acyclic projection is dependent on the positional mutex group. In order to create an optimal plan for the PC-GTSP-reducible problem, we need to know in which order the operators of transitions in vertices must be with respect to the positional projection.

From the definition of the projection, it holds the following. Let o be an operator dependent on a mutex group $\mathcal{F}'$ and let $\mathcal{T}(\Pi, \mathcal{F}') = \langle S', L, T', I', G' \rangle$ be a projection to it. Then there exists a transition $t \in T'$ such that $t = (s, o, s')$, where $s = pre(o) \cap \mathcal{F}'$ and $s' = ((pre(o) \setminus del(o)) \cup add(o)) \cap \mathcal{F}'$.

Following this property, we will introduce a definition for the start and end position of a transition in the acyclic projection.

Definition 5.5 (Start and end position)**.** Given $\mathcal{T}(\Pi, \mathcal{F}_i) \in Acyclic_\Pi$ and the positional projection $\mathcal{T}(\Pi, \mathcal{F}_P)$ with the set of states S_P, then the start position function $\gamma_B : T \to S_P$ is defined as $\gamma_B((s, o, s')) = pre(o) \cap \mathcal{F}_P$. The end position function $\gamma_E : T \to S_P$ is defined as $\gamma_E((s, o, s')) = ((pre(o) \setminus del(o)) \cup add(o)) \cap \mathcal{F}_P$.

The result of $\gamma_B(t)$ represents the state in $\mathcal{T}(\Pi, \mathcal{F}_P)$ that is in the precondition of the transition. The second function γ_E defines the state in $\mathcal{T}(\Pi, \mathcal{F}_P)$ that is the result of the operator application. Recall that every operator is dependent on the positional projection and has exactly one fact from the positional projection in the precondition.

We will show some of the start and end positions in our example problem from Figure 5.1. The transition $t = (I_{g_2}, o_3, G_{g_2})$ has $pre(o_3) \cap \mathcal{F}_P = \{p_2\}$ and $add(o_3) \cap \mathcal{F}_P = del(o_3) \cap \mathcal{F}_P = \emptyset$. Therefore, $\gamma(t)_B = \{p_2\}$ and $\gamma(t)_E = \{p_2\}$. The transition $t' = (I_{g_1}, o_{1,1}, s)$ has $pre(o_{1,1}) \cap \mathcal{F}_P = del(o_{1,1}) \cap \mathcal{F}_P = \{I_P\}$

and $add(o_{1,1}) \cap \mathcal{F}_P = \{p_1\}$. As a result, $\gamma(t')_B = \{I_P\}$ and $\gamma(t')_E = \{p_1\}$.

5.4 Problem Encoding

For the definition of the PC-GTSP problem encoding, we need to define two more objects. The first definition introduces the set of transitions between distinct states in every acyclic projection. As we have outlined, these transitions will be vertices in the encoding.

Definition 5.6. Given $\mathcal{T}(\Pi, \mathcal{F}_i) \in Acyclic_\Pi$, then $Distinct\text{-}transitions(\mathcal{F}_i) = \{t \mid t = (s, o, s') \in T_{g_i}, s \neq s'\}$.

In our example problem, $Distinct\text{-}transitions(\mathcal{F}_1) = \{t_{1,1}, t_{1,2}, t_{2,1}, t_{2,2}\}$ and $Distinct\text{-}transitions(\mathcal{F}_2) = \{t_3\}$. It is no surprise that these sets correspond to transitions that were already used for describing the clustering.

The second definition defines a special type of the cheapest path in the positional projection. This path contains only transitions in the positional projection that are independent on the acyclic projections. It will be called the independent cheapest path (ICP). The labels of edges in the problem encoding will be found using the ICP.

The reason why the ICP is used is that the transitions dependent on the acyclic mutex groups will be represented separately as vertices. If the path was not limited only to the transitions independent on the acyclic projections, the transitions dependent on the acyclic mutex groups might be included multiple times.

Definition 5.7. Given a positional projection of a PC-GTSP-reducible problem, and the cheapest path p in the positional projection between states s, s'. Then p is an independent cheapest path $ICP(s, s')$ if it does not contain any transition dependent on an acyclic mutex group.

Figure 5.5 shows the only two ICPs with non-zero costs found in the example problem.

Right now, we can define the PC-GTSP-reducible problem encoding. The encoding takes transitions between distinct states in acyclic projections as vertices and creates a complete graph. A label of an edge between two vertices v, v' is defined as a sequence of operators between the end state of v and the start state of v' in the cyclic projection. The cost of this edge is equal to the number of operators in the label.

Recall that every vertex is a transition in an acyclic projection. Two transitions are in the same cluster if they are transitions leading between the same states. Remember, that states in every acyclic transition have an ordering (Lemma 5.4). A precedence constraint exists between two clusters

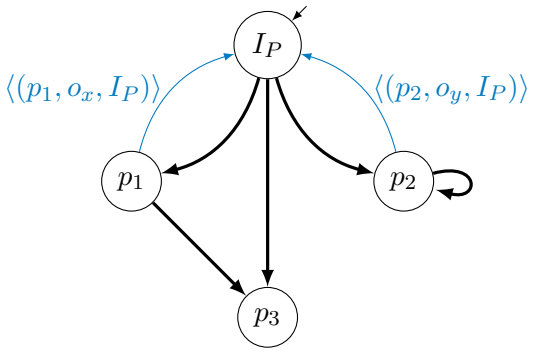


Figure 5.5: An example of the independent cheapest path in the positional projection between states. There exists two ICPs with non-zero costs. One between p_1 and I_P and one between p_2 and I_P.

C_i, C_j if they contain transitions from the same acyclic projection or the operators of transitions from one cluster are partially dependent on an acyclic mutex group whose projection contains transitions in the other cluster. A cluster of transitions starting in the state s must be after a cluster of transitions that starts in the state s' such that $s' < s$.

In the following text, we will denote a cluster $C_i \in C$ as $C_{(s,s')}$ if $cluster(v) = (s, s')$ for every $v \in C_i$. Its set of precedence constraints is denoted as $PC_{(s,s')}$ correspondingly.

The formal definition of the PC-GTSP-reducible problem encoding is as follows.

Definition 5.8 (PC-GTSP-reducible problem encoding)**.** Given the PC-GTSP-reducible problem $\Pi = \langle \mathcal{F}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$. The PC-GTSP encoding $PCGTSP(\Pi)$ is defined by a graph $PCGTSP(\Pi) = \langle V, E \rangle$ with a cost function $c : E \to \mathbb{R}$, a labeling function l, a vertex clustering C and sets of cluster precedence conditions PC_i where:

(i) $V = \bigcup_{\mathcal{T}(\Pi, \mathcal{F}_i) \in Acyclic_\Pi} Distinct\text{-}transitions(\mathcal{F}_i) \cup \{I_P, v_A\}$

(ii) $E = \{\langle p, p' \rangle \mid p, p' \in V, p \neq p'\}$

(iii) For every $e = \langle v, v' \rangle$ for $v, v' \in V \setminus \{v_A, I_P\}$, the label are the operators of $ICP(\gamma_E(v), \gamma_B(v'))$ and $c(e) = c(l(e))$. If no such path exists, let $c(e) = \infty$.

Let $e = \langle I_P, v \rangle$ and $e' = \langle v, I_P \rangle$ for $v \in V \setminus \{v_A, I_P\}$ then $l(e)$ and $l(e')$ are the operators of $ICP(I_P, \gamma_B(v))$ and $ICP(\gamma_E(v), I_P)$, respectively. Costs $c(e) = c(l(e))$ and $c(e') = c(l(e'))$. Additionally, every edge e'' incident with v_A has cost $c(e'') = 0$.

(iv) Clustering of the vertices is a set $C = \{C_1, \ldots, C_m\} \cup \{\{v_A\}, \{I_P\}\}$, where $C_i \neq \emptyset$ for every $C_i \in C$, and $C_{(s,s')} = \{v | v \in V \setminus \{v_A, I_P\}, cluster(v) = (s, s')\}$

(v) Given an acyclic projection $\mathcal{T}(\Pi, \mathcal{F}_i) = \langle S_{g_i}, \mathcal{O}, T_{g_i}, I_{g_i}, G_{g_i} \rangle$ and states $s, s' \in S_{g_i}$. Then $PC_{(s,s')} = \{\{I_P\}\} \bigcup_{(s,o,s') \in V \setminus \{I_P, v_A\}} (\{C_{cluster((s_2,o_2,s_2'))} \mid \mathcal{T}(\Pi, F_j) \in Acyclic_\Pi, pre(o) \cap \mathcal{F}_j = \{f\}, (s_2, o_2, s_2') \in Distinct\text{-}transitions(\mathcal{F}_j), s_2 < \{f\}\} \cup \{C_{cluster((s_3,o_3,s_3'))} \mid (s_3, o_3, s_3') \in V \setminus \{I_P, v_A\}, pre(o_3) \cap \mathcal{F}_i = \{f'\}, \{f'\} \leq s\})$. Additionally, $PC_{\{v_A\}} = \{C_i \mid C_i \in C \setminus \{\{v_A\}\}\}$.

A solution to the encoding $PCGTSP(\Pi)$ is the solution to the PC-GTSP in the graph $PCGTSP(\Pi)$, such that no edge e in the solution has the cost $c(e) = \infty$.

Just like in the TSP-reducible problem encoding, the PC-GTSP-reducible problem encoding contains vertices I_P and v_A. The vertex I_P is a vertex where the solution starts. The vertex v_A is the last vertex of the solution. The costs of edges incident with v_A are different than in the TSP-reducible problem encoding. The reason is that the position of vertices I_P and v_A in the solution to the PC-GTSP-reducible problem encoding is fixed using the precedence constrains. We will prove this claim in the next section.

