



**CZECH TECHNICAL
UNIVERSITY
IN PRAGUE**

**Faculty of Electrical Engineering
Department of Control Engineering**

Master's thesis

Reactive scheduling of pickup-and-delivery tasks for industrial mobile robots

Bc. Lucie Halodová

August 2021

Supervisor: Ing. Bc. Lenka Mudrich, Ph.D.

I. Personal and study details

Student's name: **Halodová Lucie** Personal ID number: **457087**
Faculty / Institute: **Faculty of Electrical Engineering**
Department / Institute: **Department of Control Engineering**
Study program: **Cybernetics and Robotics**
Branch of study: **Cybernetics and Robotics**

II. Master's thesis details

Master's thesis title in English:

Reactive scheduling of pickup-and-delivery tasks for industrial mobile robots

Master's thesis title in Czech:

Reaktivní rozvrhování „nalož-a-přivez“ úloh pro průmyslové mobilní roboty

Guidelines:

1. Familiarise yourself with:
 - a. the Pickup and Delivery problem with Time Windows (PDP-TW) [1]
 - b. failures occurring during execution of schedules addressing PDP-TW in a real industrial setting.
 - c. planning languages PDDL 2.1. [2] and PDDL 3.1. [3] and discuss which one is more suitable for modeling PDP-TW.
 - d. existing planners, such as TFD [4] and OPTIC [5].
2. Given a set of initial schedules, propose a system which would amend these schedules in reaction to failures during execution.
3. Evaluate proposed scheduling system for use in simulated industrial environments.

Bibliography / sources:

- [1] M. Desrochers, J. K. Lenstra, M. W. P. Savelsbergh and F. Soumis - Vehicle routing with time windows optimization and approximation - Vehicle Routing Methods and Studies - Elsevier Science Publisher B.V. (North Holland), 1988
[2] M. Fox and D. Long - PDDL2.1 : An Extension to pddl for Expressing Temporal Planning Domains - Journal of Artificial Intelligence Research 20 (2003) 61-124
[3] A. Gerevini and D. Long - Plan Constraints and Preferences in PDDL3, Technical Report, Department of Electronics for Automation, University of Brescia, Italy, August 2005
[4] P. Eyerich, R. Mattmüller and G. Röger - Using the Context-enhanced Additive Heuristic for Temporal and Numeric Planning - In Proceedings of the 19th International Conference on Automated Planning and Scheduling, 2009.
[5] J. Benton, A. Coles and A. Coles - Temporal Planning with Preferences and Time-Dependent Continuous Costs - In Proceedings of the 22nd International Conference on Automated Planning and Scheduling, 2012.

Name and workplace of master's thesis supervisor:

Ing. Bc. Lenka Mudrich, Ph.D., DataVision s.r.o., Ukrajinská 2a, Praha 10

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

Ing. Martin Hlinovský, Ph.D., Department of Control Engineering, FEE

Date of master's thesis assignment: **15.01.2021** Deadline for master's thesis submission: **13.08.2021**

Assignment valid until:

by the end of winter semester 2022/2023

Ing. Bc. Lenka Mudrich, Ph.D.
Supervisor's signature

prof. Ing. Michael Šebek, DrSc.
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 her 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



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 on August 13, 2021

.....
Lucie Halodová

Acknowledgment

I would like to thank all the people who supported me during this theses. My deepest gratitude goes to my supervisor Ing. Bc. Lenka Mudrich for her guidance through such a complex topic, her huge expertise in task planning and scheduling, and her patience with me. I really appreciate her individual and human approach and thank her for the amount of knowledge I could gain under her supervision.

I would also like to thank the Technology Agency of the Czech Republic for accepting the project *Guidance and Localization upgrade creating Autonomous Mobile Robots* under the TREND Programme FW03010020 and to Datavision company for allowing me to work on such an exciting project in this thesis.

Besides, I would like to thank my friends and family for their support during my whole studies. Namely, to Jitka Hodná, a partner in crime, who shared my most beautiful and the most difficult moments during my studies; to Matouš Pokorný, father Fura and the best coworker I met, for all his always great advice and reducing my stress; and to Ewan Dickson, Václav Veselý and Maťa Dubeňová for their willingness to help and correct my grammatical mistakes in the thesis.

Abstrakt

Tato práce se zabývá selháními, které nastávají při vykonávání plánů specifikovaných operátorem v industriálních prostředích jako například tam, kde autonomní pozemní vozidla převážejí zboží. Hlavním problémem je, že umělá inteligence řeší selhání neočekávatelnými způsoby pro lidské chápání, a firmy se ji proto zdráhají používat. Pro řešení tohoto problému je v diplomové práci navržena metoda opravy plánů, která využívá co nejvíce akcí z operátorova plánu a modifikuje pouze tu část, která souvisí se selháním. Tento přístup je porovnaný s přeplánováním od začátku, ve kterém je operátorův plán modifikovaný od momentu, kdy se selhání objeví. Oba přístupy jsou zhodnoceny na sadě scénářů popisujících, kdy může nastat selhání robota nebo selhání kvůli zablokované cestě v příkladovém industriálním prostředí. Experimentální výsledky potvrdily, že metoda opravy plánů dává plán, který je málo odlišný od operátorova plánu. Případová studie dochází k závěru, že oprava plánů by se měla preferovat před přeplánováním od začátku, pokud je kritickým požadavkem malý rozdíl od operátorova plánu. Na rozdíl od toho, přeplánování od začátku by se mělo preferovat před opravou plánů, pokud se žádá, aby zpoždění modifikovaného plánu a průměrné zpoždění doručení nákladu byly malé.

Klíčová slova: oprava plánů, přeplánování od začátku, selhání při vykonávání, PDDL, plánování úloh.

Abstract

This thesis deals with failures occurring during the execution of plans specified by an operator in industrial environments such as those where autonomous ground vehicles are transporting goods. The main problem is that artificial intelligence resolves failures in unexpected ways for humans so that companies are reluctant to use it. To address this problem, this thesis proposes a plan repair method, which reuses as many actions of the operator's plan as possible and modifies only the affected part related to failures. This approach is compared with replanning from scratch, in which the operator's plan is modified from the moment when a failure appears. Both approaches are evaluated on a set of scenarios describing when a failure of a robot or a failure due to a blocked path occurs in an example industrial environment. Experimental results confirmed that the plan repair method provides a plan with a small difference from the operator's plan. The case study concludes that plan repair should be preferred over replanning from scratch if a small difference from the operator's plan is crucial. In contrast, replanning from scratch should be preferred over plan repair if the performance demands, such as the delay of the modified plan or an average delay of a cargo delivery, are required to be small.

Keywords: plan repair, replanning from scratch, failures in execution, PDDL, task planning.

Contents

1	Introduction	2
1.1	Requirements specified by the REX project	3
1.2	Motivation example	4
1.3	Organization of work	6
2	State of the art	7
2.1	Routing problems	7
2.1.1	Arc routing problems	7
2.1.2	Vehicle routing problems	8
2.1.2.1	Traveling salesman problem	8
2.1.2.2	Vehicle routing problem	9
2.1.2.3	Variants of VRP	10
2.1.2.4	Pickup-and-delivery problem	11
2.2	Failures	12
2.3	Planning	13
2.3.1	Classical planning	14
2.3.2	Planning Domain Definition Language	15
2.3.2.1	PDDL 1.2	15
2.3.2.2	PDDL 2.1	17
2.3.2.3	PDDL 2.2	18
2.3.2.4	PDDL 3.0	18
2.3.2.5	PDDL 3.1	18
2.3.2.6	Discussion of suitability for modelling PDP- TW	19
2.4	Planners	19
2.4.1	Temporal Fast Downward (TFD)	20
2.4.2	OPTIC	21
2.5	Plan modification	21

3	Problem definition	25
3.1	Problem background	25
3.2	Problem statement	26
3.3	Solution approaches	26
4	Solution	27
4.1	Minimal plan repair	27
4.1.1	Model of the world	27
4.1.1.1	Modelling of PDDL domain	27
4.1.2	Proposed methods	31
4.1.2.1	Plan repair	31
4.1.2.2	Replanning from scratch	38
4.2	System	39
4.2.1	The REX system	39
4.2.2	Integration of plan modification	40
4.2.2.1	Implementation	41
5	Evaluation	45
5.1	Testing architecture	45
5.2	Generation of data	46
5.2.1	Operator's problem file	47
5.2.1.1	Analysis of planning problem describing an industrial environment	47
5.2.1.2	Initial problem file	50
5.2.1.3	Initial PDDL plan	50
5.2.2	Testing scenarios	51
5.2.2.1	Generation of AGV failure	52
5.2.2.2	Generation of failure due to blocked path	53
5.3	Metrics description	55
5.3.1	Plan difference	55
5.3.2	Total plan delay	56
5.3.3	Average cargo delivery delay	56
5.4	Experimental results	56

5.4.1	Plan difference	57
5.4.2	Total plan delay	59
5.4.3	Average cargo delivery delay	62
5.5	Summary and discussion	64
6	Conclusion	66
6.1	Future work	67
	References	70
A	Attached code	74
A.1	PDDL domain	74
A.2	Initial PDDL problem	75
A.3	Initial PDDL plan	78
B	Graphs	79

List of Figures

1	Possible usage of the REX ecosystem [1]	4
2	Testing scenario pictured as a topological map	5
3	The seven bridges of Königsberg [2]	8
4	Optimal tours for large number of instances in TSP [3]	9
5	Hierarchy of objects used in the domain	28
6	Plan repair method. Violet rectangular blocks are forming the plan repair method. Elliptical blocks represent data.	38
7	The REX ecosystem	40
8	Overview of the REX system relevant to the plan repair	42
9	Simplified architecture of the system for testing purposes	46
10	Two extreme configurations of waypoints	48
11	Visualization of plans of individual AGVs. The red colour marks paths required to deliver the first assigned cargo, the yellow colour to deliver the second assigned cargo and the blue colour denote paths necessary to get to the docking centre.	51
12	Plan difference	57
13	Total plan delay	60
14	Average cargo delivery delay	62
15	Plan difference for all testing scenarios	80
16	Total plan delay for all testing scenarios	81
17	Average delay of a cargo delivery for all testing scenarios	82



List of Tables

1	Results from a set of problems with 3 AGVs, 10 cargos and an increasing number of waypoints from 3 to 12 in configuration <i>all</i> and <i>line</i> run on OPTIC and TFD planners.	49
2	Results from a set of problems with 3 AGVs, 20 cargos and an increasing number of waypoints from 3 to 22 in configuration <i>all</i> and <i>line</i> run on OPTIC and TFD planners.	49
3	Results of statistical values for <i>plan difference</i> for all testing scenarios	59
4	Results of statistical values for <i>total plan delay</i> for all testing scenarios	62
5	Results of statistical values for <i>average cargo delivery delay</i> for all testing scenarios	64

Listings

1	Predicates used in the PDDL domain	28
2	Action <i>load</i> in the PDDL domain	29
3	Action <i>unload</i> in the PDDL domain	30
4	Action <i>drive</i> in the PDDL domain	30
5	PDDL domain	74
6	Initial PDDL problem	75
7	Initial PDDL plan	78

List of Algorithms

1	refine_goals($G_{disturbed}$)	33
2	replay_plan($\pi, s_{init}, L_{failures}$)	34
3	get_current_state(π, s_{init})	34
4	simulate_future($\pi, s_{current}$)	35
5	join_plans($\pi_{operator}, \pi_{new}, O_w$)	36
6	Repair plan	37



Chapter 1

Introduction

The fourth industrial revolution has already started in the most developed parts of the world. It proposes the vision of two connected worlds - the world of real physical objects and the virtual world representing these objects. This revolution brings a whole new philosophy changing the overall society. The affected areas range from the industry, over the technical standardization, safety, educational system, law regulations, science and research, labour market up to social impacts. The consequences of the fourth industrial revolution led to the formulation of the Industry 4.0 Initiative [4, 5]. The aim of this initiative is to achieve and strengthen the long-term competitiveness of the Czech Republic on a global scale.

Industrial production forms the centre of this fourth industrial revolution, and hence, it is also called Industry 4.0. Each part of the production is represented as a complex distributed system consisting of many autonomous subsystems. The smart integration of these self-sufficient subsystems does not need any central hierarchical control because every unit is able to communicate with each other. However, the whole production system without a central controlling authority is still an extreme case nowadays. These modular systems will be connected through standardized communication protocols ensuring the openness of the entire system for variable subsystems and their reusability in other systems.

Industry 4.0 is also connected with smart factories. The basic elements of the smart factories are cyber-physical systems, which will be able to communicate with each other, exchange information, analyze data in order to predict for example failures and adapt in real-time to the changing conditions. The production process will be optimized and able to react to the unexpected changes caused for example by a failure of a robot.

The first prerequisite for the application of the Industry 4.0 is the digitalization of the production processes and the collection of the data for subsequent usage. The Industry 4.0 has the vision of the vertical integration of all subsystems from the lowest level of automatization, over the management of the production to the planning of the company resources by ERP systems (Enterprise Resource Planning). The automatization of processes with the robotization of the environments further requires the Internet of Things technologies. Finally, the last steps are covered by the research in the area of cybernetics and artificial intelligence to autonomously plan the production processes or suggest the optimal logistic routes. The level of companies' readiness for Industry 4.0

can differ a lot.