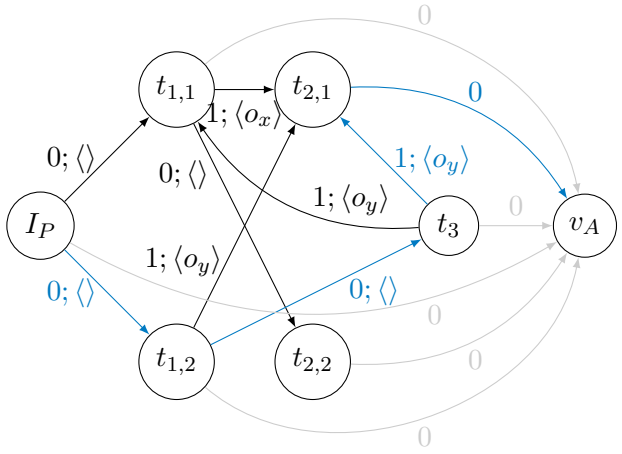
As we have stated, all of the vertices except I_P and v_A are transitions. Therefore we will mostly refer to them as transitions. Additionally, from now on, we may call the PC-GTSP-reducible problem encoding simply as the encoding. In case that we will need to refer to the TSP-reducible problem encoding, we will use the full name.

In Figure 5.6, there is a visualisation of the example problem encoding. The transitions are labeled according to which operator they contain. In the encoding there are five clusters. The cluster $C_{(I_{g_1},s)}$ contains transitions $t_{1,1}$ and $t_{1,2}$ as $cluster(t_{1,1}) = cluster(t_{1,2}) = (I_{g_1}, s)$. The cluster $C_{(s,G_{g_1})}$ contains transitions $t_{2,1}$ and $t_{2,2}$ as $cluster(t_{2,1}) = cluster(t_{2,2}) = (s, G_{g_1})$.

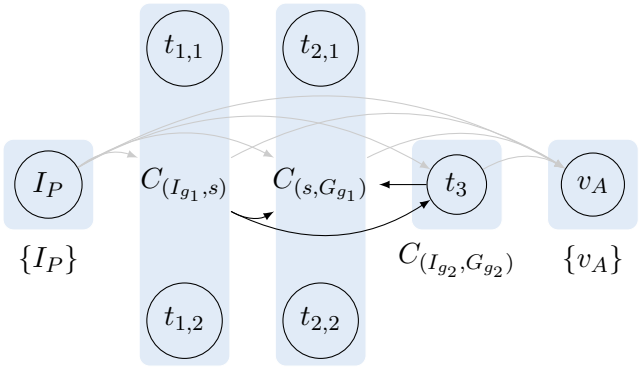
The cluster $C_{(I_{g_1},s)} \in PC_{(s,G_{g_1})}$ because $I_{g_1} < s$. Moreover, $C_{(I_{g_1},s)} \in PC_{(I_{g_2},G_{g_2})}$ because $pre(o_3) \cap s \neq \emptyset$. The cluster $C_{(I_{g_2},G_{g_2})} \in PC_{(s,G_{g_1})}$ because $pre(o_{2,1}) \cap G_{g_2} = pre(o_{2,2}) \cap G_{g_2} \neq \emptyset$.

5.5 Every Solution to the Encoding is an Optimal Plan

With the encoding defined, in this section, it will be shown that every solution to $PCGTSP(\Pi)$ can be transformed into an optimal plan in Π. As we have outlined in previous sections, the transformation of the solution will be a concatenation of operators in labels of edges, and operators of transitions in vertices.



(a) : Vertices and edges of the encoding.



(b) : Clusters in the encoding and their precedence contraints.

Figure 5.6: Encoding $PCGTSP(\Pi)$ of the example problem shown in Figure 5.1. In Figure 5.6a, only edges with costs other than infinite, and edges between distinct clusters are shown. Most of the edges e in the figure are marked with two items in the format $X;Y$, where X is the cost $c(e)$ and Y is the label $l(e)$. The rest of the edges are incident with v_A and are marked only with their cost. In Figure 5.6b, an edge leads from a cluster C_i to a cluster C_j if $C_j \in PC_i$. The solution to this problem encoding is colored blue.

In the transformation into the plan, the operators from labels will make transitions only in the positional projection. The operators of transitions in vertices will make transitions in the acyclic projections and possibly even in the positional projection.

The labels of edges contain operators that are dependent only on the positional mutex group. The operators of transitions in vertices are always dependent both on acyclic mutex groups and the positional mutex group.

We have already discussed that the reason why the PC-GTSP-reducible encoding contains vertices I_P and v_A is similar to the TSP-reducible problem encoding. These vertices ensure that every solution will start and end in them. In the first lemma, we will show that every solution to the PC-GTSP encoding must start with the vertex I_P and end with the vertex v_A.

Lemma 5.9. *Let p be a solution to $PCGTSP(\Pi)$. Then $p = \langle I_P, e_1, v_1, \ldots, e_n, v_A \rangle$, i.e., p starts with I_P and ends with v_A.*

Proof. This lemma follows directly from Definition 5.8. For every cluster $C_i \in C \setminus \{\{v_a\}\}$, it holds that $C_i \in PC_{\{v_A\}}$. Additionally, for every cluster $C_j \in C \setminus \{\{I_P\}\}$, it holds $\{I_P\} \in PC_j$. Therefore, p has to start with I_P and end in v_A. Otherwise, p would break precedence conditions and would not be a solution to $PCGTSP(\Pi)$. □

Now, let us show that the label of an edge between two consequent transitions v, v' in the solution to the encoding is applicable to the end position of the transition v and the result of this application contains the start position of the transition v'. This lemma is important to show that the labels of edges and operators of transitions create a valid operator sequence. Recall that every plan must be a path in all projections to its facts. If this lemma were not true, then the concatenation of operators of transitions dependent on acyclic projections and labels of edges would not be operators of a path in the positional projection.

Lemma 5.10. *Let $p = \langle I_P, e_1, v_1, \ldots, e_m, v_A \rangle$ be a solution to $PCGTSP(\Pi)$. Then, for every e_i, $i \in \{2, \ldots, m-1\}$, it holds $\gamma_B(v_i) \subseteq l(e_i)[\gamma_E(v_{i-1})]$, and $\gamma_B(v_1) \subseteq l(e_1)[I_P]$.*

Proof. According to Definition 5.8, every label of an edge $e = \langle v, v' \rangle$ between $v, v' \in V \setminus \{v_A, I_P\}$ is $ICP(\gamma_E(v), \gamma_B(v'))$. Therefore, every operator in $l(e)$ is independent on acyclic projections and is applicable to its state of origin $\gamma_E(v)$. It holds $\gamma_B(v_i) \subseteq l(e_i)[\gamma_E(v_{i-1})]$.

If the edge e starts in I_P, the label is $l(e) = ICP(I_P, \gamma_B(v'))$. The previous statements hold even for this case, so $\gamma_B(v') \subseteq l(e)[I_P]$. □

In the next lemma, we will show that every two transitions in the cheapest path from the initial state to the goal state in an acyclic projection are from different clusters. Recall that every solution to PC-GTSP has to visit exactly one state from every cluster. If multiple transitions on this cheapest path

were from the same cluster, the transformation from the solution would not contain every operator of this path.

Lemma 5.11. *Let $\mathcal{T}(\Pi, \mathcal{F}_i) \in Acyclic_\Pi$ and let $\pi_i = \langle t_1, \ldots, t_k \rangle$ be the cheapest path from I_{g_i} to G_{g_i} in $\mathcal{T}(\Pi, \mathcal{F}_i)$. Then $cluster(t) \neq cluster(t')$ for every two transitions $t, t', t \neq t'$, in π_i.*

Proof. Recall that every path from the initial to the goal state in the acyclic projection must visit every state. If π_i is the cheapest path, then it minimizes its cost. Therefore, it contains no transition $(s, o, s) \in T_{g_i}$ that does not change the state. Additionally, the projection is acyclic, so π_i visits every state exactly once. As a result, $cluster(t) \neq cluster(t')$ for every $t, t', t \neq t'$ in π_i. □

In the follow-up lemma, it is shown that every solution to $PCGTSP(\Pi)$ must contain transitions that create the cheapest path between the initial and the goal state for every acyclic projection. Recall that every acyclic mutex group contains exactly one goal fact and the positional mutex group does not contain any. This lemma is important for showing that the transformed solution into the plan ends in the goal state.

Lemma 5.12. *Let $p = \langle I_P, e_1, v_1, \ldots, e_m, v_A \rangle$ be a solution to $PCGTSP(\Pi)$ and $\mathcal{T}(\Pi, \mathcal{F}_i) \in Acyclic_\Pi$. Then transitions $v_{k_1}, \ldots, v_{k_l} \in T_{g_i}$ in p create the cheapest path $\langle v_{k_1}, \ldots, v_{k_l} \rangle$ from I_{g_i} to G_{g_i} in $\mathcal{T}(\Pi, \mathcal{F}_i)$ and $k_1 < k_2 < \cdots < k_l$.*

Proof. 1. Every solution to $PCGTSP(\Pi)$ has to visit every cluster of the problem. Recall that it holds $Distinct\text{-}transitions(\mathcal{F}_i) \subset V$ for the encoding. Additionally, every two transitions in the cheapest path from I_{g_i} to G_{g_i} are in different clusters (Lemma 5.11). As a result, every solution to $PCGTSP(\Pi)$ has to contain transitions $v_{k_1}, \ldots, v_{k_l} \in T_{g_i}$ such that $\langle v_{k_1}, \ldots, v_{k_l} \rangle$ is the cheapest path from I_{g_i} to G_{g_i} in $\mathcal{T}(\Pi, \mathcal{F}_i)$. Otherwise, some of the clusters would not be visited.

2. By contradiction. Suppose that p contains $v_{k_1}, \ldots, v_{k_l} \in T_{g_i}$ such that $\pi_i = \langle v_{k_1}, \ldots, v_{k_l} \rangle$ is a cheapest path, and $v_{k_r} \in \pi_i$ appears in p before $v_{k_q} \in \pi_i$ for $k_q < k_r$. As shown in Lemma 5.4, there exists an ordering of states such that every path in $\mathcal{T}(\Pi, \mathcal{F}_i)$ visits these states in this exact order. The precedence conditions are created using this ordering. If v_{k_r} appears in the path p before v_{k_q}, then $k_r < k_q$. Otherwise, the precedence conditions would not be met. But this a contradiction because $k_q < k_r$.