This thesis is assigned by the Czech company Datavision s.r.o. which designs and delivers solutions to industrial environments. The thesis is motivated by one of the current projects of this company, called REX. This project is concerned with the creation of the intelligent system for the fleet of cooperating mobile robots in accordance with the concepts of Industry 4.0. The REX system will be able to accept requests of pick-up and delivery tasks, autonomously plan and schedule those tasks to robots. It will also be able to change the plan based on an occurrence of unexpected events such as a change of the environment or a failure of a robot. Therefore, the aim of this thesis is to design a module capable of resolving failures by autonomous repairing of plans.

The proposed system in this thesis will introduce a minimal amount of artificial intelligence (AI) in order to resolve failures. This approach is needed because many companies with existing manufacturing processes are reluctant to use AI because they do not believe in it enough. They still use scheduled plans by experienced operators and resolve failures by themselves. If we can design such an AI system which would find simple, well-understanding changes to the plan given by an operator, we can persuade companies to trust and use such a system.

The minimal amount of AI will modify the operator's plan as little as possible. Hence, the proposed method will change only the part of the plan affected by failures and let most of the original part unchanged. In this way, the unaffected part of the plan can continue to execute, and during this time, the faulty part can be repaired. If the flawless part of the plan is assumed to be finished, our system wrapping a state of the art planner finds a plan achieving goals of the affected part of that plan and joins these plans together. Therefore, the resulting plan preserves as much as possible from the operator's plan.

The significant benefit of this approach is that the production does not have to stop. In existing manufacturing processes, the production has to be stopped until failures are resolved, resulting in a huge loss of profit. Thanks to the repairing process during the execution, our system does not block the production until a new plan is provided. Consequently, the loss of profit is significantly suppressed.

■ 1.1 Requirements specified by the REX project

The thesis is constrained by the specification of the REX project. Thus, the aim of this section is to provide an insight into REX specification related to this thesis assignment.

The REX system will ensure the fleet management of the *autonomous mobile robots (AMR)* including their control and planning. Thanks to the modularity of the system, the fleet can contain various mobile platforms from different vendors. The heterogeneous fleet can consist of *autonomous ground vehicles (AGV)* already used in factories as well as new AGVs, AMRs, drones or platforms controlled by people like forklifts or tractors, which can be localized and the information used for the planning. To support the flexibility of the factory, the production line can be composed of several AGV. The vision of the possible usages of the REX system is depicted in Figure 1.

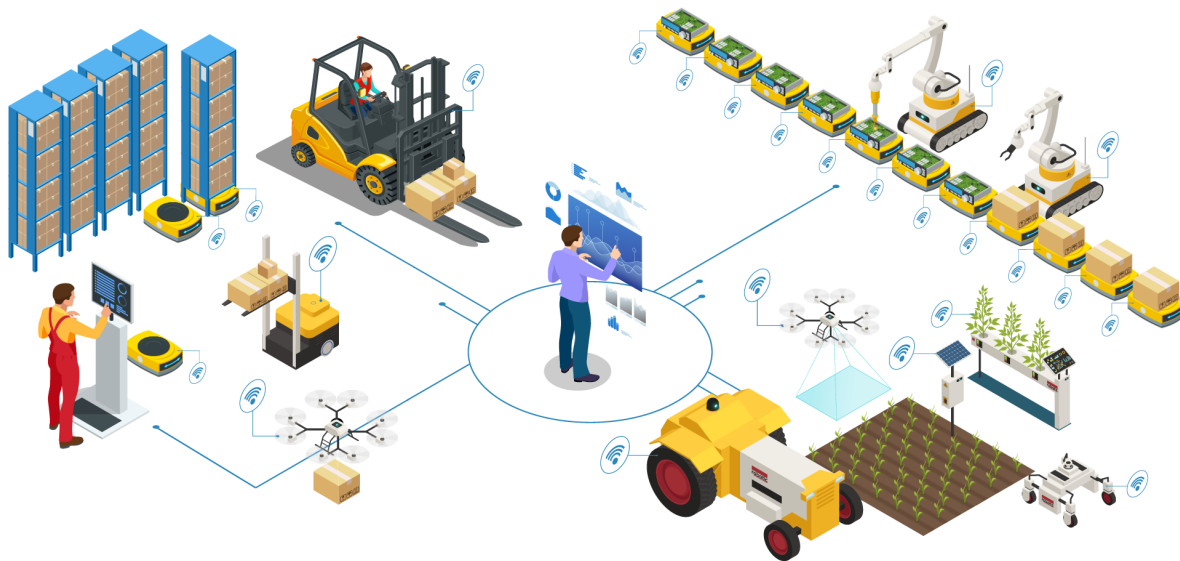


Figure 1: Possible usage of the REX ecosystem [1]

The most prevalent failures occurring in factories are usually of three types - failure of a robot, blocking off a path and change of time requirements. This thesis will focus only on AGV and the first two types of failures.

Such a system can be applied in various branches of the industry related to the production with storages and logistics. Particularly in automotive, which is one of the key branches of the Czech industry, and thus it is advisable to deploy the system first there, but also in agriculture or healthcare. REX can help during pandemics like COVID-19 as well. One possible scenario is a field hospital, where infected samples can be transported autonomously without infecting other people. Moreover, fewer people will have to work in factories, or they will be allowed to work remotely. Consequently, people would meet less and reduce the risk of the transmission of the infection.

This project is called *Guidance and Localization upgrade creating Autonomous Mobile Robots* and is co-financed by the Technology Agency of the Czech Republic under the TREND Programme FW03010020.

■ 1.2 Motivation example

The problem addressed in this thesis is illustrated in the following motivational example, which is used throughout the rest of the work.

The testing scenario is situated into a factory concerning production and logistics of its material, motivated by a real factory setting such as one owned by the Skoda

company ¹. In such a factory, the fleet of robots autonomously delivers material to conveyor belts. The fleet consists of three homogeneous robots that are able to pick a cargo at a storage centre and deliver it to a conveyor belt, where the material is processed. Robots can park and recharge in a docking centre. The task of a robot is to drive from the docking centre to the storage centre and pick up a requested material. Then it should drive to a requested location at a conveyor belt, drop the material and return back to the docking station.

The scenario is represented on a topological map, see Figure 2. The weighted graph consists of nine nodes denoting locations connected by edges indicating bidirectional paths weighted by a distance between nodes. In the map, there are two conveyor belts with six locations coloured blue intended for a robot to drop the requested material. The storage centre coloured as a green node serves as a pick-up location for a material, further referred to as cargo. The orange node marks the docking centre, and the grey node is only a crossroad allowing a robot to change the path, including turning back along the same path. Each node, where two or more paths meet, also serves as a crossroad.

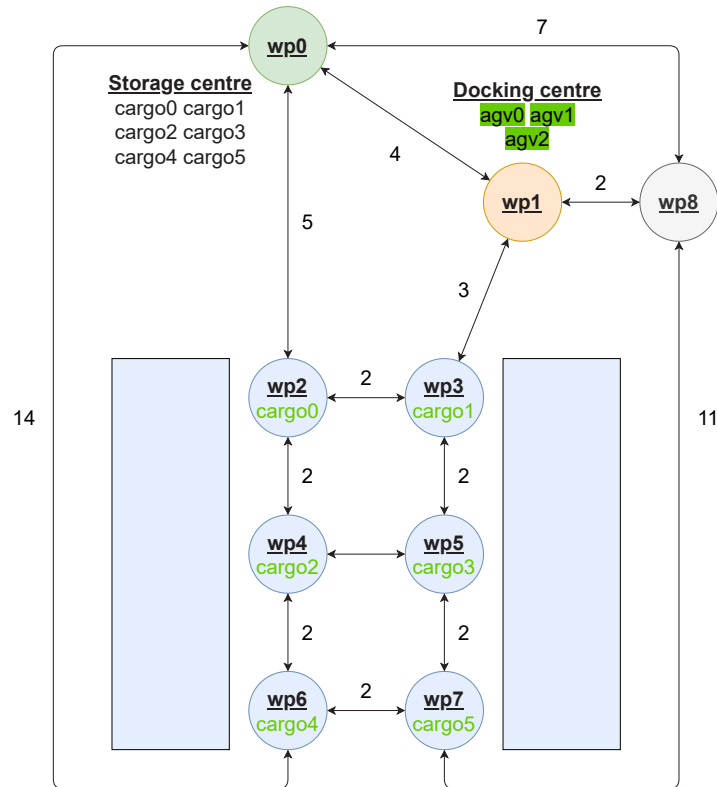


Figure 2: Testing scenario pictured as a topological map

The reason for such a simple scenario is threefold. Firstly, only three robots are sufficient to show a failure of a robot. These robots are modelled as homogeneous

¹See the video introducing the current technology used in Skoda factory: <https://www.youtube.com/watch?v=5C-mCFnzQLU>


because the heterogeneity would not influence our approach and thus can be neglected. Secondly, the map is composed of enough nodes and edges to investigate a failure due to a blocked path. Thirdly, the size of the example allows us to present the repair process to readers and easily validate its correctness and behaviour.

If a failure of a robot occurs, the rest of the fleet will serve the goals of this faulty robot. Consider a robot called *agv0*. This robot should deliver *cargo0* to the location *wp0* and *cargo3* to location *wp5*. Suppose a failure of this robot arises, for example, before its goals are served. In that case, the rest of the fleet (i.e. *agv1* and *agv2*) will split the goals of *agv0* robot equally and achieve them after the successful accomplishment of their own scheduled plan.

If a failure due to a blocked path happens, the plan of influenced robots will be modified to drive along different paths. Consider the robot *agv0* from the previous paragraph. The robot starts its journey at location *wp1*. Suddenly a path between locations *wp1* and *wp0*, where the robot should drive to the storage centre, becomes untraversable. Therefore, the plan of this robot is modified to drive from *wp1* to *wp8* and from *wp8* to *wp0* to get to the storage center to be able to achieve its goals.

■ 1.3 Organization of work

The structure of the thesis is divided into six chapters. The following chapter introduces the plan modification for which the relevant topics regarding vehicle routing problems, failures in execution, planning in PDDL language and corresponding planners are also described. In chapter three, the solved problem is stated in more detail. Chapter four is dedicated to the proposed method for modification of plans when failures in execution occur. The integration of this method into the REX system is also included in this chapter. The proposed system is evaluated compared to the operator's plan. The thesis is concluded in chapter six, providing an overview of the work done and describing possible future steps.



Chapter 2

State of the art

The aim of this chapter is to provide an overview of topics related to repairing plans in automated industrial environments.

Firstly, routing problems are discussed to gain knowledge about optimal logistic routes. Further, failures typically occurring in industrial environments are investigated. The PDDL language represents the plans, and therefore its variants are examined. The suitability of PDDL versions for modelling a pickup and delivery problem with time windows is discussed. Planner supporting PDDL languages are review. Finally, the main focus is on the ways how plans can be modified.

■ 2.1 Routing problems

Routing problems deal with the question of how to decide which routes to use and in which order to fulfil the required task. Routing problems can be divided from the viewpoint of the graphs into two groups. The first group is *arc routing problems* [6] which are based on the Eulerian graphs, and the second one is *vehicle routing problems* [7] that are based on the Hamiltonian graphs. There are also *general routing problems* [8], which are the generalization of these two.

■ 2.1.1 Arc routing problems

In arc routing problems, requests are dispersed along the arcs. These problems are based on the Eulerian graphs. In these graphs, all edges are traversed, but none is repeated.

Eulerian graphs were developed based on the *Königsberg Bridges Problem* formulated by Leonhard Euler in the 18th century. This is also connected with the origins of the graph theory [2]. The problem is formulated as follows. In the town of Königsberg, seven bridges are crossing two branches of the river as depicted in Figure 3. The question is whether a person can walk over each bridge exactly once. Leonhard Euler proved that no such solution exists for this problem [9]. The bridges here are the edges, all of which should be traversed but at most once.

2.1 ROUTING PROBLEMS

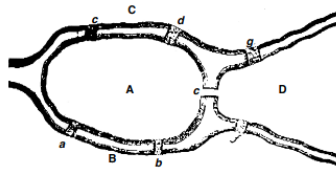


Figure 3: The seven bridges of Königsberg [2]

One of very famous applications of Eulerian graphs is a *Chinese Postman Problem* [10, 11]. This problem is inspired by a postman who has to go through all roads in his district before returning to the post office. The question is how to do it while walking the smallest number of kilometres. The problem can be formulated as a graph in which crossroads are the vertices and roads are the edges evaluated by the number of kilometres. Again, the postman has to traverse all edges but to minimize the number of kilometres, none of the edges should repeat. However, this sometimes can not be fulfilled to satisfy the requirements, and therefore, some edges are allowed to repeat, but the number of kilometres should be as low as possible.

Arc routing problems are applied for example in mail delivery, garbage collecting or street services such as street sweeping, salt gritting or snow ploughing. For more information about the arc routing problem, refer to the book [6].