□

In our example problem, the solution is $p = \langle I_P, \langle I_P, t_{1,2} \rangle, t_{1,2}, \langle t_{1,2}, t_3 \rangle, t_3, \langle t_3, t_{2,1} \rangle, t_{2,1}, \langle t_{2,1}, v_A \rangle, v_A \rangle$. The transtitions $t_{1,2}$ and $t_{2,1}$ are transitions in the cheapest path $\langle t_{1,2}, t_{2,1} \rangle$ in the acyclic projection $\mathcal{T}(\Pi, \mathcal{F}_1)$ and they appear in the correct order. The transition t_3 is a transition in the cheapest path $\langle t_3 \rangle$ in the acyclic projection $\mathcal{T}(\Pi, \mathcal{F}_2)$.

In the problem structure discussion, we have described that the precedence conditions of clusters are defined based on the preconditions of operators in transitions. We will now prove that these precedences are really correctly formed and correspond to the ordering of operators of transitions in the optimal plan.

The next lemma shows that if a transition t has precondition on some state in a different acyclic projection, then it will appear in the solution right when the state containing it is reached. In other words, transitions leading to this state are before t in the solution and transitions leading from this state are after t.

Lemma 5.13. *Let p be a solution to $PCGTSP(\Pi)$ and let $t = (s_0, o_0, s_0') \in T_{g_i}$ be a transition in acyclic projection $\mathcal{T}(\Pi, \mathcal{F}_i)$. Let f be a fact from acyclic mutex group $\mathcal{F}_j$ such that $f \in pre(o)$. Let t be in the solution p. Then a transition t' reaching the state $\{f\}$ is in p before t and a transition t'' leading from the state $\{f\}$ is after t in the solution p.*

Proof. 1. By contradiction. Suppose that p is a solution and $t' = (s', o', \{f\})$ is visited after t. For the operator of the transition t it holds $f \in pre(o)$. For the state s' must hold $s' < \{f\}$ (Lemma 5.4). Therefore, $cluster(t') \in PC_{cluster(t)}$, but this is a contradiction because we assumed that t' is visited after t.

2. By contradiction. Suppose that p is a solution and $t'' = (\{f\}, o'', s'')$ is visited before t. Obviously, $\{f\} \leq \{f\}$ (Lemma 5.4). Therefore, $cluster(t) \in PC_{cluster(t'')}$ (Definition 5.8), but this is a contradiction, as we assumed that t'' is visited before t.

□

The clustering of the transitions in the example problem encoding was already shown in Figure 5.4. Just to recheck the correct order of transitions in the solution $p = \langle I_P, \langle I_P, t_{1,2}\rangle, t_{1,2}, \langle t_{1,2}, t_3\rangle, t_3, \langle t_3, t_{2,1}\rangle, t_{2,1}, \langle t_{2,1}, v_A\rangle, v_A\rangle$. The transition t_3 must be visited after $t_{1,2} = (I_{g_1}, o_{1,2}, s)$ because $pre(o_3) \cap s \neq \emptyset$. The transition $t_{2,1}$ must be visited after $t_3 = (I_{g_2}, o_3, G_{g_2})$ because $pre(o_{2,1}) \cap G_{g_2} \neq \emptyset$. All of these conditions are satisfied even in the solution p.

Right now, it is possible to define the transformation we have informally described at the beginning of this section. Right after the definition, we will prove that the transformed solution is an optimal plan.

Definition 5.14. Given a solution $p = \langle I_P, e_1, v_1, \ldots, e_{m+1}, v_A\rangle$ to the encoding $PCGTSP(\Pi)$, then the operator sequence corresponding to p is $plan(p) = seq(l(e_1), o_1, l(e_2), o_2, \ldots, o_m)$ such that v_i is labeled by the operator o_i for $i \in \{1, \ldots, m\}$.

Theorem 5.15. *Let $p = \langle I_P, e_1, v_1, \ldots, e_{m+1}, v_A\rangle$ be a solution to Π. Then $plan(p)$ is an optimal plan in Π with the cost $c(plan(p)) = c(p) + m$, where m is the number of vertices $v \in V \setminus \{I_P, v_A\}$ in p.*

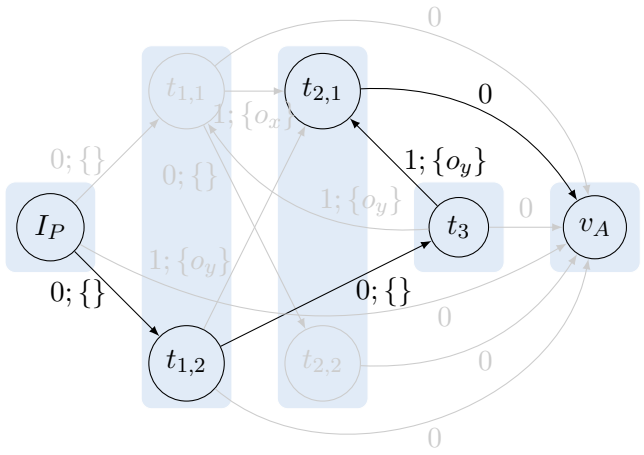


Figure 5.7: A solution to the PC-GTSP-reducible problem encoding from Figure 5.6. The edges with infinite costs and edges within clusters were intentionally left out.

Proof. 1. As shown in Lemma 5.10, every label of an edge $e = \langle v, v' \rangle$ is applicable to the result of the end state of its origin transition v such that the precondition in $\mathcal{F}_P$ of the end transition v' is satisfied. Hence, the label $l(e_1)$ is applicable to $\mathcal{I}$. Moreover, the whole sequence $plan(p)$ is a path in $\mathcal{T}(\Pi, \mathcal{F}_P)$.

2. As we have shown in Lemma 5.12, every solution to $PCGTSP(\Pi)$ contains transitions in the cheapest path from I_{g_i} to G_{g_i} in every acyclic projection $\mathcal{T}(\Pi, \mathcal{F}_i)$. These transitions appear in the correct order, so its ordering remains unchanged. Moreover, as proven in Lemma 5.13, they appear in the correct order relative to its preconditions from different acyclic projections. As a result, the whole sequence $plan(p)$ is applicable to $\mathcal{I}$.

3. Recall that $\mathcal{G} \cap \mathcal{F}_P = \emptyset$ and $|\mathcal{G} \cap \mathcal{F}_i| = 1$ for every acyclic projection $\mathcal{T}(\Pi, \mathcal{F}_i)$. Every solution to $PCGTSP(\Pi)$ contains transitions in the cheapest path from I_{g_i} to G_{g_i} in every acyclic projection. As a result, it holds $\mathcal{G} \subseteq plan(p)[\mathcal{I}]$.

4. The cost of every edge is minimal, as it is the cheapest path in $\mathcal{F}_P$. Additionally, the total length of the solution p is minimal. The cost of every edge between $v, v' \in V \setminus \{v_A\}$ is defined as $c(e) = c(l(e))$. Every edge leading to v_A from $v \in V \setminus \{v_A\}$ has a cost $c(e) = 0$. Therefore, $c(plan(p)) = c(e_1), \ldots, c(e_{m-1}) + c(o_1), \ldots, c(o_{m-1}) = c(p) + m$.

□

The solution to our example from Figure 5.6 is shown in Figure 5.7 and is a path $p = \langle I_P, \langle I_P, t_{1,2} \rangle, t_{1,2}, \langle t_{1,2}, t_3 \rangle, t_3, \langle t_3, t_{2,1} \rangle, t_{2,1}, \langle t_{2,1}, v_A \rangle, v_A \rangle$. The la-

bels are $l(\langle I_P, t_{1,2}\rangle) = \{\}$, $l(\langle t_{1,2}, t_3\rangle) = \{\}$ and $l(\langle t_3, t_{2,1}\rangle) = \{o_y\}$. Therefore, $plan(p) = \langle o_{1,2}, o_3, o_y, o_{2,1}\rangle$.

5.6 Every Optimal Plan Is a Solution to the Encoding

With the transformation of the solution in an optimal plan established, let us head into the opposite direction. In this section, we will present the transformation from an optimal plan to a solution to the problem encoding.

In the first lemma, it is shown that every plan must contain the cheapest path from the initial to the goal state in every acyclic projection. This lemma holds true because every acyclic projection contains a goal fact. This goal fact is in the goal state of the projection. This lemma is necessary for the proof of the plan transformation. Recall that the solution to the problem encoding has transitions in clusters defined as pairs of their incident states. If these cheapest paths were not in the plan, some of the clusters would never be visited.

Lemma 5.16. *Let π be an optimal plan of the problem Π and $\mathcal{T}(\Pi, \mathcal{F}_i) = \langle S_{g_i}, \mathcal{O}, T_{g_i}, I_{g_i}, G_{g_i}\rangle$ be an acyclic projection. Then π contains the operators of the cheapest path from I_{g_i} to G_{g_i}.*

Proof. Recall that $G_{g_i} \subseteq \mathcal{G}$. Therefore, there has to exist an operator o in every plan of Π such that $G_{g_i} \subseteq add(o)$. The state G_{g_i} is a state in the acyclic projection $\mathcal{T}(\Pi, \mathcal{F}_i)$ with the initial state I_{g_i}. Obviously, a path from I_{g_i} to G_{g_i} must be in every plan of Π in order to reach the goal fact in G_{g_i}. As π is the optimal plan, the path must be the cheapest. □

The plan to the example problem is $\pi = \langle o_{1,2}, o_3, o_y, o_{2,1}\rangle$. The plan contains the operators $o_{1,2}$ and $o_{2,1}$ of the cheapest path $\langle t_{1,2}, t_{2,1}\rangle$ in the acyclic projection $\mathcal{T}(\Pi, \mathcal{F}_1)$. Additionally, the plan contains o_3 and this is the operator of the cheapest path $\langle t_3\rangle$ in the acyclic projection $\mathcal{T}(\Pi, \mathcal{F}_2)$.

With this property settled, it is possible to move to more complex lemmas about optimal path structure. The following lemma shows that there are two operators o, o' whose transition is dependent on some acyclic projection. Then if operators between them have transitions independent on acyclic projections, they are the cheapest path in the projection $\mathcal{T}(\Pi, \mathcal{F}_P)$. Moreover, this path is from the state that is a part of the result of the operator o application to the state that is a part of the precondition of o'.

This lemma will be used for showing that these paths are equal to the edges in the encoding. They have the same cost and lead between the same states. Recall, that the result of the operator application and the operator precondition is the same as γ_E and γ_B functions from Definition 5.5.

Lemma 5.17. *Let $\pi = \langle o_1, \ldots, o_j, o_{j+1}, \ldots, o_{k-1}, o_k, \ldots, o_m \rangle$ be an optimal plan of Π. Let $\pi' = \langle o_{j+1}, \ldots, o_{k-1} \rangle$ be a sequence of operators dependent only on $\mathcal{F}_P$ and let operators o_j, o_k be dependent on the positional mutex group and an acyclic mutex group. Then π' are the operators of the ICP in the positional projection between the state $s = ((pre(o_j) \setminus del(o_j)) \cup add(o_j)) \cap \mathcal{F}_P$ and the state $s' = pre(o_k) \cap \mathcal{F}_P$.*

Proof. From the definition of STRIPS operators, the resulting state of applying o in state s is $o[s] = ((s \setminus del(o)) \cup add(o))$ and the operator is applicable in s if $pre(o) \subseteq s$. Recall that every operator in PC-GTSP-reducible problem is dependent on the positional mutex group. Since it is mutex group, for every operator o in π, the state in positional projection of application in the initial state of all the operators from π up to o is defined by $((pre(o) \setminus del(o)) \cup add(o)) \cap \mathcal{F}_P$. The start state of the consequent operator o' is $pre(o') \cap \mathcal{F}_P$. Since the plan π is optimal, the cost of the transitions between these operators must be minimal. □

In the plan $\pi = \langle o_{1,2}, o_3, o_y, o_{2,1} \rangle$ to the example problem, exists only one non-empty sequence of operators dependent only on the positional mutex group. That is $\langle o_y \rangle$ and the operator o_y is a label of the transition between states p_2 and I_P. In the plan, it appears between operators o_3 and $o_{2,1}$. It holds $((pre(o_3) \setminus del(o_3)) \cup add(o_3)) \cap \mathcal{F}_P = p_2$ and $pre(o_{2,1}) \cap \mathcal{F}_P = I_P$.

Now, the plan splitting will be derived. The following lemma proves that every optimal plan can be split using the operators dependent on acyclic projections. The rest of the operators are the cheapest paths in the positional projection from the previous lemma. Notice, that this splitting copies the form of edges (the cheapest paths) and vertices (operators dependent on acyclic projections). Moreover, the similarity is even bigger as these edges and vertices in the plan periodically alternates.

Lemma 5.18 (Plan splitting)**.** *Let π be an optimal plan in Π. Then π can be split such that $\pi = seq(\pi_1, o_1, \pi_2, o_2, \ldots, o_m)$, where $o_1, \ldots, o_m$ are operators dependent on $\mathcal{F}_P$ and some $\mathcal{F}_i$ such that $\mathcal{T}(\Pi, \mathcal{F}_i)$, and π_i is a (possibly empty) sequence of operators dependent only on $\mathcal{F}_P$.*

Proof. Every operator in the PC-GTSP-reducible problem is dependent on $\mathcal{F}_P$. Hence, every operator in the plan π is in one of those categories. As the operator sequences π_i can be even empty, then the sequence must exist. □

In the example problem plan $\pi = \langle o_{1,2}, o_3, o_y, o_{2,1} \rangle$, the splitting is $\pi = seq(\langle\rangle, o_{1,2}, \langle\rangle, o_3, \langle o_y \rangle, o_{2,1})$.

In the next definition, the transformation of an optimal plan to a sequence of edges and vertices is given. Right after the definition, it is shown that this transformation is a solution to the problem encoding.

The transformation is closely related to the previous lemma. It utilizes the similarity between plan splitting and paths in the graph that was already

informally outlined.

Definition 5.19. Given an optimal plan $\pi = seq(\pi_1, o_1, \pi_2, o_2, \ldots, o_m)$ in Π, we denote the sequence of edges and vertices in $PCGTSP(\Pi)$ as $path(\pi) = \langle I_P, e_1, v_1, \ldots, v_m, \langle v_m, v_A\rangle, v_A\rangle$, where v_i is the transition between distinct states labeled by o_i in an acyclic projection, and $e_i = \langle v_{i-1}, v_i\rangle$.

Theorem 5.20. *Let $\pi = seq(\pi_1, o_1, \pi_2, o_2, \ldots, o_m)$ be an optimal plan in Π. Then $path(\pi)$ is a solution to $PCGTSP(\Pi)$ with a cost $c(path(\pi)) = c(\pi) - m$.*

Proof. 1. Recall that every transition that is dependent on an acyclic projection is a transition between distinct states in exactly one acyclic projection. Therefore the selection of $v_i \in path(\pi)$ is unique.

2. Every edge e_i is incident with the vertices and $c(e_i) = c(\pi_i)$ as every π_i is the cheapest path between $\gamma_E(v_{i-1})$ and $\gamma_B(v_i)$ (Lemma 5.17).

3. The $plan(\pi)$ starts in I_P and ends in v_A. Moreover, π contains the cheapest path from I_{g_i} to G_{g_i} in every acyclic projection (Lemma 5.16). Therefore, a transition from every cluster $C_i \in C$ is visited.

4. Clearly, the precedence constraints of clusters $\{v_A\}$ and $\{I_P\}$ are satisfied. Every other cluster C_i has all its precedence constraints satisfied as well because the order of operators in $path(\pi)$ respects the ordering defined in Lemma 5.4.

5. There are no transitions $v, v' \in path(\pi)$ such that $cluster(v) = cluster(v')$ because the projections $\mathcal{T}(\Pi, \mathcal{F}_i)$ are acyclic (Lemma 5.4).

6. The paths in acyclic projections are the cheapest (Lemma 5.16) and paths in $\mathcal{T}(\Pi, \mathcal{F}_P)$ are also the cheapest (Lemma 5.17). The total cost of π is minimal and so is the cost of $path(\pi)$.

7. For the cost of every π_i, it holds that $c(\pi_i) = c(e_i)$, as shown in Lemma 5.17, every π_i is the independent cheapest path in $\mathcal{F}_P$. The cost of the plan is given as a sum of costs of every π_i and costs of the operators dependent on some acyclic projection. Therefore, $c(path(\pi)) = \sum_{i=1}^{m} c(e_i) = \sum_{i=1}^{m} c(\pi_i)$ and $c(\pi) = m + \sum_{i=1}^{m} c(\pi_i)$. As a result, $c(path(\pi)) = c(\pi) - m$.

□

The optimal plan of our example problem from Figure 5.1 is $\pi = \langle o_{1,2}, o_3, o_y, o_{2,1}\rangle$. The splitting of this plan is $\pi = \langle \pi_1, o_{1,2}, \pi_2, o_3, \pi_3, o_{2,1}\rangle$ where $\pi_1 = \pi_2 = \langle\rangle$ and $\pi_3 = \langle o_y\rangle$. As a result, $path(\pi) = \langle I_P, \langle I_P, t_{1,2}\rangle, t_{1,2}, \langle t_{1,2}, t_3\rangle, t_3, \langle t_3, t_{2,1}\rangle, t_{2,1}, \langle t_{2,1}, v_A\rangle, v_A\rangle$.

5.7 Conclusion

In this chapter, we have defined PC-GTSP-reducible problems. These problems are a subset of STRIPS problems that can be solved using the PC-GTSP. We have shown the transformations from the PC-GTSP solution to the optimal plan and vice versa.

Compared to TSP-reducible problems, PC-GTSP-reducible problems have their advantages and disadvantages. PC-GTSP-reducible problems can contain a larger variety of operators. Operators can be partially dependent on acyclic mutex groups, and every path from the initial to the goal state in every acyclic projection can contain multiple transitions dependent on the positional mutex group. TSP-reducible problems are more restricted. The paths from the initial to the goal state in every acyclic projection can contain exactly one transition dependent on the positional mutex group. Moreover, the operators cannot be partially dependent on acyclic mutex groups. The advantage of the TSP-reducible problem is that its encoding can be solved using TSP. As we have discussed in Chapter 2, the PC-GTSP is a generalization of the TSP, and the TSP solvers are able to solve problems of much larger sizes than PC-GTSP solvers.

In the next chapter, the results of computational experiments of TSP-reducible and PC-GTSP-reducible problems will be shown.

Chapter 6

Experimental Results

In Chapter 4 and Chapter 5, we have proposed encodings of TSP-reducible and PC-GTSP-reducible problems as TSP and PC-GTSP instances, respectively. In this chapter, we will experimentally evaluate these proposed encodings and discuss their performance.

We have implemented a detector of TSP-reducible and PC-GTSP-reducible problems. Its implementation is in Python, and as its input, it takes the Cplan[1] FDR output. This detector builds projections to mutex groups of the problem, and it verifies whether TSP-reducible or PC-GTSP-reducible problem structure conditions are met. We have run this detector on all of the problem instances of IPC 2011, 2014, and 2018 optimal tracks. With two exceptions, checked instances did not meet the condition of having all but one acyclic projections to mutex groups. However, the Visit-All problem was determined as a TSP-reducible and OpenStacks as a PC-GTSP-reducible problem.

In the following sections, we will experimentally evaluate the performance of solving their instances as the TSP and the PC-GTSP, respectively. These results will be compared to the performance of A* search with different heuristic functions of the Fast-Downward planner (Helmert, 2006). Fast-Downward is an award-winning planner that is widely used for domain-independent planning and offers a large variety of different heuristics.

From these offered heuristics, we have decided to include a blind heuristic, which is basically the BFS. This heuristic made the search uninformed and was included as an evaluation baseline. On top of this heuristic, we have selected several state-of-the-art heuristics. We have used the relaxation heuristic h_{max} (Bonet and Geffner, 2001), the Landmark-Cut (lmc) abstraction heuristic (Helmert and Domshlak, 2009), the merge and shrink heuristic with SCC-DFP merge strategy and bisimulation non-greedy shrink strategy ($m\&s$) (Helmert et al., 2014; Sievers et al., 2016), the operator

[1] `https://gitlab.com/danfis/cplan`

counting heuristic (*oc*) Pommerening et al. (2014), and the potential heuristic optimized for all states (*pot*) (Seipp et al., 2015).