■ 2.1.2 Vehicle routing problems

Requests in these problems are placed at the vertices. They are based on the Hamiltonian graphs, which are graphs in which vertices and edges do not repeat, but all vertices have to be visited, and it is allowed or required to end up in the starting point.

Finding a Hamiltonian graph is much harder than an Eulerian graph [12]. Finding an Eulerian graph in its most simple version belongs to the class P , which means that the solution can be found in the polynomial time [6]. Whereas finding a Hamiltonian graph belongs to the class NP , meaning there is not known any algorithm that would solve it in a polynomial time [11].

The travelling salesman problem is first introduced in the following sections, then the vehicle routing problem and its variants. Finally, the description of the pickup and delivery and its variant with time windows is described.

■ 2.1.2.1 Traveling salesman problem

One of the very famous applications of Hamiltonian graphs is the *Traveling salesman problem (TSP)* [13]. In this problem, a salesman wishes to visit a number of cities and then to return back to the home city. The question is how to find the order of these cities so that the time or distance is minimized. In this problem, cities are the vertices and roads among them are the edges evaluated by the number of kilometres.

The salesman has to visit all vertices in the graph, but none of them should repeat as well as the edges, and the number of kilometres should be minimized.

An optimal tour was found already in 1954 for 49 cities in the USA [13], see Figure 4a. Researchers continually upgraded their methods, and in 2004, an optimal tour for 24 978 cities in Sweden was found. It is considered as the largest solution applied on cities [14], see Figure 4b. However, the largest number of TSP instances for which the optimal tour was found is 85 900 in a chip design application. The computation time of this record is very long, over 136 CPU years with total running time of computer platforms over 568 hours [15].

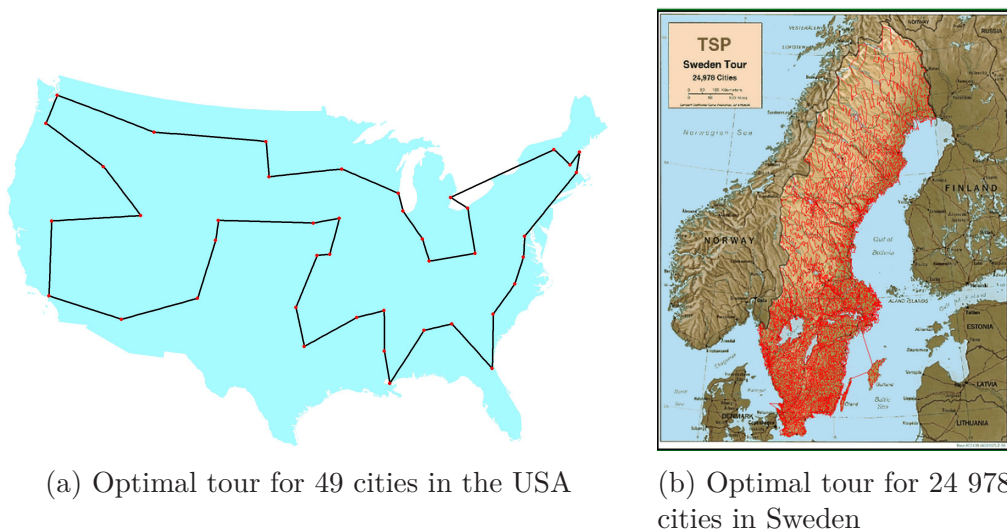


Figure 4: Optimal tours for large number of instances in TSP [3]

■ 2.1.2.2 Vehicle routing problem

The *Vehicle Routing Problem (VRP)* is a generalization of the Traveling Salesman Problem. In this problem, requests are no longer satisfied by one person, the salesman, but by a fleet of vehicles. This extension results in a more complex problem in which the concern is not only about the order in which the places should be visited but also about which vehicle should fulfil which requirement. If this is known in advance, then it reduces to multiple TSPs. However, there are many applications in the real world in which no constraints are given such that the assignment of vehicles to requirements is fixed. This class of problems focuses on both the assignment and the sequencing.

In the classical VRP, all vehicles start in the depot and then return to the same place. Each vehicle performs exactly one route. The goal is to serve customers placed in the vertices of a graph in the way that the total routing cost is minimized. Each customer is served exactly once.

2.1 ROUTING PROBLEMS

■ 2.1.2.3 Variants of VRP

The class of VRPs is very broad, including many variants that aim to fill the gap in real-world applications. Some differences among the variants are so slight that some papers concerning the same problem are named according to different variants of VRP.

VRP was first introduced in 1959 as the Truck Dispatching Problem [16]. In this problem, an optimal route for a fleet of trucks delivering gasoline is sought so that the total number of kilometres of this fleet is minimized. However, the trucks have a limited capacity that can be transported and therefore, the constraint on the capacity is also discussed. Considering the capacity constraint forms a variant of VRP called a *capacitated vehicle routing problem (CVRP)*, which is the most basic variant of VRP and is widely studied. Customers' demands then represent the amount to be delivered.

Variants of VRP can be classified according to the decisions that have to be considered [17, 7]. In *VRPs with profits*, it might be impossible to serve all customers, and thus, a decision has to be made whether to serve a customer with respect to the overall profit. Unserved customers can be penalized or served customers rewarded. *VRPs with split deliveries* indicates a group in which customers can be visited more than once because their demands can be served by multiple vehicles. In *VRPs with multiple commodities*, commodities can be in different locations and delivered only by some vehicles. Deliveries from suppliers to customers can use intermediate transfer points. This problem is referred to as *combined shipments*. Vehicles that can be reused are addressed by *VRPs with multiple use of vehicles*, but then recharging or refuelling has to be taken into account. If vehicles start or end their routes in different depots, then the problem is called a *Multiple depot VRP*.

Several other variants are classified according to the common characteristics that can be added to the aforementioned variants [7]. As for the fleet characteristics, the vehicles can be identical and thus, the fleet is *homogeneous* or the vehicles can vary in capacity, costs, speeds or accessibility to customers and thus, the fleet is *heterogeneous*. In *dynamic VRPs*, some information such as customers' locations or demands become available during the execution. The occurrence of vehicle breakdowns or delays is also considered a dynamic, and the rescheduling of routes is required. In *stochastic VRPs*, some information such as customers' demands or travel times are uncertain and can be described as random variables. The impact of these uncertainties on the cost is analyzed. *Open VRPs* are VRPs in which vehicles do not have to return to the depot. If some services are indirect, it forms problems in which locations close enough to customers are visited. *Vehicle scheduling problems* refers to problems in which customers or locations are only visited, e.g. routing of public transport. One of the very important variants is also a *pickup-and-delivery problem*, which is discussed in more detail in the next section.

Another characteristic is a repeating pattern like deliveries of goods that repeat over time. For example, in *periodic VRP*, the goods are delivered on a visiting pattern such as every Monday. Repeating of supplies is also discussed in *Inventory routing problems*, in which retailers share their inventory with vendors who can decide about suitable supplies. The application is, for instance, in deliveries to supermarkets or fuel delivery

to gas stations.

Furthermore, VRPs can also be constrained by *time windows* considering when the route can start or end. In the *VRP with time windows*, each edge has a traversal time, and each vertex has a time window, in whose time interval the service should be performed. If the vehicle comes earlier than in the time interval, it has to wait, whereas coming late is prohibited. If being late is allowed, then we call it *soft time windows*, which can be modelled by a penalization function. Multiple time windows are also possible. The periodic patterns or driving and rest periods can also be included.

In VRPs, various constraints can be taken into account. One of these constraints already mentioned is capacity and is addressed for example in CVRPs. There are also loading constraints modelling that for example food cannot be transported with washing powder or the order of goods if the vehicle can be loaded or unloaded only from one side. Other constraints can be on the length, duration or cost of the route, number of vehicles allocated to a specific depot or number of specific routes. There can be constraints also on synchronization of, for example, tasks or movements. When considering trucks and their trailers, the constraints, for example whether the trailer can be manoeuvred in a specific place, are discussed.

The objective can be to maximize the profit or customer satisfaction, minimize the total time or waiting times, distance, etc. Some of the objectives can be conflicting, e.g. minimizing the number of vehicles and the total time or distances. Therefore, the objectives are hierarchical, applied one after another.

■ 2.1.2.4 Pickup-and-delivery problem

Pickup-and-delivery problem (PDP) is a variant of VRP dealing with deliveries to customers and collections from them. Collections from customers are called pickups. Problems with collections are also known as *many-to-one* VRPs and with distributions as *one-to-many* VRPs.

From the viewpoint of transportation, two classes of problems can be distinguished. The first class deals with the transportation of goods from the depot to the so called linehaul customers and from the backhaul customers to the depot. These problems are also known as *one-to-many-to-one*, and are addressed mainly by *VRP with simultaneous pickup and delivery*. In this problem, both the delivery from the depot to the customer and pickup from the customer to the depot are performed by one vehicle in a single visit. An example can be the delivery of milk and a simultaneous collection of empty bottles. The route is built in the way that the customer with a higher amount of deliveries than pickups is visited first. A relaxation of this problem is a *VRP with divisible deliveries and pickups* in which deliveries and pickups can be done in two visits by the same vehicle. One of the problems in this class is also *VRP with backhauls*, in which all deliveries have to precede the pickups. Problems in which only pickups or deliveries, but not both, occur are known as *single demand*.

The second class deals with the transportation between linehaul customers and backhaul customers or also between pickup and delivery locations. These problems are also

2.2 FAILURES

known as *many-to-many* VRP because neither of these locations is a depot. This class of problems is called *VRP with pickups and deliveries*. In *one-to-one* PDP, the load is transported from one pickup vertex to one delivery vertex. Each commodity has only one origin and one destination instead of many origins and many destinations in many-to-many VRP. These problems are also known as *Pickup-and-delivery VRP*. In the context of people transporting, these problems are also called *dial-a-ride problem*, e.g. routing of school buses. Pickup and delivery points can also be paired.

Pickup and Delivery problem with Time Windows

As mentioned earlier, time windows can be hard or soft. In the case of *hard time windows*, the time interval cannot be violated, and therefore the vehicle has to wait to begin the service if it arrives earlier. Whereas in the case of *soft time windows*, the time interval can be violated and punished by a penalty cost. Time windows can also be one-sided, for example if only the delivery time is important.

Therefore, some routes in PDP or VRP are no longer considered feasible because they violate the time windows. In [7], the authors determine that time windows with wide time intervals are more difficult to solve because the number of feasible routes increases. However, infeasible solutions can be allowed and penalized, which makes it again more difficult. Generally, PDPs are harder to solve than VRPs due to the precedence constraints [7].

In [18], the authors solve a multi-pickup and delivery problem with time windows. Each vehicle has to perform multiple pickups and then the delivery in a single tour in this problem. The problem can be applied to companies for ordering food such as *Uber Eats* allowing the customer to order meals from several restaurants. The authors developed a new heuristic algorithm to solve the problem. An unpaired pickup and delivery problem solved in [19] is concerned with the air transportation of cargos and its flight itinerary decisions so that the cargos are delivered before the due date. In addition, [20] tackle recharging of autonomous mobile robots, whereas in [21] batteries are swapped.

For more information about VRP or PDP and their variants, refer to the book [7].

■ 2.2 Failures

This section discusses possible failures that can occur during the execution of schedules addressing PDP-TW in industrial environments. First, failures from the viewpoint of planning are considered, followed by failures from the industrial point of view.

According to [22], failures can be categorized into three groups with respect to the planning problem. In the first group, *weak failure* of the execution of the plan means that performing an action does not bring some of the expected positive effects. In the second group, *strong failure* of the execution of the plan means that the action cannot

be performed due to its inapplicability (i.e. some of the expected preconditions are not satisfied). Weak failures often lead to strong failures. The third group represents any other failures that can happen for example because the state is changed, but not by the agents. In this case, it depends on domains.

Several failures can appear during the execution of schedules dealing with PDP-TW and make some execution steps and their successors impossible. Considering the tasks for ground vehicles only because this thesis focuses on them, the three most frequent issues can happen.

The first of them is any type of vehicle's disorder. The reason for these defects is various. For example, low battery causing the low level and high level of a robot not responding to any orders; hardware-like breakdowns such as stuck motors or a technology used for pickups and deliveries of goods, and non-working computer controlling the robot. Platform-related malfunctions such as punctured tires and other mechanical wear and damage making the robot unable to fulfil the order.

There can also be external issues that might cause a fault of a vehicle. A typical example is network issues. A fleet management software is usually placed on a server and communicates with individual robots. If the connection is broken or there are any other problems with networking, the vehicle cannot satisfy any further tasks due to this.

External influences can affect the fulfilment of orders. These problems are directly related to industrial environments. The second group of most frequent failures represents issues when the robot cannot pass through a path assigned by the server. The reason can be an obstacle blocking the path or, based on the navigation technology, a magnetic tape which the robot follows is damaged, for instance. Similar issues can happen at pickup and delivery places. Moreover, the lack of resources in the pickup location makes the transportation of goods impossible.