All of the tests were conducted under the International Planning Competition test criteria. For every test, the tested program was given 30 minutes and 4GB of memory to solve the problem using a single CPU core. If the tested program ran out of time or memory, the test was terminated.

Because our implementations use the Cplan translator for preprocessing PDDL files, we have replaced the original Fast-Downward translator with it. This was done to ensure that differences in planning performance are not influenced by using different translators for our implementation and Fast-Downward planning.

6.1 Visit-All

The first tested problem was Visit-All. This problem was already discussed in Chapter 4. As a reminder, it is an IPC 2011 and 2014 problem that models an agent in a grid, and this agent has to visit all of the specified locations.

In the International Planning Competition, Visit-All problems were in one of two forms. The goal was to visit either every location in the grid (Variant 1) or a subset of the locations in the grid (Variant 2). As we will show later on, these two forms varied a bit in the results of the Fast-Downward planner, as most of the heuristics were able to solve more instances of Variant 2 than Variant 1. Additionally, some of the instances might have contained a redundant fact that the initial location was visited. This fact was removed in the preprocessing.

As we have stated earlier, the Visit-All problems consist of a regular structure. The locations are organized into a square grid, where each location has at most four neighbors. Additionally, every location can be reached from the initial state.

All of the Visit-All tests were conducted on Intel Xeon Gold 5120 with 4GB RAM. Computational resources were provided by the ELIXIR-CZ project (LM2015047), part of the international ELIXIR infrastructure.

6.1.1 Implementation

All of the IPC problems are defined in the Planning domain definition language (PDDL). Our implementation of the Visit-All TSP-reducible solver is written in C, and it is using the translator from Cplan for the translation of the PDDL into the STRIPS. This STRIPS problem was first verified whether it is a TSP-reducible problem. Then the positional projection was identified,

		TSP	*blind*	h_{max}	*lmc*	*m&s*	*oc*	*pot*
Variant 1	Solved	24	5	5	5	5	19	12
	Not solved	0	19	19	19	19	5	12
	Solved ratio	1.00	0.21	0.21	0.21	0.21	0.79	0.5
Variant 2	Solved	16	7	8	11	8	11	11
	Not solved	0	9	8	5	8	5	5
	Solved ratio	1.00	0.43	0.50	0.69	0.50	0.69	0.69

Table 6.1: Comparison of the number of solved Visit-All instances. Our solver is denoted as *TSP*.

and the encoding created. For the calculations of the cheapest paths and edge labels, the Floyd-Warshall algorithm was used.

Recall that the encoding is a TSP instance. This instance was solved using the Concorde solver (Applegate et al., 2006). It is an optimal state-of-the-art solver for the TSP that is able to solve TSP instances of sizes more than 85000 vertices (Applegate et al., 2011). Concorde solver was used with the default settings, and the TSP instance was given to it in the TSPLIB file format (Reinelt, 1995) with edge weights specified as an adjacency matrix.

6.1.2 Results

We have created a test set containing all of the Visit-All instances from IPC 2011 and IPC 2014. In total, 40 problem instances of grid sizes ranging from 2×2 up to 18×18 were in the test set. Out of them, 16 instances were Variant 2 with a subset of goal locations. The remaining 24 instances were Variant 1, where every location must be visited.

In Table 6.1, there is a summary of how many instances each approach was able to solve. The solved ratio is a proportion of the number of solved instances to the total number of instances of the same variant.

Our solver was able to plan all of the instances in the test set successfully. Out of the Fast-Downward heuristics, *oc* was the most successful one. In total, it was able to find a plan for 30 out of 40 instances. More specifically, it has found a plan for 19 out of 24 instances of Variant 1, and a plan for 11 out of 16 instances of Variant 2. Fast-Downward heuristics generally had a problem with the memory limit of 4GB. The only exception was *oc* and *lmc* that were always terminated due to the time limit. It is interesting to note the difference between ILP based heuristics (*oc* and *pot*) and their ability to solve much more Variant 1 instances than any other Fast-Downward heuristics, while their performance on Variant 2 was roughly the same.

The runtime dependency on the instance size of Fast-Downward heuristics and our solver is in Figure 6.1. It depicts how much time different approaches

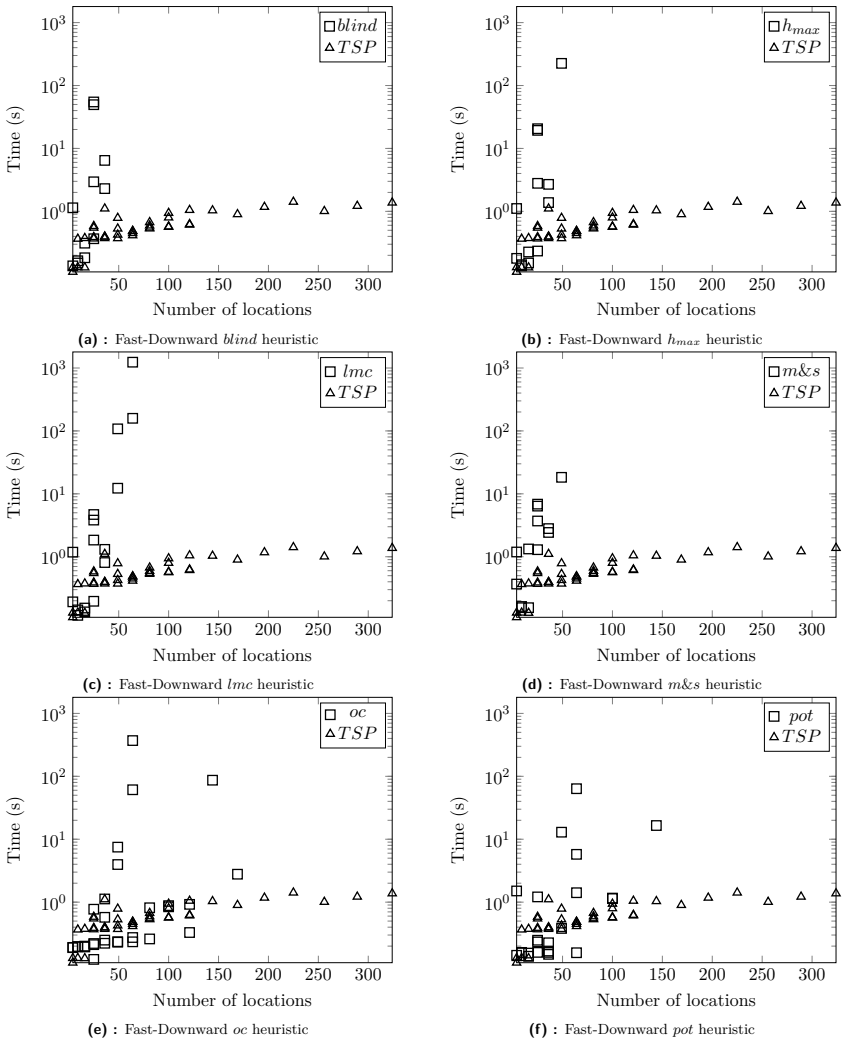
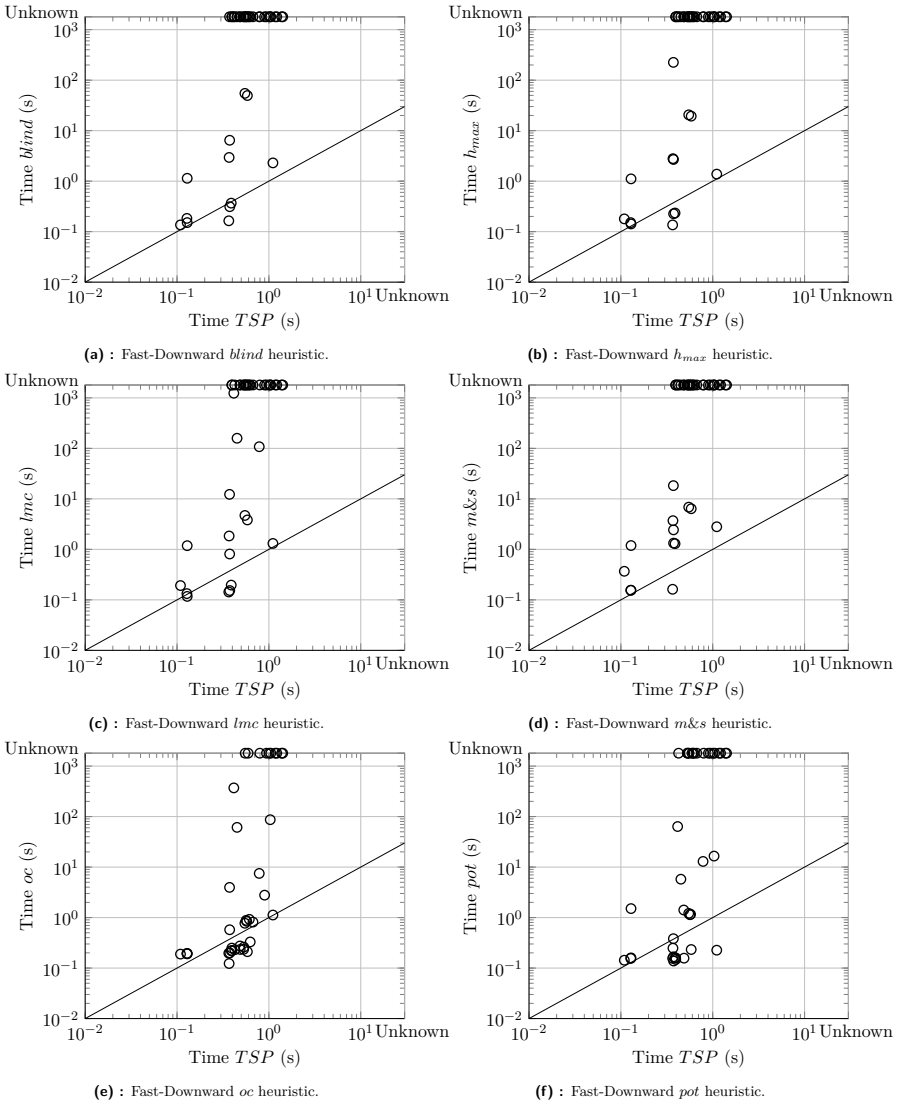


Figure 6.1: Run time comparison of different Fast-Downward heuristics with our solver depending on the problem size. Each of the figures shows only successfully solved instances.

took on instances of a specific size. Recall that instances are square grids. Therefore, if, for example, the number of locations is 64, then the problem instance contains 8×8 grid.

From these graphs, it is clearly visible, that running times of all of the Fast-Downward heuristics rose very quickly with the increasing number of locations. Our solver was able to manage nearly constant running time on all of the Visit-All instances. All of the Fast-Downward heuristics were faster than our solver on small instances. Most prominent are heuristics *oc* and *pot*. This property exists because of the overhead of our solver. Recall that we need to check whether the STRIPS problem is TSP-reducible and create the encoding to the TSP instance. This overhead becomes negligible with the increasing size of problem instances because the solving of the TSP becomes



(a) : Fast-Downward *blind* heuristic.

(b) : Fast-Downward h_{max} heuristic.

(c) : Fast-Downward *lmc* heuristic.

(d) : Fast-Downward *m&s* heuristic.

(e) : Fast-Downward *oc* heuristic.

(f) : Fast-Downward *pot* heuristic.

Figure 6.2: The direct comparison of runtimes of Fast-Downward heuristics and our solver. Every point in the graph is a single Visit-All instance.

the most time-demanding task.

Direct comparisons of running times of Fast-Downward heuristics and our solver are provided in Figure 6.2. Every one of these scatter plots compare the time needed for solving an instance by our solver and a single Fast-Downward heuristic. A single point in the graph is a problem instance from our test set. Its coordinates are the time needed for solving an instance by our solver (x-axis) and Fast-Downward heuristics (y-axis). If an instance was not solved within 30 minutes or the planner allocated more than 4GB of memory, its time is plotted as Unknown.

From these plots, it is possible to clearly see the smallest instances that Fast-Downward heuristics were faster on. The fact that our solver had

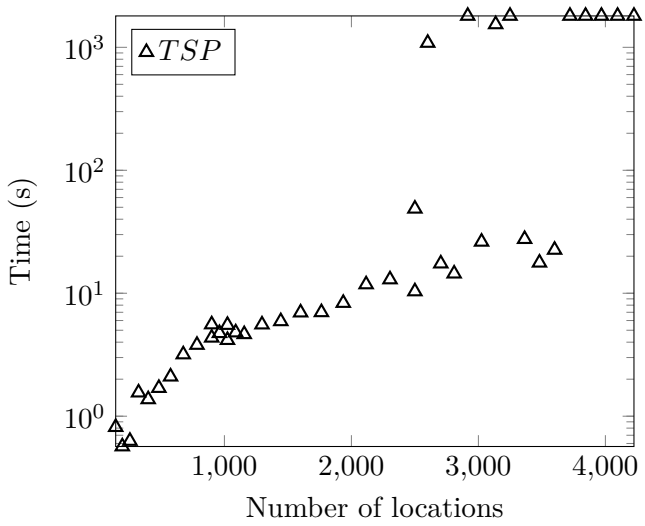


Figure 6.3: The runtime dependency of TSP-reducible problem encoding on the size of Visit-All instances. The time is in seconds and plotted in logarithmic scale. The set of instances is taken from IPC 2011 and 2014 satisficing track.

approximately constant running times is also apparent from the grouping of the points in the proximity of the $1s$ mark on the x-axis.

Our solver was able to find a plan for every instance in the test set within 1 second. Therefore, the experiments on the instances from the optimal track did not provide us any approximation of the size of instances that our solver will not be able to plan successfully. To find such an approximation, we have created a second test set. This test set contained all 40 Visit-All Variant 1 instances from the satisficing track of IPC 2011 and IPC 2014. The sizes of grids in this test set were much larger and span from 18×18 up to 65×65.

Results of our solver are shown in Figure 6.3. It was possible to solve most of the instances up to the size 60×60. During the testing, a few outliers appeared that took more time than most of the other instances. These outliers were most likely created by the heuristics used in the Concorde solver as the instances themselves do not differ in any way. All of them are regular grid instances where every location must be visited. The encoding process itself creates the graph every time in the same way, and the Concorde planner took, on average, 95% of the total run time on every instance in this test set.

We have also conducted an experiment to find out how long it would take for our solver to find a solution to the largest instance. In this experiment, there was no time limit; only the memory limit on 4GB of RAM was still in place. The solving of this largest satisficing instance with 65×65 grid took approximately 6 hours.

6.2 OpenStacks

The second tested problem is called OpenStacks. Once again, it is a problem from the International Planning Competition that was present in both optimal and satisficing track.

This problem depicts a scenario where we have a number of orders that we have to ship to our customers. Every order starts as a waiting order and contains a set of products that we have to make. When we decide to start preparing an order, it becomes open. Only when all of the products in the order are made, it can be shipped. The last condition is that a product can be made only when all of the orders containing it are open.

As our test set, we have chosen all of the 20 problem instances from IPC 2011. Additionally, we have included 24 problem instances generated using an OpenStacks pddl-generator[2]. These instances were included because IPC 2011 instances start with too many orders, products, and stacks. Therefore, it would be hard to measure the run time dependency on the problem size.

For every IPC OpenStacks problem instance, it holds that the number of stacks, orders, and products is equal. We will denote this number as n.

The size of problem instances from IPC 2011 ranges from $n = 10$ up to $n = 29$. Our generated instances have sizes between $n = 1$ and $n = 8$.

The OpenStacks problems contain operator costs 0 and 1. However, without formal proof, every solution to the same problem with all costs equal to 1 has the same optimal plan. This is due to the fact that for every product we always need to use one operator to make it, and for every order, we always need two operators - to open it and to ship it. These operators have zero costs. The only type of operators that have cost equal to one are the operators that open a new stack. In both metric and non-metric version of the problem, their number must be minimal. As a result, all of the tested problems had all operator costs equal to 1.

Just as experiments with Visit-All, all of the tests were conducted with the 30 minutes time limit and the 4GB memory limit. The OpenStacks testing was done on a single core of the Intel i5 4670k CPU.

Our OpenStacks PC-GTSP-reducible problem solver is implemented in Python and uses the Cplan translator for the translation of the PDDL problem into the STRIPS problem. First, this STRIPS problem is verified whether it is a PC-GTSP-reducible problem, and then the positional and acyclic projections are created. According to our proposed PC-GTSP-reducible problem encoding, we create a PC-GTSP instance.

We have tested two distinct approaches on how to solve these instances.

[2] `https://github.com/AI-Planning/pddl-generators`

The first one has been using the LKH-3 solver (Helsgaun, 2017), and the second one the ILP representation proposed by Salman (2015).

6.2.1 LKH-3 Version

In this subsection, we will discuss the first approach with the LKH-3 solver. LKH-3 is an approximate solver for multiple variants of the TSP. One of them is the SOP (so-called PC-ATSP). As we have shown in Chapter 2, there exists a transformation from the GTSP to the ATSP.

In our implementation, the PC-GTSP problem was transformed into PC-ATSP using the transformation proposed by Noon and Bean (1993). This transformation creates a zero-cost cycle in every cluster and increases costs of edges incident with the cluster. This ensures that all of the vertices in the cluster are visited before moving to a different cluster. The precedence constraints that were between each of the clusters were converted into vertex precedence constraints, i.e., every vertex from the cluster preceding C has to be in the precedence conditions of every vertex in the cluster C.

The LKH-3 solver was tested in various settings. We have tried varying the number of runs between 1 and 100, the number of trials between 100 and 1,000,000, and also various different initial tour algorithms, namely, Greedy algorithm, Nearest Neighbor algorithm, a walk in the graph, and Boruvka's algorithm. As the input of the LKH-3 we were providing our PC-ATSP instance in the TSPLIB file format with edge weights specified as an adjacency matrix.

Experimental testing of LKH-3 on the test set provided very poor results. The PC-GTSP-reducible problem encoding of the OpenStacks instances contained a high number of edges that were assigned the infinite cost because there exists no cheapest path between their incident vertices in the cyclic projection. The LKH-3 solver does not support the non-existence of an edge between two vertices and is defined only for the problems on the complete graph. Even though these non-existent edges were assigned very high costs, the solver was still including these edges in solutions because it is an approximate solver.

From the definition of the PC-GTSP-reducible problem (Chapter 5), only the solutions with no edges with the infinite cost are considered valid and can be transformed to the STRIPS plan. As a result, the LKH-3 solver was unable to provide us such approximate solutions that can be transformed into a STRIPS plan. Only for the smallest instances ($n < 4$) in the test set, correct solutions were obtained.

	ILP	*blind*	h_{max}	*lmc*	*m&s*	*oc*	*pot*
Solved	19	40	41	41	40	35	41
Not solved	25	4	3	3	4	9	3
Solved ratio	0.43	0.91	0.93	0.93	0.91	0.80	0.93

Table 6.2: Comparison of the number of solved OpenStacks instances.

6.2.2 ILP Version

Our second approach was using the ILP for solving the PC-GTSP instance. An ILP solver CPLEX[3] (version 12.10) was used for this task. It is a state-of-the-art commercial ILP solver that provides very good results in comparison to the other ILP solvers (Jablonsky, 2015).

In our solver, we were transforming the PC-GTSP instance into the ILP representation proposed by Salman (2015). This representation has a polynomial number of subtour elimination constraints and needs $O(n^2 + 2m^2)$ variables and $O(n^2 + m^3)$ constraints, where n is the number of vertices and m is the number of clusters.