The third most frequent type of failure is related to time demands. Transported material is scheduled in PDPs-TW. Thus, there can appear delays in deliveries of cargos. This fault can be caused because of the two aforementioned most frequent types of failures. Another reason can be the deceleration of vehicles when avoiding each other or on crossroads, depending on the navigation technique used. The majority of other reasons are connected with unsuitable schedules in which cargos are scheduled to be delivered late. However, these issues are related to the scheduling of PDP-TW, and thus, they are not intended to be fixed during the execution of given schedules. Time windows are not handled in this work because it creates a complexity that goes beyond the scope of the diploma thesis.

■ 2.3 Planning

Planning is a form of general problem-solving. There are three key components to be able to solve it. There are models for defining, classifying and understanding problems,

2.3 PLANNING

languages for representing problems and algorithms for solving them. This section will briefly introduce the planning languages. The following section will describe planners able to solve problems.

■ 2.3.1 Classical planning

Classical planning is planning for a restricted state-transition system that is also commonly referred to as *STRIPS planning*. *Stanford Research Insitute Problem Solver (STRIPS)* is an early planner for such a system, and the same name was later used to refer to the formal language based on the inputs to this planner.

The representation is based on predicate logic and it can be divided into a *planning domain*, a *planning problem* and a solution for a problem, i.e. a *plan* [23].

Domain A planning domain is a state-transition system consisting of

- a finite set of *states*,
- a finite set of *actions* and
- a state-transition function.

A state is a finite set of ground literals, i.e. logical atoms that are either true or false. If a literal is in a state, we say that it holds in this state. If it is not in a state, it does not hold. Further, a literal that is not explicitly specified in a state is assumed not to hold in that state.

An action is a ground instance of a planning operator represented as a triple composed of the operator's *name*, *preconditions* and *effects*. A name is defined as a symbol with variables appearing in the operator, e.g. $n(x_1, \dots, x_n)$, where n is a unique symbol and x_1, \dots, x_n are variables. Preconditions and effects are a set of literals. We further say that an action is *applicable* in a state if positive literals of action's preconditions hold in this state and negative does not. In other words, an action's preconditions are a subset of the state.

A state transition function is changing the truth values of literals in the state. If an action is applicable in a state, then negative literals of action's effects are removed from the state, and positive literals of action's effects are added to the state. The application of action's effects to the state can also be imagined as changing literals contained in action's effects from positive to negative and vice versa in the state.

Problem A planning problem is a triple composed of

- a planning domain,
- *initial state*, i.e. a state of the state space,

- a set of ground literals that have to hold in a state in order to be a *goal state*.

The ground literals are evaluated literals by a variable defined in a problem.

Plan A plan is a solution to a problem. It is a sequence of applicable actions from the initial state to the goal state. A plan with a sequence of actions $\pi_1 = \langle a_1, \dots, a_k \rangle$ have a length of its number of actions k (or also called plan steps). A *concatenation* of two plans $\pi_1 = \langle a_1, \dots, a_k \rangle$ and $\pi_2 = \langle a'_1, \dots, a'_j \rangle$ is a plan $\pi = \pi_1 \cdot \pi_2 = \langle a_1, \dots, a_k, a'_1, \dots, a'_j \rangle$. The plan is a *minimal* solution for a problem if no other solution contains a fewer number of actions.

The set of all *successors* of a state is a set of state-transition functions such that its actions are applicable in this state. The transitivity can be applied to the set of all successors resulting in a transitive closure forming a set of states *reachable* from the state. A plan has a solution if a goal state is in the set of states reachable from the initial state.

An *ordering constraint* is a constraint specifying that an action has to come before another action. If there is no ordering constraint, actions can run concurrently.

A *causal link* represents a relationship between two actions and shows the reason why an action was added. A causal link links a precondition of an action to an effect of another previous action. Between these two actions, there must be an ordering constraint. There can be a *threat* on a causal link. It can happen if there is a causal link between two actions sharing a positive literal p and another action ordered between them has in its effects a negative literal $\neg p$. A possible solution to this threat is a change of orderings of actions.

■ 2.3.2 Planning Domain Definition Language

Planning Domain Definition Language (PDDL) is an action language for specifying state transition systems and modelling a world. It was developed by Drew McDermott for the first International planning competition in 1998 [24].

The first version of this language starts at 1.2. The language was later extended to PDDL 2.1 [25] in 2003, PDDL 2.2 [26] in 2004 and PDDL3 [27] in 2005.

■ 2.3.2.1 PDDL 1.2

PDDL 1.2 [24] is inspired by *STRIPS* and several other formalisms. The intent of PDDL was to express physics. The language was designed to separate a domain description and problem description. Moreover, one such definition is allowed per file. The planning problem is then created by a pairing of a problem description with a corresponding domain description. The huge advantage of the separation is that the domain description can be pair to several problem descriptions.

2.3 PLANNING

Domain The definition of a domain starts with a keyword `define` followed by (`domain <name>`) specifying a name of this domain. The name is important for later pairing with a problem description.

Then a listings of *requirements* follows. It represents the fact that it was not expected that all planners would handle all features of the language. Therefore, there are subsets of features, which should be declared in each domain under requirements. These requirements allow a planner to tell if it is able to handle the domain. If the requirements are missing, the default requirement is `:strips` specifying the basic functionality as in classical representation.

The domain definition further contains *predicates* followed by definitions of *actions*. A predicate is represented as an atomic formula with variables denoted by a question mark. The list of all predicates in the domain is defined under the keyword `predicates`. Predicates are either true or false, and if a predicate is not specified, it is assumed to be false. These predicates represent the state, and all their possible combinations form the state space.

The definition of action is similar to that in a classical representation. After the keyword `action`, there is a name followed by three keywords `parameters`, `precondition` and `effect`. Parameters are all variables appearing in the action. Preconditions and effects are predicates using these variables. Again as in the classical representation, preconditions must be satisfied in order to apply the action's effects. The effects then change the values of predicates representing the current state.

The `:strips` requirement or a missing requirement usually support only the aforementioned features supporting mainly the basic addition or deletion of action's effects. The language allows several other additional features such as *constants*, *typing*, *extending* or *axioms*. However, these features are usually not supported by a majority of planners.

The keyword `extends` followed by a domain name allows inheriting requirements, types, constants, actions, axioms and timeless predicates from that domain.

The keyword `types` defined by a requirement *typing* allows to use objects of different types. Any object is by default a built-in type *object*. There is one more built-in type called *number*. It is possible to define more types and then create hierarchies and use the typed variables in the definition of predicates or action's parameters. Planners supporting this notion can then substitute only a subset of variables that might result in savings of time and memory. Moreover, it can operate with predicates taking types higher in the hierarchy. Actions can also take the top-hierarchy type and represents the transition for a more general group of types. But then predicates for a specific type cannot be used. It is also possible to use the keyword `either` followed by types instead of a single type. This feature is useful as a union of disjunctive hierarchical groups. Although this keyword seems to be a part of the requirement *typing*, not all planners supporting *typing* also support *either*.

The keyword `constants` allows symbols that will have the same meaning in all problems for this domain. The keyword `timeless` is followed by a list of literals that

are taken to be always true and cannot be changed by any action in the domain.

By default, an action's preconditions and effects can only be a literal or a list of literals connected by a logical operator **and**. If a requirements *:disjunctive-preconditions* is used, logical connections **or** and **imply** are also allowed in preconditions and logical operator **not** can be applied on the conjunction as well. Requirement *:universal-preconditions* allows the use of **for all** providing that the condition is true across all objects of a type or all objects. Requirement *:existential-preconditions* allows the use of **exists** providing that the condition is true if at least one object of a type exists. Both *:universal-preconditions* and *:existential-preconditions* requirements can be also expressed only as the requirement *:quantified-preconditions*.

If an action is applied, effects define which values should be set to true or false. Whereas in *STRIPS* representation, predicates are deleted or added. Requirement *:conditional-effects* allows the use of **for all** and *:conditional-effects* the use of **when**. It represents the secondary precondition meaning that if this precondition is true before the action, then the effect occurs. If this condition is not true, the action can be still applied, but the effect will not happen. If requirement *:action-expansions* is used, a keyword **expansion** is used instead of **effects**. This expansion is intended for performing abstract actions used by hierarchical planners. For example, an action *move* could be defined to describe a movement by a train, a car or a plane. An action expanded into several subactions intended to be in serial has to define the ordering among them. This is not required for parallel subactions.

■ 2.3.2.2 PDDL 2.1

PDDL 2.1 [25] was the official language for the 3rd IPC in 2002². This version introduces numeric fluents, durative actions and metrics.

Numeric fluents allow modelling also non-binary resources such as fuel, capacity, distance, time and so on. They are defined via functions in the domain file and initialized in the initial state in a problem file. Their values can be then changed using numerical operations on them in effects. They can also be accessed in conditions and their value compared to another value or a number.

Durative actions are actions similar to that in PDDL 1.2, but several changes were made to be able to represent a more realistic model of the world. These actions now have a duration that is represented by a numerical variable with the usage of numerical fluents. Thanks to the fact that time can be represented, durative actions can distinguish three time instants defined by keywords **at start**, **over all** and **at end** to be able to differentiate between conditions and effects applicable at the beginning of the action, during the duration of the entire action and at its end.

Metrics can be specified to improve the quality of the plan. In the metrics, there can be numerical variables that are evaluated at the end of the plan or a variable *total-time* denoting makespan, i.e. the length of the plan.

²<https://ipc02.icaps-conference.org/>

2.3 PLANNING

■ 2.3.2.3 PDDL 2.2

PDDL 2.2 [26] was the official language of the 4th IPC in 2004³. It introduces derived predicates and time-initial literals.

Timed initial literals enable that the literal becomes true or false at a specific time. In the previous version, it was supposed that predicates are either true or false at the start. This more realistic way of modelling allows representing that some resources become available from a certain time, and thus, they can be manipulated only afterwards.

Derived predicates can then model dependency between facts. For example, it allows evaluating if a vehicle is available based on the facts that it is charged and its loading space is empty.

■ 2.3.2.4 PDDL 3.0

This version was the official language of the 5th IPC in 2006⁴. PDDL 3.0 [27] introduces constraints and preferences.

Preferences are a form of soft constraints allowing to define soft goals or preconditions. Soft goals represent desirable but not so necessary goals. If such a goal is not satisfied, the solution is still acceptable. There is a cost for not meeting the goal, which can be incorporated into the plan metric and then influence the resulting quality of the plan.

Constraints represent hard constraints such as a strong goal. Such a goal has to be satisfied in order to consider the plan valid. It can also reduce the number of states needed to explore by describing additional logic. Constraints are useful for problems where vehicles are required to visit each location at most once. It is also possible to specify a time window for a predicate to be true.

■ 2.3.2.5 PDDL 3.1

PDDL 3.1 was the official language of deterministic track in the 6th IPC in 2008⁵. It is the latest version of the language, and it has been used up to IPC 2018 so far⁶.

PDDL 3.1 introduces object fluents which are state variables mapped to a finite domain. Object fluents also supports undefined value. For example, this value can be used when a durative action is moving an object o from the first location to the second location. Then, $\text{location}(o)$ can be set to undefined at the start and to the second location at the end. Other actions which require the object o to be at the first location or at the second location during this movement action cannot be used. Further, object

³<https://ipc04.icaps-conference.org/deterministic/>

⁴<https://ipc06.icaps-conference.org/deterministic/>

⁵<https://ipc08.icaps-conference.org/deterministic/PddlExtension.html>

⁶<https://ipc2018-classical.bitbucket.io/>

fluents are initially undefined and thus should be initialized analogously to numerical fluents.

This extension of the language allows the use of so-called multi-valued state variables, and therefore, this version can also represent SAS+ formalism. While this formalism has become to be popular, the usage of functional fluents is very limited.

■ 2.3.2.6 Discussion of suitability for modelling PDP-TW

At least the PDDL 2.1 is necessary for modelling PDP-TW because it introduces durative actions and thus allows concurrency. Both these aspects are necessary for describing schedules addressing PDP-TW.

PDDL 2.2 allows using timed initial literals, which can be suitable for modelling failures. However, failures are processed before invoking the planner so that they are not a direct input for PDDL but for our system first. Our system can handle time when the failure occurs, and thus, it can be classified as capable of limited usage of timed initial literals.

PDDL 3.0 further extends the previous versions by preferences whose violation costs can be reflected in a plan metric. These preferences could be used to describe the problem more appropriately with returning to the dock as a soft goal. Vehicles are then advised to go there but not required because the fulfilment of this task is not so essential for transporting goods. These preferences also enable to specify in which order cargos should be transported. While both these aspects model the environment more realistically, it does not have to be suitable for all specific PDP-TW. It depends on concrete usage and the environment. In the industrial environment solved in this thesis, both these features could be considered as rather not suitable. However, PDDL 3.0 allows modelling of time windows which are necessary for describing PDP-TW.

PDDL 3.1 adds object fluents that are not necessary for modelling PDP-TW.