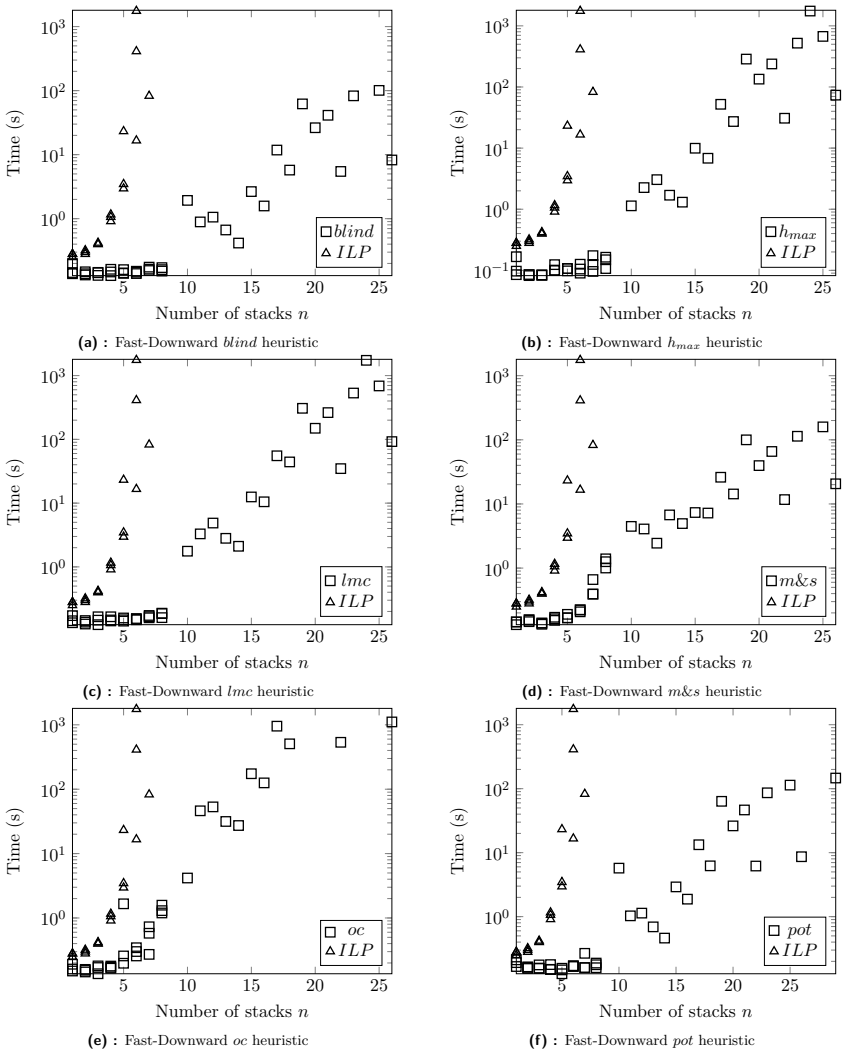
The number of solved instances from the OpenStacks test set is shown in Table 6.2. From the table, it is apparent that all of the Fast-Downward heuristics were able to solve more instances than our solver.

The runtime dependency on the problem size is plotted in Figure 6.4. From the figure, it is clearly visible that our solver was able to solve only small instances. On instances with $n \geq 7$, our solver quickly ran out of the memory. On the contrary, the best Fast-Downward heuristics were able to find a plan for instances with $n \leq 26$.

The runtime comparisons of the time needed for solving an instance by our solver and Fast-Downward heuristics are shown in the scatter plots in Figure 6.5. A single point in the graph is a problem instance from our test set. Its coordinates are the time needed for solving an instance by our solver (x-axis) and Fast-Downward heuristics (y-axis). From these plots, it is visible that our solver was slower than Fast-Downward heuristics and capable of solving fewer instances.

These poor results were caused by the mentioned number of variables and constraints needed for the ILP PC-GTSP representation. Recall that in our PC-GTSP-reducible problem encoding, the number of vertices is equal to the number of transitions between distinct states in acyclic projections. The number of these transitions in an OpenStacks instance can be calculated. Given an OpenStacks STRIPS problem with s stacks, o orders, and p products, then, without formal proof, the number of transitions in the acyclic projection

[3] `https://www.ibm.com/analytics/cplex-optimizer`



(a) : Fast-Downward *blind* heuristic

(b) : Fast-Downward h_{max} heuristic

(c) : Fast-Downward *lmc* heuristic

(d) : Fast-Downward *m&s* heuristic

(e) : Fast-Downward *oc* heuristic

(f) : Fast-Downward *pot* heuristic

Figure 6.4: Run time comparison of different Fast-Downward heuristics with our solver depending on the problem size. Each of the figures shows only successfully solved instances.

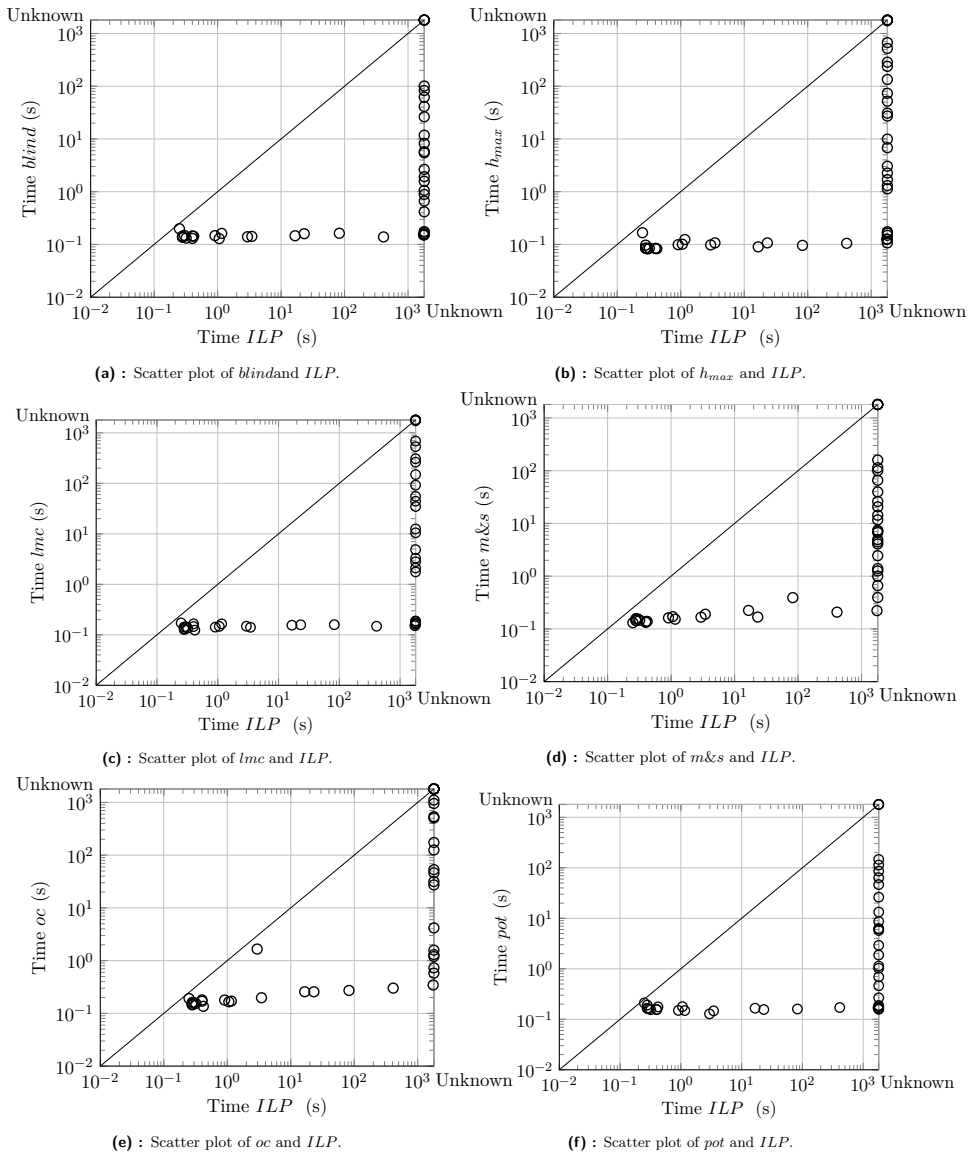
is equal to $2so + sp$. Combined with the number of both variables and constraints that grows very quickly with increasing size of instances, the ILP PC-GTSP representation have a significant overhead.

6.3 Conclusion

Our solver using the TSP-reducible problem encoding for solving IPC Visit-All problem instances was shown to be very effective. Using the encoding, our TSP solver outperformed state-of-the-art heuristics of the Fast-Downward planner both in terms of running time and number of instances solved. It was possible not only to optimally solve all of the 40 instances of the optimal track

of the IPC but even solving 32 out of 40 larger instances from the satisficing track of this competition.

The solver using the ILP to solve the PC-GTSP-reducible problem encoding of OpenStacks problem instances has exhibited poor experimental results. This solver had slower run times and was capable of solving fewer instances than heuristics of Fast-Downward. These results were due to the large number of variables and constraints in the ILP.



(a) : Scatter plot of *blind*and ILP.

(b) : Scatter plot of h_{max} and ILP.

(c) : Scatter plot of *lmc* and ILP.

(d) : Scatter plot of *m&s* and ILP.

(e) : Scatter plot of *oc* and ILP.

(f) : Scatter plot of *pot* and ILP.

Figure 6.5: The scatter plot of the direct comparison of run times of the Fast-Downward planner and our solver. Every point is one problem instance. On the x-axis, there is a runtime of our solver. On the y-axis, there is a runtime of the Fast-Downward planner.

Chapter 7

Conclusion

In this work, we have focused on defining a set of STRIPS problems that can be solved using TSP variants. Using splitting of the set of facts into pairwise disjoint mutex groups, we have formulated two classes of such problems. These two classes of problems differ in various restrictions on the structure of projections into these mutex groups.

First, we have defined TSP-reducible STRIPS problems and proposed their encoding into a TSP instance, which is a graph. We have proven that every solution to this instance can be transformed into an optimal plan in the original STRIPS problem. Additionally, we have defined the relationship between the cost of the solution to the TSP instance and the cost of the optimal plan. Furthermore, we have proven that the IPC problem called Visit-All is TSP-reducible.

Afterward, we have defined PC-GTSP-reducible STRIPS problems and shown their encoding. In this case, the encoding is a PC-GTSP instance. PC-GTSP-reducible problems do not have the structure as constrained as TSP-reducible problems, as their projections into mutex groups can be more complex.