To conclude, PDDL 3.0 is most suitable for modelling PDP-TW. However, as stated earlier, time windows bring a great complexity when failures are handled. Because this complexity goes beyond this thesis, time windows are not considered. Therefore, after this relaxation of the problem, PDDL 2.1 is sufficient for modelling PDP-TW. Moreover, significantly more open-source planners are available for use in PDDL 2.1 than PDDL 3.0 and above.

■ 2.4 Planners

Several planners participated in International Planning Competitions (IPC), which places interesting task planning challenges. Two planners attracted our interest and are described in more detail in the following sections.

■ 2.4.1 Temporal Fast Downward (TFD)

Temporal Fast Downward (TFD) [28] was published in 2009 and represents a group of planners focusing on temporal planning. This type of planning allows plans with concurrency thanks to the temporal ordering between actions. In contrast, classical planning takes into account only causal links. Previous planners to that time could find only sequential plans rescheduled in post-processing. TFD supports concurrent durative action and numeric fluents, and thanks to that, it can represent more real-world applications.

This planner translates PDDL formulations into SAS+ formalism, which can handle multi-valued state variables and logical dependencies. SAS+ is composed of state variables, initial state, goal state, axioms and durative actions. State variables are divided into logical and numerical variables. Numerical variables allow numerical operations such as addition, subtraction, multiplication, division and comparison operations such as less, greater, equal or combinations of two of them. These operations, together with classical operations on logical variables, are described by axioms. Durative actions can have start, persistent and end conditions; start and end effects; and a duration represented by a numerical variable. The planner also supports conditional effects. As a result, the authors claim that this temporal numeric SAS+ formalism can handle the whole PDDL 2.1 level 3 except duration inequalities and metrics.

The planner performs a heuristic search in the space of time-stamped states. These states contain a time stamp, a state with grounded variables, persistent and end conditions that have to hold until the time stamp with a delta of time, and effects with the remaining time after which they will apply if their persistent and end conditions hold. If there are no more conditions and effects in the state, it is called a progressive state. The search starts at time-stamped state with time 0, no conditions and effects, and repeats the progression until no further improvements appear. The planner performs an A* search in which durative actions are inserted to a certain time, and the time is incremented. There are small time increments to the time to ensure that if an action access a variable, any other action cannot update it. Duplicate actions differing only in this small time increment are eliminated. The search minimizes the function of the sum of the time and a heuristics estimating the cost to the goal.

The context-enhanced additive heuristic is extended to cope with temporal actions and numeric fluents. Because this heuristic was designed for non-temporal actions, it was necessary to create instant actions from the durative actions with the cost of its duration, start actions representing the durative action after applying start effects with the cost of the duration of the operator and waiting actions with the cost of delta of time. For the representation of numerical variables, it is sufficient to estimate the cost of changing the values and hence, actions that move variables closer to the desired value are identified. The objective of the search is to find plans with a low parallel makespan, but the concurrency is not taken into account. However, durative actions are summed into sequential makespans which usually correspond to parallel makespans.

The planner provides low-makespan plans of high quality, and it outperformed state-

of-the-art temporal planning systems in the article. Moreover, it can solve problems requiring plans with overlapping actions.

■ 2.4.2 OPTIC

OPTIC [29] was published in 2012 and belongs to the group of planners dealing with preference-based partial satisfaction planning. This planner does not aim to focus on minimizing makespans typical for temporal planning. Instead, the main interest is in preferences influencing the cost to be minimized.

The planner supports discrete penalty costs as defined in PDDL3, and it can also handle time-dependent costs from PDDL+. Thus, it can cope with a discrete model of the world provided by PDDL3, where deadlines appear precisely at a particular time. The cost function is then discretized into a set of deadlines. However, this approach may improperly represent the cost, and also deadlines can be missed. Therefore, the planner also supports the time-dependent costs increasing gradually over time, representing a continuous model of the world as in [30]. The continuous cost functions are modelled using PDDL+ and a variable to track the time. This variable is updated by a process that has no conditions. The effect of the process is only to increase the variable by one per time unit.

A mixed integer program (MIP) is used to find a preference-cost-optimal schedule. PDDL3 allows defining temporal preferences such as soft deadlines, goals or preconditions and can describe how these preferences were violated. The metric is then based on how these violations affect the total plan cost. The actions in the plan have to be scheduled according to the costs influencing the violations of preferences. MIP then optimizes the cost of preferences, assignment of timestamps to steps, and other metrics subject to the ordering constraints to find a minimal cost. To guide the search, a relaxed plan heuristic is used whose principle is similar to iterative deepening A* (IDA*). Actions are then scheduled at the earliest possible time to obtain the lowest cost.

This approach showed good scalability. In contrast, it suffers from difficulties to find a solution to larger problems within 30 minutes limited to 4 GB memory. However, if a solution is found, the first plan has a good makespan.

■ 2.5 Plan modification

There are several reasons why one might want to modify a plan rather than generate a new one from scratch. One can feel the intuition that doing so is more efficient, faster or easier. Several authors supported this idea [31, 32, 33, 34]. However, from the analytical point of view, the worst-case scenario of these intuitively better methods might be even harder than generating a plan from scratch [35]. As stated in the paper, it depends on the context.

2.5 PLAN MODIFICATION

Kambhampati distinguished three situations when one finds the modification useful [31]. The first situation is *plan reuse*. In this situation, the existing plan is used to solve new planning problems. First of all, a similar plan has to be mapped to the new situation. This typically involve searching for a plan stored in a *plan library*, which has a very similar *init state* and *goal state*. Nevertheless, this plan cannot usually be directly used in the new planning problem because there is a gap between the old plan and the new one. There might be several inconsistencies necessary to repair. They are of three types - unused effects because some goal is unnecessary in the new situation, missing effects supporting new goals and possibly threads to links. Therefore, some links are removed, some added, and some redirected in [31] so that unnecessary parts of the plan are removed, new tasks are added, and the plan is validated to be correct. The resulting plan is then reduced with the usage of the planner. According to Kambhampati, it is faster to start with a partially reduced plan than generate it from scratch. Although the proposed method introduces 20-98% savings in the blocks-world domain, the matching part is not discussed, and a similar plan to reuse is given. [35] argue that this matching part can have a significant impact on savings. The second situation is referred to as *replanning*, in which the current plan is modified in response to failures due to time execution; and the last one is called *incremental planning* when the current plan is updated in response to evolving specifications.

The idea of storing various plans is also typical for *refinement planning* [32, 36]. The set of all possible potential solutions, called a *candidate set*, is reduced by constraints until a solution is found. Different refinement strategies can be applied to obtain a partial plan. The refinement planning can also be useful for extending the plan with actions that will reach the goal. [32] extends the refinement planning with *unrefinements* in which actions are removed from the tree, and a heuristic is computed to estimate from which node a solution is obtained. The tree represents the dependency of action, and hence, it is possible to backtrack which actions have unused effects and thus can be removed, and the plan can be shrunk.

The derivational analogy in case-based planning is presented in [33]. The trace of previously made decisions is kept and applied to create candidate sets. Then the refinements techniques are applied to gain the solution. The proposed method results in savings in search space compared to planning from scratch. It is also more efficient because the first found candidate is used without any requirements on preserving as many plan steps as possible.

It is important to differentiate various purposes of plan modification. Plan modification done in the plan reuse context can be seen as a more efficient way how to obtain new plans. The reason for that is usually in the sense that creating a plan is an operation consuming a huge amount of time and memory. Therefore, to lessen this consumption, some parts of already created plans can be reused to create new ones. However, there is a number of caveats to bear in mind [35]. On the one hand, a huge amount of time can be consumed on searching the best candidate from the plan library which would fit the new planning problem so that the modifications done to reuse this plan are quite fast. On the other hand, the less time is spent searching for a similar

plan, the more time then the modification part lasts. Nevertheless, plan modification can be faster than a generation from scratch in some cases, such as the usage of effective matching of plans. Nowadays, considering modern planners and hardware, the generation of the plan is not that painful as it used to be earlier.

Plan modification can also be done in the replanning context. In this context, the emphasis is put on retaining as many same steps in the new plan as in the original one, which is usually specified by the user. The new plan is usually required due to some occurred failures or the changed specification of the goal.

Komenda et al. [22] defined a multi-agent plan repair. The execution of the original plan fails after some number of steps and gets into the *failed state*. The aim is to find a solution to the new planning problem starting in that *failed state*. The minimal repair of the plan is such that mutual differences between the original plan and the new one are minimized. If the new planning problem is just constructed from the *failed state* and then given to a planner to solve it, then this principle is called *replanning*.

In [22], there are presented three plan repair algorithms. A number of actions were executed from the original plan when the failure occurred, and the situation has changed to the failed state s_f . In the first algorithm, a planner is invoked to get from the failed state s_f into some state of the original plan so that the remainder of the actions of the original plan can continue executing. The longer this remainder is, the more preserving the resulting plan is. In the second algorithm, the remainder of actions from the original plan is reused starting at the failed state s_f , and this results in some state s_{lazy} . From this state s_{lazy} , a planner is invoked to get into the goal state. Both previous algorithms are combined in the third. Firstly, the parametrized number of actions f are reused from the original plan starting at the failed state s_f and resulting in some state s_{pre} . Secondly, the parametrized number of actions g are reused from the original plan ending in the goal state and starting in some state s_{suf} of the original plan. The state s_{pre} is obtained by applying the actions, while the state s_{suf} is gained by backtracking from the goal state. A planner is then invoked to provide a plan starting at the state s_{pre} and resulting in the state s_{suf} . If the plan is not found, different combinations of f and g are tried. Repair algorithms were compared to replanning from scratch in terms of communication overhead among agents in distributed planning and perform more efficiently.

Another approach related to plan modification is plan merging. In [37], partial order plans with durative actions are merged together. The input plans are plans for tasks obtained separately. The aim is then to merge these plans together into a single plan so that actions can be interleaved, resulting in a shorter plan. There are two key features in this approach. Firstly, not all actions from the input plans have to be used. Secondly, some new actions can be added to satisfy the causality of the resulting plan. In [38], the goals are assigned to agents who provide plans individually. These plans are then merged by simple concatenation. If goals are not achieved, a centralized planner is called. If the merged plan is not valid, then it is reused as an input to the RRT-based planner. The resulting valid plan is then parallelized. [39] deals with the problem of conflicts in terms of threatening causal links when trying to merge plans of individual

2.5 PLAN MODIFICATION

agents. The plan to be included in the joint plan can be delayed, but then this delay is included in a cost penalty whose higher number encourages an agent to deviate from its plan. Agents iteratively revise their plans with respect to the joint plan until convergence has reached.

The plan can also be modified in the sense of contingency. In [40], the *contingency planning* is presented for UAV domain missions. There are two resources in this domain limiting the number and type of tasks achievable by a vehicle - the finite amount of a battery and a data storage space. In these missions, the aim is to obtain as much data as possible with respect to the battery and also to be able to recover them, e.g. by going to the surface or transmitting them in order to increase the storage space. To be able to do so, the vehicle is allowed to change its plan during the execution by switching among the generated plans according to the available resources. The proposed method is based on the Markov decision process with penalties and rewards.

A criterion on how to measure preserving as many plan steps of the original plan as possible is defined in [34]. This measure is called *plan stability*, and it can be computed as the difference between two plans. This difference is obtained as a number of actions occurring in the new plan but not in the original plan, plus the number of actions appearing in the original plan but not in the new plan. The importance of both sides is supported by the following example. The series of drive actions in the original plan can be replaced by a flying action in the new plan. The number of actions occurring in the new plan but not in the original plan is 1. However, the plan is completely different, and the measure of it is shown by the second side of the criterion - the number of actions appearing in the original plan but not in the new plan. In this paper, the criterion is used as a heuristic for plan modification to estimate the cost of adding or removing an action in a partial plan. The performed experiments showed that the plan adaptation approach is usually faster and has a significantly better *plan stability* than generating a new plan from scratch. It is also interesting to observe that for small problems, there is only one way how to solve them, and the solution is found by both approaches.



Chapter 3

Problem definition

The purpose of this chapter is to define the general problem addressed in the thesis. This chapter starts with a general introduction to the background of the solved problem, and then it presents the problem tackled in this thesis. Finally, solution approaches are suggested.

■ 3.1 Problem background

The problem for which this thesis resolve failures is a pickup-and-delivery problem, more specifically the *one-to-many-to-one PDP* in industrial environments such as factories with production and logistics. In this problem, each robot typically delivers goods from a depot (e.g. a storage centre) to a customer (e.g. a conveyor belt) and from the customer back to the depot. The latter part can represent for example a collection of empty boxes back to the storage centre. As in the classical VRP, the question is which robot should serve which delivery and in which order. The solution to this problem is given by an operator or a state of the art planner in the form of a plan.

The additional constraints on our PDP are as follows. The fleet of robots can be heterogeneous, but each robot has its capacity of one box, i.e. it is either full or empty. All robots start in a dock station and, in the end, return back. Robots are reused for different deliveries such as in *VRP with multiple use of vehicles*, but their battery is assumed to be sufficient to serve all requests. There are no other constraints such as time windows because it is out of the scope of this thesis.

The given plan is represented by a set of durative actions as defined in PDDL 2.1. These actions represents a pickup, a delivery and driving. During the execution of such a plan, failures can occur so that execution of (some) tasks cannot proceed. Failure addressed in this thesis are of two types - failure of a robot and failure due to a blocked path. The aim of this thesis is to resolve these failures so that the execution can continue.

■ 3.2 Problem statement

Suppose that the plan specified by an operator is given. While this plan is executing, a failure occurs at some time instant, resulting in a change of the state. Due to this failure, (some) tasks of the plan cannot be fulfilled. The task is to find a plan that achieves the goals of the operator's plan while preserving most of its plan steps.

■ 3.3 Solution approaches

We suggested two methods to solve this problem - plan repair and replanning from scratch.

Plan repair. This approach is based on modifying the operator's plan. The idea is to preserve the plan steps unaffected by the failure and provide a new plan only for the influenced steps. To obtain this new plan, the unaffected goals of the operator's plan are included in the initial state of the corresponding problem, and the goal state contains mainly the goals not satisfied due to failures. These two plans are then joined into the resulting plan. The detailed description of this method is in chapter 4.1.

Replanning from scratch. This is an approach that provides a whole new plan. The initial state is equal to the state in which a failure occurred, and goals are equal to those of the operator's plan. Although a state of the art planner is used in both methods, this approach does not respect the preservation of the operator's plan.



Chapter 4

Solution

Two approaches to solve the problem stated in chapter 3 are described in this chapter. The aim is to present the proposed method of repairing a plan in which a failure has occurred while preserving as much as possible of the original plan specified by an operator. The method of repairing a plan by replanning from scratch when a failure appears is demonstrated for comparison reasons. The second part of this chapter concerns the integration of these methods into the REX system.

■ 4.1 Minimal plan repair

This section focuses mainly on the description of the plan repair method, but the important differences in the replanning from scratch approach are introduced at the end of this section.

First of all, a model of the world is presented. This thesis focuses on industrial environments in which AGVs are transporting goods. Hence, both methods are adapted to such a world which is described by the PDDL language. A planning domain modelling the world is then necessary to obtain a new plan. The proposed methods are discussed in more detail in the following sections.

■ 4.1.1 Model of the world

Modelling of a world corresponding to an industrial environment such as a factory with production and logistics was done in the PDDL language. In the next sections, the decisions made to model the domain representing the world are described. The description of the corresponding initial problem representing the task of the pickup-and-delivery problem follows.

■ 4.1.1.1 Modelling of PDDL domain

In this section, a PDDL domain used in the thesis is described, and its modelling is explained. The PDDL domain is modelled in PDDL 2.1, which allows the use of *durative-actions*. These durative actions enable the possibility of keeping track of the

4.1 MINIMAL PLAN REPAIR

time required in real-time applications. The entire domain is attached in Appendix A.1.

The domain uses types whose hierarchy is depicted in Figure 5. All objects are of type *object* by default. Then there are two children extending the type *object* - *waypoint* and *locatable*. A *waypoint* represents nodes in a graph, i.e. possible locations, in which robots can perform some actions. The type *locatable* is further divided into *agv* used for robots and *cargo* used for transported goods. Both these objects change their position in time, and hence, the type *locatable* can be referenced to use this property instead.

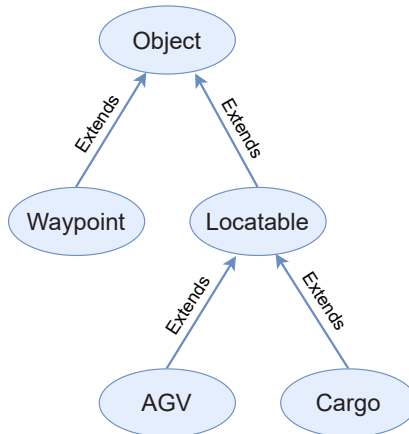


Figure 5: Hierarchy of objects used in the domain

Predicates used to model properties of the aforementioned objects are described in Listing 1. It is necessary to know where the robots and goods are. This is represented by the predicate *at* - the *locatable* object (either an *agv* or a *cargo*) is *at* a *waypoint*. If the cargo is loaded into a robot, it is no longer at the waypoint but in that robot. This is represented by predicate *in* - a *cargo* is *in* an *agv*. Further, there has to be a *path* from a *waypoint* to another *waypoint*, to be able to drive there by a robot. A failure due to a blocked path can be then easily obtained by the negation of this predicate. The rest of the predicates are properties of each robot. The capacity of the robot is modelled either as *full* or *empty*. Notice that these predicates represent the same property, i.e. they are the negation of each other. The redundancy of these two predicates is necessary to be able to use planners that do not support *negative-preconditions*. In order to be able to simulate a failure of a robot, the information whether the robot is *alive* is needed.

Listing 1: Predicates used in the PDDL domain

```
1  (:predicates
2    (at ?obj - locatable ?wp - waypoint)
3    (in ?cargo - cargo ?agv - agv)
4    (path ?from - waypoint ?to - waypoint)
5    (empty ?agv - agv)
6    (full ?agv - agv)
7    (alive ?agv - agv)
8  )
```

No more predicates are needed because of several assumptions about the environment and robots. A path is assumed to be a single bidirectional path, i.e. there always exist a *path* from a first *waypoint* to a second *waypoint* and vice versa. If a path is blocked because of an unavoidable obstacle, then it is blocked in both directions. AGVs are assumed to be smart enough to avoid each other and small obstacles on a path which is wide enough for such manoeuvres. Thanks to this assumption, AGVs can go around those AGVs which became not alive on a path. Additionally, more AGVs can drive on the same path. If an AGV is not alive at some waypoint, it is assumed that there is still enough room for another AGV to load or unload. Therefore, it is not necessary to model how to block a waypoint. These assumptions simplify the modelling of a world and problem-solving so that the solution can be provided in a reasonable time.

A function is used to represent a different duration of driving along paths. A function (`travel_time ?wp1 - waypoint ?wp2 - waypoint`) allows to define a travel time between two waypoints. This value is initialized for every two connected waypoints in a problem file.

The description of actions changing the state of the world follows. There are only three actions necessary to represent the pickup-and-delivery problem - the action *load* representing the pickup in a storage center, the action *drive* to be able to get to another location and the action *unload* for delivering goods. These actions are described in more detail in the following paragraphs.

The action *load* means that an *agv* loads a *cargo* at a *waypoint*, see Listing 2. For this action to be applicable, all conditions have to be met. The *agv* has to be *alive* and *at* the *waypoint* for the entire time that this action lasts, i.e. two minutes. This duration is inspired by Driverlog domain⁷, the value is not important for the proposed method. At the start of the action, the *agv* has to be *empty* and the *cargo* has to be *at* the *waypoint*. The effect of performing this action is that the *cargo* is not *at* that *waypoint*, but *in* the *agv* at the end. This means that the *agv* is not *empty* anymore, but becomes *full*.

Listing 2: Action *load* in the PDDL domain

```

1  (:durative-action load
2    :parameters (?agv - agv ?cargo - cargo ?wp - waypoint)
3    :duration (= ?duration 2)
4    :condition (and
5      (over all (alive ?agv))
6      (over all (at ?agv ?wp))
7      (at start (at ?cargo ?wp))
8      (at start (empty ?agv))
9    )
10   :effect (and
11     (at start (not (at ?cargo ?wp)))
12     (at end (in ?cargo ?agv))
13     (at start (not (empty ?agv)))
14     (at end (full ?agv))
15   )

```

⁷Available at https://helios.hud.ac.uk/scommv/IPC-14/domains_temporal.html

4.1 MINIMAL PLAN REPAIR

16)

The action *unload* is very similar, see Listing 3. An *agv* unloads a *cargo* at a *waypoint* and it again lasts two minutes. During this action, the *agv* has to be *alive* and *at* the *waypoint*. At start of this action, the *agv* has to be *full* because the *cargo* has to be *in* this *agv*. After applying this action, the *cargo* is no more *in* the *agv*, but appears *at* the *waypoint* at the end. This fact corresponds to the *empty agv*, which is no longer *full*.

Listing 3: Action *unload* in the PDDL domain

```
1  (:durative-action unload
2  :parameters (?agv - agv ?cargo - cargo ?wp - waypoint)
3  :duration (= ?duration 2)
4  :condition (and
5  (over all (alive ?agv))
6  (over all (at ?agv ?wp))
7  (at start (in ?cargo ?agv))
8  (at start (full ?agv))
9  )
10 :effect (and
11 (at start (not (in ?cargo ?agv)))
12 (at end (at ?cargo ?wp))
13 (at start (not (full ?agv)))
14 (at end (empty ?agv))
15 )
16 )
```

The last described action allows an *agv* to *drive* from a *waypoint* to another *waypoint*, see Listing 4. The duration of this action is dependent on the *travel time* between these waypoints. To perform this action, the *agv* has to be *alive* over the entire time and start *at* the *waypoint* where the path begins. The *path* has to exist between these waypoints during this action. The effect gets the *agv* to the *waypoint* where the path ends, which means that the *agv* is no longer at the *waypoint* where the path begins.

Listing 4: Action *drive* in the PDDL domain

```
1  (:durative-action drive
2  :parameters (?agv - agv ?from - waypoint ?to - waypoint)
3  :duration (= ?duration (travel_time ?from ?to))
4  :condition (and
5  (over all (alive ?agv))
6  (at start (at ?agv ?from))
7  (over all (path ?from ?to))
8  )
9  :effect (and
10 (at start (not (at ?agv ?from)))
11 (at end (at ?agv ?to))
12 )
13 )
```

■ 4.1.2 Proposed methods

Two possible approaches solving failures in the execution of a plan are described in this section. The first approach called *plan repair* respects as many steps of the operator's plan as possible, while the second approach called *replanning from scratch* does not, but it uses the current state of the environment and creates a completely new plan.

For both methods, the operator's plan has to be processed first, and the corresponding planning problem can be recreated. The operator's plan is analyzed after the failure in the execution has occurred. These failures have the result that not all of the original goals of the restored original problem can be achieved if the execution continues. Therefore, it is required to provide a new plan for goals that cannot be satisfied by the operator's plan. A new planning problem has to be created to obtain a plan for these goals. This plan is then integrated back into the execution. Two solutions that provide a plan achieving all original goals even though failures occur are presented in this section.

Restoration of problem from operator's plan. The operator's plan can be analyzed so that the corresponding problem can be recreated. The preconditions and effects of actions in the operator's plan can be obtained with the knowledge of the domain. The initial state and goals of the problem can be then restored from these preconditions and effects. Causal links between actions can be recreated to show the reason why an action was added to satisfy the goal.

Analysis of a failure in a plan. Suppose a failure occurs so that some actions cannot be executed anymore and some actions are not influenced by this failure at all and hence, can continue to execute. Preconditions of influenced actions are not satisfied due to the failure, and thus these actions cannot be applied, resulting in some goals that cannot be reached. These goals are called disturbed goals. Actions that are not influenced by a failure have satisfied preconditions, are applicable and lead to goals that can be reached. These reachable goals are called valid goals.

■ 4.1.2.1 Plan repair

In this approach, reachable goals are supposed to be finished to preserve as many plan steps of the operator's plan as possible. Doing so means that this part of the operator's plan is reused in the repaired plan. The rest of the operator's plan is connected with disturbed goals and thus has to be modified. The aim is to found a new plan for these disturbed goals and join it with the original operator's plan. In the following sections, it is described how to obtain this repaired plan.

4.1 MINIMAL PLAN REPAIR

Creation of goals

To be able to provide a new plan for disturbed goals, it is critical to ensure that a plan can be found for these goals. Therefore, an analysis of these goals is discussed in this section first, followed by a solution on how to generate goals for the new problem to obtain a new plan.

The problem addressed in this thesis is a pickup-and-delivery problem and therefore, only two types of goals can appear in the disturbed goals. The first one is that an *agv* is *at a waypoint* and the second one is that a *cargo* is *at a waypoint*.

A plan cannot be found if an AGV that becomes not alive at some waypoint is requested to be at another waypoint. If a failure of an AGV occurs, then this AGV cannot perform any actions in the real world. This situation is represented in the corresponding model of the world by a literal saying that an *agv* is not *alive*. This literal is in the preconditions of all actions in the model of a world, and therefore, any such action cannot be performed with this AGV. This impossibility also means that this AGV cannot drive to another waypoint. Therefore, any plan cannot be provided to satisfy such a request.

The reason why any plan cannot be provided is that planners have to achieve a state in which all goal literals hold. If one of the goal literals cannot be reached, then a solution cannot be found. Although, the rest of the goal literals can be reachable. In our domain, this behaviour of planners means an *agv* that is not *alive* cannot be requested on any other waypoint than it already is. Otherwise, no solution could be found on how to satisfy a cargo at a waypoint. Even though such a literal is reachable and a plan on how to serve this cargo could be found, for example, any other *alive agv* existing in the domain could transport this cargo.