For PC-GTSP-reducible problems, we have proven that there exists a transformation of every solution to their encoding into the optimal plan in the original STRIPS problem. Again, we have shown a relationship between the cost of the solution to the encoding and the cost of the optimal plan.

In the last part, we have implemented a solver for the Visit-All problem as an example of TSP-reducible problems. This solver was transforming the STRIPS problem into the TSP-reducible problem encoding and was solving the TSP instance using a state-of-the-art TSP solver called Concorde. Additionally, a solver for the PC-GTSP-reducible IPC problem called OpenStacks was implemented. This solver was using the ILP solver CPLEX to solve the PC-GTPS-reducible problem encoding created from the problem's STRIPS representation. The performance of both of these solvers was experimentally

evaluated and compared to several types of heuristic search of the state-of-the-art solver Fast-Downward.

These experimental tests demonstrated very good results of the Visit-All TSP-reducible problem encoding. This approach had better run times and was capable of solving more problem instances than any of heuristic searches of Fast-Downward.

The PC-GTSP-reducible problem encoding of OpenStacks exhibited poor results due to a large number of vertices that are necessary for the encoding. This approach had worse run times than any of heuristic searches of Fast-Downward, and it was capable of solving fewer instances.

In further works, it would be beneficial to check other possibilities of solving the PC-GTSP and experimentally evaluate whether they will provide better results than the domain-independent planner. Additionally, some other classes of STRIPS problems solvable as different TSP variants might be found.

Appendix A

Bibliography

Alcázar, V., Borrajo, D., Fernandez, S., and Fuentetaja, R. (2013). Revisiting regression in planning. pages 2254–2260.

Applegate, D., Bixby, R., Chvatal, V., and Cook, W. (2006). Concorde tsp solver.

Applegate, D., Bixby, R., Chvátal, V., and Cook, W. (2011). *The Traveling Salesman Problem: A Computational Study*. Princeton Series in Applied Mathematics. Princeton University Press.

Ascheuer, N., Jünger, M., and Reinelt, G. (2000). A branch and cut algorithm for the asymmetric traveling salesman problem with precedence constraints. *Computational Optimization and Applications*, 17(1):61–84.

Bang-Jensen, J., Gutin, G., and Yeo, A. (2004). When the greedy algorithm fails. *Discrete Optimization*, 1(2):121 – 127.

Bonet, B. and Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence. 2001 Jun; 129 (1-2): 5-33.*

Bylander, T. (1994). The computational complexity of propositional strips planning. *Artificial Intelligence*, 69:165–204.

Carpaneto, G. and Toth, P. (1987). Primal-dual algrorithms for the assignment problem. *Discrete Applied Mathematics*, 18(2):137–153.

Castelino, K., D'Souza, R., and Wright, P. K. (2003). Toolpath optimization for minimizing airtime during machining. *Journal of Manufacturing Systems*, 22(3):173–180.

Charikar, M., Motwani, R., Raghavan, P., and Silverstein, C. (1997). Constrained tsp and low-power computing. In Dehne, F., Rau-Chaplin, A., Sack, J.-R., and Tamassia, R., editors, *Algorithms and Data Structures*, pages 104–115, Berlin, Heidelberg. Springer Berlin Heidelberg.

Christofides, N. (1976). Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University.

Croes, G. A. (1958). A method for solving traveling-salesman problems. *Operations Research*, 6(6):791–812.

Dantzig, G., Fulkerson, R., and Johnson, S. (1954). Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410.

Deng, Y., Liu, Y., and Zhou, D. (2015). An improved genetic algorithm with initial population strategy for symmetric tsp. *Mathematical Problems in Engineering*, 2015.

Edelkamp, S. and Helmert, M. (2000). Exhibiting knowledge in planning problems to minimize state encoding length. In Biundo, S. and Fox, M., editors, *Recent Advances in AI Planning*, pages 135–147, Berlin, Heidelberg. Springer Berlin Heidelberg.

Edmonds, J. (1967). Optimum branchings. *Journal of Research of the national Bureau of Standards B*, 71(4):233–240.

Fikes, R. E. and Nilsson, N. J. (1971). Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3):189 – 208.

Fischetti, M., Lodi, A., and Toth, P. (2007). *Exact Methods for the Asymmetric Traveling Salesman Problem*, pages 169–205. Springer US, Boston, MA.

Fišer, D. and Komenda, A. (2018). Fact-alternating mutex groups for classical planning. *Journal of Artificial Intelligence Research*, 61:475–521.

Gouveia, L. and Pires, J. M. (1999). The asymmetric travelling salesman problem and a reformulation of the miller–tucker–zemlin constraints. *European Journal of Operational Research*, 112(1):134–146.

Gouveia, L. and Pires, J. M. (2001). The asymmetric travelling salesman problem: on generalizations of disaggregated miller–tucker–zemlin constraints. *Discrete Applied Mathematics*, 112(1):129–145. Combinatorial Optimization Symposium, Selected Papers.

Gutin, G., Yeo, A., and Zverovich, A. (2002). Traveling salesman should not be greedy: Domination analysis of greedy-type heuristics for the tsp. *Discrete Applied Mathematics*, 117:81–86.

Haslum, P. and Geffner, H. (2000). Admissible heuristics for optimal planning. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*, AIPS'00, page 140–149. AAAI Press.

Helmert, M. (2006). The fast downward planning system. *J. Artif. Int. Res.*, 26(1):191–246.

Helmert, M. (2009). Concise finite-domain representations for pddl planning tasks. *Artificial Intelligence*, 173(5):503 – 535. Advances in Automated Plan Generation.

Helmert, M. and Domshlak, C. (2009). Landmarks, critical paths and abstractions: what's the difference anyway? In *Nineteenth International Conference on Automated Planning and Scheduling*.

Helmert, M., Haslum, P., and Hoffmann, J. (2007). Flexible abstraction heuristics for optimal sequential planning. In *Proceedings of the Seventeenth International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'07, page 176–183. AAAI Press.

Helmert, M., Haslum, P., Hoffmann, J., and Nissim, R. (2014). Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM*, 3.

Helsgaun, K. (2000). An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130.

Helsgaun, K. (2017). An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. Technical report.

Huang, R., Chen, Y., and Zhang, W. (2012). Sas+ planning as satisfiability. *Journal of Artificial Intelligence Research*, 43:293–328.

Ilavarasi, K. and Joseph, K. S. (2014). Variants of travelling salesman problem: A survey. In *International Conference on Information Communication and Embedded Systems (ICICES2014)*, pages 1–7.

Jablonsky, J. (2015). Benchmarks for current linear and mixed integer optimization solvers. *Acta Universitatis Agriculturae et Silviculturae Mendelianae Brunensis*, 63:1923–1928.

Jonker, R. and Volgenant, T. (1983). Transforming asymmetric into symmetric traveling salesman problems. *Operations Research Letters*, 2(4):161–163.

Karp, R. M. (1972). *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA.

Katz, M. and Hoffmann, J. (2014). Mercury planner: Pushing the limits of partial delete relaxation. *IPC 2014 planner abstracts*, page 43–47.

Kumar, R. and Li, H. (1996). On asymmetric tsp: Transformation to symmetric tsp and performance bound. *Journal of Operations Research*, 47(6).

Lin, S. (1965). Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44(10):2245–2269.

Lin, S. and Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516.

Liu, F. and Zeng, G. (2009). Study of genetic algorithm with reinforcement learning to solve the tsp. *Expert Systems with Applications*, 36(3, Part 2):6995 – 7001.

Miller, C. E., Tucker, A. W., and Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. *J. ACM*, 7(4):326—-329.

Noon, C. E. and Bean, J. C. (1993). An efficient transformation of the generalized traveling salesman problem. *INFOR: Information Systems and Operational Research*, 31(1):39–44.

Pommerening, F., Röger, G., Helmert, M., and Bonet, B. (2014). Lp-based heuristics for cost-optimal planning. In *Twenty-Fourth International Conference on Automated Planning and Scheduling*, pages 226–234.

Razali, N. M., Geraghty, J., et al. (2011). Genetic algorithm performance with different selection strategies in solving tsp. In *Proceedings of the world congress on engineering*, volume 2, pages 1–6. International Association of Engineers Hong Kong.

Reinelt, G. (1995). Tsplib 95. Technical report, Interdisziplinäres Zentrum für Wissenschaftliches Rechnen (IWR). (Manuscript and problem instances available at http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp95.pdf).

Salman, R. (2015). Algorithms for the precedence constrained generalized travelling salesperson problem. Master's thesis, Chalmers University of Technology. University of Gothenburg.

Sarin, S. C., Sherali, H. D., and Bhootra, A. (2005). New tighter polynomial length formulations for the asymmetric traveling salesman problem with and without precedence constraints. *Operations Research Letters*, 33(1):62–70.

Seipp, J., Pommerening, F., and Helmert, M. (2015). New optimization functions for potential heuristics. In *Proceedings of the Twenty-Fifth International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'15, page 193–201. AAAI Press.

Shobaki, G. and Jamal, J. (2015). An exact algorithm for the sequential ordering problem and its application to switching energy minimization in compilers. *Computational Optimization and Applications*, 61(2):343–372.

Sievers, S., Wehrle, M., and Helmert, M. (2016). An analysis of merge strategies for merge-and-shrink heuristics. In *Proceedings of the Twenty-Sixth International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'16, page 294–298. AAAI Press.

Álvaro Torralba, Alcázar, V., Borrajo, D., Kissmann, P., and Edelkamp, S. (2014). Symba*: A symbolic bidirectional a* planner. *International Planning Competition 2014*, page 105–108.

Appendix B

CD Contents

1. *domain-checker.zip* - Reducibility detector - Program that detects whether the problem is TSP-reducible or PC-GTSP-reducible.
2. *visitall-solver.zip* - Visit-All TSP solver - Solver using the TSP-reducible problem encoding to solve the Visit-All problem
3. *pcgtsp-solver.zip* - OpenStacks PC-GTSP solver - Solver using the PC-GTSP-reducible problem encoding to solve the OpenStacks problem