There are situations where a cargo cannot be served as well. There are two reasons for such a situation - either a failure of AGVs or a failure due to blocked paths. In the former case, there are no alive AGVs existing in the domain that could serve this cargo. In the latter case, there does not have to be any connection of paths to the place where this cargo is or where it is requested to be. All these situations result in no solution so that no plan can be provided. However, there is no other way to obtain a plan. Such failure can be solved only by human interaction in a factory. Therefore, such failure does not have to be analysed and solved in advance because it is a responsibility of a state of the art planner.

It is also not necessary to analyse in advance whether it is possible to require an alive AGV to be at a waypoint. If this literal is a goal, then the AGV is requested to be at the waypoint where a docking centre is. If there is no path leading to the docking centre, no plan can be provided. Although, the rest of the goals, such as serving a cargo, can be satisfied. It is possible to analyse the connectivity of paths in advance and then not to require such a request on a position, but this is a responsibility of a planner. Moreover, all AGVs have to be in the docking centre at the end to recharge.

It is important to mention that a request for a cargo to be at a waypoint is much stronger than for an AGV to be at a waypoint. If a cargo is not at a waypoint, the

production of the factory has to stop, so that this is a strong failure. Whereas if an AGV is not at a waypoint, where a docking centre is, this AGV will soon run out of battery, resulting in only a weak failure. This weak failure can, of course, become a strong failure if all AGVs run out of battery because no AGVs can serve cargos anymore, and the production of the factory has to stop anyway.

New goals for the new problem can be generated by refining the disturbed goals. From the analysis discussed in this section, it is necessary to remove all literals requesting an AGV that is not alive to be at a waypoint. Handling only this removal has two significant advantages. Firstly, the refinement is quite fast, and secondly, it has a high level of achievement. Consequently, it is possible to require a cargo at a waypoint even though there will be no solution.

The algorithm describing the refinement of disturbed goals is in Algorithm 1. It iterates over all disturbed goals. If there is a literal saying that an *agv* is *at a waypoint*, this AGV is checked. If it is not alive, the literal is discarded.

Algorithm 1 refine_goals($G_{disturbed}$)

Input: Disturbed goals $G_{disturbed}$

Output: Refined disturbed goals $G_{refined}$

```

1: for each literal  $l_g$  in disturbed goals  $G_{disturbed}$  do
2:   if  $l_g$  is (at ?agv ?wp) then
3:     if ?agv is not alive then
4:       continue;
5:     end if
6:   end if
7:    $G_{refined} \leftarrow l_g$ 
8: end for
9: return  $G_{refined}$ 

```

Creation of initial state

The initial state of the new PDDL problem is developed from the current state of the world in which the execution of some actions has stopped due to the occurrence of a failure. At this moment, the plan repair approach is launched to provide the initial state for the definition of the problem to provide a new plan for the inexecutable part of the operator's plan. To preserve as many plan steps of the operator's plan as possible, valid goals are assumed to be finished. This assumption has to be reflected in the initial state. Goal literals from valid goals have to hold, i.e. they have to be included in the initial state. Moreover, the initial state has to reflect the state of the world after these goals have been achieved so that no other conflicts in the state of the world appear. The current state from which the initial state is derived also contains the literals that caused the failure of the execution of some actions in the plan.

4.1 MINIMAL PLAN REPAIR

Algorithm 2 $\text{replay_plan}(\pi, s_{init}, L_{failures})$

Input: Operator's plan π , initial state of the operator's problem s_{init} , failure literals that induced plan repair $L_{failures}$

Output: Future state s_{future}

- 1: $s_{current} \leftarrow \text{get_current_state}(\pi, s_{init})$
 - 2: $s_{current} \leftarrow s_{current} \cup L_{failures}$
 - 3: $s_{future} \leftarrow \text{simulate_future}(\pi, s_{current})$
 - 4: **return** s_{future}
-

The analysis of the plan revealed that the initial state could be obtained by simulating the future execution of the plan. At the moment when the failure occurred, the current state is changed to reflect this situation. This information is represented by literals describing the failure in the current state. From this current state, the rest of the plan is simulated. Some actions are no longer applicable due to failure literals, and thus, their effects are not applied to the current state. This way, some further actions will not be applied because their preconditions depend on the effects of previous actions. Therefore, the part of the operator's plan leading to disturbed goals is not applied and does not change the current state. Whereas actions that are still applicable in the current state are applied, resulting in the state in which valid goals of the original operator's plan are achieved. This state is called the future state. By this approach, as many as possible plan steps of the operator's plan are reused.

This simulation of the execution is described as the replay of the plan in Algorithm 2. First of all, it is necessary to obtain the current state of the execution, i.e. the state in which plan repair is launched. The generation of the current step is needed because the provided plan contains all plan steps from the start of the execution. Previously executed steps are then crucial for setting the correct scheduled time of the new plan, as discussed later. Therefore, the plan carries the information of which actions were already executed. Further, failures that caused this plan repair are put into this current state to reflect that failures changed the current state of the world. Finally, the plan

Algorithm 3 $\text{get_current_state}(\pi, s_{init})$

Input: Operator's plan π , initial state of the operator's problem s_{init}

Output: Current state $s_{current}$

- 1: $s_{current} \leftarrow s_{init}$
 - 2: **for** each action a in plan π **do**
 - 3: **if** a is executed **then**
 - 4: **if** preconditions of a hold **then**
 - 5: apply effects of a to $s_{current}$
 - 6: **end if**
 - 7: **end if**
 - 8: **end for**
 - 9: **return** $s_{current}$
-

can be simulated until its final state. This final state is the initial state of the new problem.

The generation of the current state is provided in Algorithm 3. The current state starts as the initial state of the plan. It is further checked for each action from the plan whether it was already executed and, if so, whether the preconditions of this action hold. If this action is applicable in the current state, the effects of this action are applied so that they change the current state. In a similar manner, the future state is simulated in Algorithm 4. The difference is that the future state starts as the current state, and only actions that were not yet executed and are applicable are applied. It is important to notice that these actions lead only to valid goals because disturbed goals cannot be achieved due to failure literals present in the state. In this way, the future state reflects the state in which valid goals are already achieved, and moreover, it already contains the failure literals.

Algorithm 4 `simulate_future($\pi, s_{current}$)`

Input: Operator's plan π , current state in execution $s_{current}$

Output: Future state s_{future}

```

1:  $s_{future} \leftarrow s_{current}$ 
2: for each action  $a$  in plan  $\pi$  do
3:   if  $a$  is executed then
4:     continue;
5:   else if preconditions of  $a$  hold then
6:     apply effects of  $a$  to  $s_{future}$ 
7:   end if
8: end for
9: return  $s_{future}$ 

```

Plan for disturbed goals

A new plan for disturbed goals is gained by the state of the art planner. First of all, the PDDL problem file has to be created. The definition of the problem is composed of new goals provided by the refinement of disturbed goals, the new initial state generated by the replay of the executable remainder of the operator's plan, and objects that are equal to objects of the restored problem corresponding to the original operator's plan. A system call is then used to wrap a state of the art planner and call it to solve the new problem provided the domain file and the automatically generated problem file. The output of the planner is a plan for the disturbed goals.

Joining plans

The newly obtained plan for disturbed goals has to be joined with the original operator's plan that provides steps to achieve valid goals, and thus, this part of the operator's plan

4.1 MINIMAL PLAN REPAIR

is reused. The assumption that all valid goals are finished and the creation of the initial state restrict the possibilities where the newly obtained plan can be placed into the operator's plan. The initial state created by replaying the operator's plan constrains the conjunction because the part of the operator's plan that is reused by this process is assumed to be finished. Therefore the plan for the disturbed goals of the operator's plan can be put after this part, i.e. at its end.

Algorithm 5 $\text{join_plans}(\pi_{operator}, \pi_{new}, O_w)$

Input: Executable remainder of the operator's plan $\pi_{operator}$, plan for disturbed goals of the operator's plan π_{new} , objects from the problem corresponding to original operator's plan O_w

Output: Repaired plan $\pi_{repaired}$

```
1:  $\pi_{repaired} \leftarrow \pi_{operator}$ 
2:  $agvs \leftarrow$  all agvs objects from  $O_w$ 
3: for each  $agv$  in  $agvs$  do
4:    $\pi_{agv} \leftarrow$  select subplan of  $\pi_{new}$  regarding  $agv$ 
5:   if  $\pi_{agv}$  is empty then
6:     continue;
7:   end if
8:    $last\_action \leftarrow$  last action regarding  $agv$  in  $\pi_{repaired}$ 
9:    $\pi_{repaired} \leftarrow$  insert  $\pi_{agv}$  after  $last\_action$  of  $\pi_{repaired}$ 
10: end for
11: return  $\pi_{repaired}$ 
```

Durative actions are scheduled, and thus, it is necessary to handle the resulting plan for each AGV separately to maintain the correct scheduled time of each action. The rest of the executable operator's plan and the newly obtained plan are divided into subplans related to each AGV. The corresponding subplans are then concatenated. The time of each action in the newly obtained subplan of each AGV is updated so that it fits the previous action from the executable rest of the operator's subplan of the corresponding AGV. See Algorithm 5.

The benefit of such conjunction is that the rest of the executable operator's plan can continue to execute while the disturbed part is repaired and then connected to the end. After the repair and conjunction of plans, the resulting plan can continue to execute because all AGVs and cargos are at expected waypoints, and the state of the world is expected so that further actions can be performed.

Plan repairer

A failure in the execution of the operator's plan launches the plan repair method. First of all, the operator's PDDL plan file is parsed, and the corresponding problem file is restored. The problem file contains the information about the initial state, goals and objects of the operator's plan. Further, the operator's plan is analyzed so that the new

problem for the affected part of the operator’s plan can be automatically generated. A state of the art planner provides a plan that is parsed and joined with the operator’s plan so that the whole process can repeat.

The automatic generation of the problem for the affected part of the operator’s plan attempts to reuse most of the original operator’s plan. The operator’s plan is analyzed, and its execution is simulated until the end. The replay of the operator’s plan provides a future state in which goals unaffected by failures are achieved while goals influenced by failures are not. Moreover, this future state carries information about failures. Goals influenced by failures are disturbed goals for which a new plan has to be provided. A state of the art planner ensures the generation of the new plan provided a domain and problem files. The problem file is automatically generated from the future state, which is the initial state of the new problem, and refined goals developed from operator’s goals. Such an initial state of the problem assures that as many of the operator’s plan steps as possible are preserved. The goals have to be refined to guarantee that the resulting plan can be provided for cases in which a solution to the task exists in general.

Algorithm 6 Repair plan

Input: Current plan file $F_{current}$, domain file F_{domain} . Objects O_w , goals G and initial state s_{init} from the problem corresponding to original operator’s plan. Failure literals that induced this plan repair $L_{failures}$.

Output: Current plan file $F_{current}$

- 1: $\pi_{current} \leftarrow$ parse plan from $F_{current}$
 - 2: $s_{future} \leftarrow$ replay_plan($\pi_{current}, s_{init}, L_{failures}$)
 - 3: $G_{refined} \leftarrow$ refine_goals(G)
 - 4: $F_{problem} \leftarrow$ create PDDL problem file, in which init is s_{future} , goal is $G_{refined}$ and objects are O_w
 - 5: $F_{replanned} \leftarrow$ call a planner with domain file F_{domain} and problem file $F_{problem}$
 - 6: $\pi_{replanned} \leftarrow$ parse plan from $F_{replanned}$
 - 7: $\pi_{repaired} \leftarrow$ join_plans($\pi_{current}, \pi_{replanned}, O_w$)
 - 8: $F_{current} \leftarrow$ save $\pi_{repaired}$
 - 9: **return** $F_{current}$
-

Algorithm 6 realizes the plan repair method⁸. The current plan is parsed and replayed within the occurred failure so that the future state is obtained. The future state is the final state of the plan after all applicable actions are performed. A new problem is generated from refined goals and the future state as the init. A new replanned plan is obtained by solving this problem. Finally, the resulting plan is parsed, joined with the current plan and saved to the PDDL file. If another failure occurs, the process can repeat until the plan is finished.

The data flow diagram of the algorithm is depicted in Figure 6. The operator’s PDDL plan file is parsed in Plan Parser. The resulting current plan is analysed in Plan Analyzer based on the failures that occurred during the execution of the plan

⁸Check of empty goals, plans or files is omitted for better clarity.

Joining plans. The newly obtained plan for disturbed goals is joined with the operator's plan to ensure continuity. The newly obtained plan is again divided into subplans for each AGV. The corresponding subplans are then concatenated with the corresponding already executed subplans of the operator's plan. Algorithm 5 is the same also for this approach with the exception that the last action regarding an AGV in the operator's plan is the last *executed* action. Scheduled actions in the operator's plan are discarded, and new actions from the new plan are added.

Replanning method. The method is very similar to the plan repair algorithm 6. Failures during the execution of a plan induce the replanning from scratch method. The current operator's plan is parsed and replayed so that the current state of the execution, including failures, is obtained. The goals achieved in this state are valid goals, and the rest are the disturbed goals for which a new plan has to be provided. Operator's goals are refined to ensure the plan can be provided. A new planning problem is then created using the current state as the initial state and refined goals as the goal. A new plan is gained by a state of the art planner, which takes the domain file and the new planning problem file. The resulting plan is parsed and joined with the operator's plan so that the scheduled time of actions in the plan fits the last executed action. This plan is saved and used as the current plan so that the whole process can repeat.

■ 4.2 System

The system for the organization of robots is called REX, and its overview described in more detail is given in [1]. In this chapter, REX is briefly introduced, and the main concern is then focused on the part of the system referred to as Minimal plan repair. The responsibility of this system is to react to failures occurring during the execution and provide a repaired plan which will be very similar to a plan created by an operator.

■ 4.2.1 The REX system

The system is composed of three main modules depicted in Figure 7. Each robot will have its own localization unit so that it is possible to obtain its position at any time. If a robot is controlled autonomously, it will also have a control unit responsible for navigation and control, i.e. the execution. Each robot will communicate with the server stored in a cloud. The server represents the central control authority that connects all involved mobile platforms into a fleet.

The position information from the localization unit is necessary mainly for navigation from one place to another one. It is also useful for the consequent planning. The task of the control unit is to move to the goal position based on the initial position provided by the localization unit and to control the carried technology such as a lift. This way, it will perform the plan obtained from the server system while sending back the feedback from the execution. The server system consists of the planning and scheduling of tasks

4.2 SYSTEM

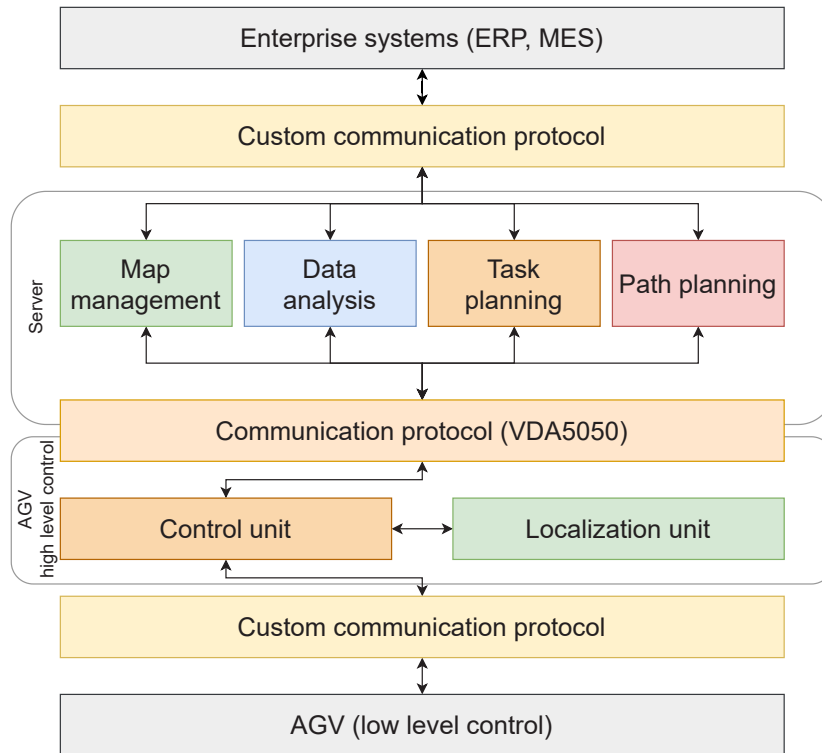


Figure 7: The REX ecosystem

to the fleet of mobile robots, map management, diagnostics and visualizations. All three modules are connected with each other through communication protocols allowing the modularity of the system in accordance with Industry 4.0. They work as stand-alone subsystems usable in other systems as well. The communication protocol with the hardware platform enables independence of suppliers and provides the openness of the system as well as its universality. The server system interacts with companies systems like ERP through a communication interface. It is supposed that production processes are already digitalized to a certain level. Thanks to that, the integration of the system with the customer company will be much faster and easier. The proposed system in this thesis is part of the task planning and scheduling subsystem of the server system.

■ 4.2.2 Integration of plan modification

Depending on the level of automatization of the factory, three stages can be distinguished. In the first stage, the factory has low automatization. In the second one, it is partly automatized. The third stage is a highly automated factory. The REX system can be applied to all three levels.

A low automatization of the factory usually means that material required for production is transported by people. The plan for the transportation of goods is usually done by experienced operators, also known as dispatchers. There are usually not many enterprise systems in the factory. If the REX system is applied, operators can enter

their plan into the REX web application and then robots, which can also be supplied by Datavision, can perform the scheduled tasks. If a failure during the execution of the plan occurs, the system can handle it.

In the partially automatized factory, there can be enterprise systems with manufacturing systems that control the production more automatically. In such factories, the transportation of goods can still be performed by people, but the scheduled tasks are assigned by a system. The operator usually specifies the production plan in the system, and the rest can be controlled on-demand automatically. For example, if there is not enough specific material at a conveyor belt, the system automatically assigns robots to bring the material there. If the REX system is deployed in such a factory, the REX web application is connected through a communication interface to the enterprise and manufacturing systems in which the production plan is specified by the operator and controlled by the system. Scheduled tasks can be then performed by robots, and if a failure appears, our system can fix it.

If the factory is highly automatized, then it usually has enterprise and manufacturing systems that control the production and robots that perform these tasks. These robots are usually not very autonomous so that they only follow a magnetic line. In case a failure occurs during the execution of a plan, there is no automatic solution to that, and it has to be resolved by people. The REX system can contribute to such automatized production by automatic resolution of failures so that the execution can continue.

In all three cases, a plan is either entered directly into the REX web application by an operator or transmitted through a web interface from enterprise and manufacturing systems. Finally, this plan carries the information on which robot should transport which material to which place at what time. Such a plan is then represented by PDDL and sent through the server interface into Plan Monitor, see Figure 8. The Plan Monitor block sends the plan, which is transformed and transmitted through a VDA5050 communication protocol, to a robot's control and execution block. This control unit then moves the robot and controls its pickups and dropoffs to perform the assigned tasks. The feedback from the execution is reported back to the Plan monitor. When some failures occurs, the new plan is found in the Plan repair and sent back to the Plan monitor so that the execution of the plan can continue autonomously.

To preserve the operator's plan as much as possible, the original operator's planning problem has to be reconstructed, and the information gained is then used to repair the plan in this way. This original operator's planning problem is reconstructed at the time when the operator's plan is entered into a system, and it is kept in the Problem knowledge block.

■ 4.2.2.1 Implementation

This section focuses on Plan monitor, Plan repair and Problem knowledge blocks which were implemented in this thesis. The domain file implemented in PDDL 2.1 was already discussed in more detail in section 4.1.1.1.

4.2 SYSTEM

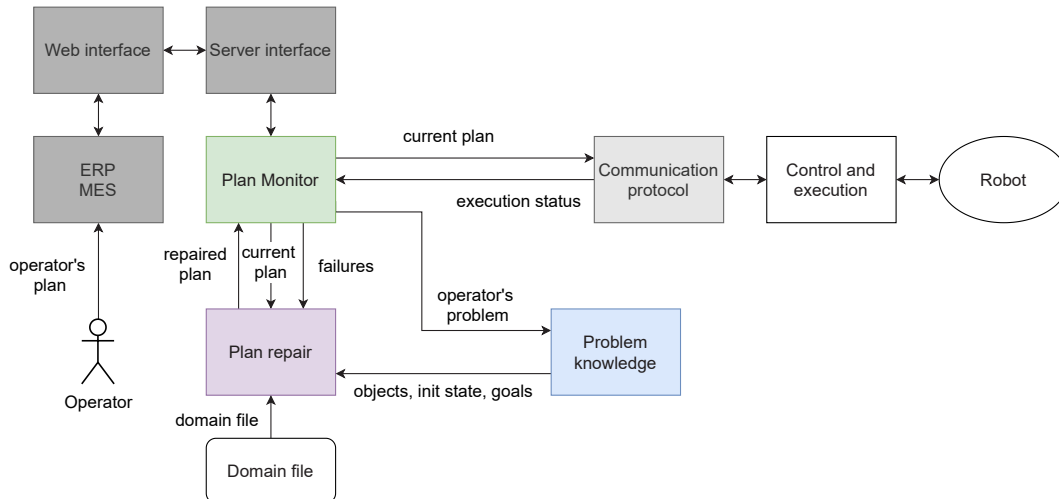


Figure 8: Overview of the REX system relevant to the plan repair

First of all, more details on the working of the plan repair in the system are provided. Finally, implementation details are described at the end of this section.

Plan monitor

The input to the Plan monitor block is a PDDL plan. The whole plan is sent to a robot's control unit and transformed into a behaviour tree which is then executed. The status of the execution is reported back to the Plan monitor, which keeps the information about the current execution state up to date. If a failure occurs, the status that the action was not executed successfully is reported. The Plan monitor also receives the reason for a failure in the form of a PDDL literal. The failure is processed, and the plan repair is triggered.

The processing of a failure also allows a preemptive replanning to some extent. If the Plan monitor receives that an action was not successfully executed, it aborts all actions which are relevant to the affected AGV and should be performed after the failed action. Moreover, the Plan monitor validates the whole plan and determines which actions will also be unsuccessful due to this failure in future. It aborts these actions and all relevant actions after it. Doing so allows also repairing the plan of the so far unaffected robots.

Suppose that *agv0* cannot *drive* from *wp3* to *wp5* because there is an unavoidable obstacle on this path, see the visualisation of plans in Figure 11. The robot will report in the feedback that this action failed and the reason for it is that path from *wp3* to *wp5* is blocked. During the validation of the plan, preconditions of all scheduled actions are checked. If preconditions of an action do not hold because of this failure anymore, the action is aborted. This way, all following actions that should be applied after this action are also aborted. As a result, the rest of the plan of *agv0* will be aborted. Moreover, the action of *agv1* to drive from *wp3* to *wp5* will also be aborted. The rest of the plan of *agv1* since this action will also be aborted, even though *agv1* is still executing previous

actions successfully.

It could be possible to cancel also some previous actions that leads *agv1* to *wp3*. This approach could shorten the plan of *agv1* in the way that it could drive other paths going around the path from *wp3* to *wp5* earlier. But if doing so, fewer actions from the operator's plan would be preserved. This level of preemptive replanning would be valuable only in the replanning from scratch approach. Consequently, this level of the method is not used not to influence the preservation of the original operator's plan.

Problem knowledge

The Problem knowledge block holds the original operator's problem. The input to the REX system is only in the form of a plan. Therefore, the planning problem corresponding to the operator's plan has to be restored. The planning problem is restored in the Plan monitor when an input PDDL plan arrives. This original operator's problem is an input of the Problem knowledge block. The problem is kept until the plan is performed till the end. This problem consists of objects, the initial state and the goal. All these components are required in the Plan repair as described in 4.1.2.1.

Implementation details

The majority of the REX subsystems are written in ROS⁹ 2 in version Foxy¹⁰ in C++. ROS 2 is used because of the following reasons. First, this framework supports the concept of modularity that is independent of a programming language. A package is a set of code that can be written in C++ or in Python (some other programming languages are also supported). Each package is then built and can be used as a standalone system which can be reused in other projects and applications. Second, ROS 2 is intended to be a more real-time¹¹ than its previous version ROS 1¹². Third, the latest version of ROS called Noetic¹³ is supported until May, 2025 and no more versions of ROS will be released¹⁴. Fourth, such a robotics framework provides services, hardware abstraction, low-level device control or passing of messages between processes. All these tools and libraries make development easier. Last, ROS 2 is widespread so that lots of libraries and open-source packages can be found and used. C++ is used because it has better support in ROS 2 than other languages.

Plan monitor, Plan repair and Problem knowledge are ROS 2 packages implemented in C++. Plan repair is an Action server, and Plan monitor is an Action client to this server. Plan monitor sends a request containing the current plan and failures to Plan

⁹Robot Operating System

¹⁰The official documentation of ROS 2 Foxy is available at <https://docs.ros.org/en/foxy/index.html>

¹¹Real-time programming in ROS 2: <https://docs.ros.org/en/foxy/Tutorials/Real-Time-Programming.html>

¹²The official documentation of ROS is available at <http://wiki.ros.org/Documentation>

¹³<http://wiki.ros.org/noetic>

¹⁴Distributions of ROS 1: <http://wiki.ros.org/Distributions>

4.2 SYSTEM

repair. The server responds with a repaired plan in a result of the action message after the request (to repair a plan) is processed. The Action server is used because it is a non-blocking type of service, i.e. the request is processed in another thread so that the Plan monitor is not blocked by waiting for the response. The Plan monitor node cannot be blocked because it has to handle the communication with the control unit of a robot.

Problem knowledge is a Service server that keeps the original operator's problem. Plan repair is a Service client to this server. Plan repair requests the objects, the initial state and the goal from the original operator's problem so that this knowledge can be used to generate a new planning problem in the Plan repair node automatically. This problem is then used with the domain file to provide a new plan fixing the failures in the execution of the operator's plan. This service is necessarily a blocking one because the code in the Plan repair package is dependent on the response of the server and cannot continue without it. Moreover, the processing operation is very fast, so that the Action server is not needed.



Chapter 5

Evaluation

The goal of this chapter is to evaluate the two proposed methods. The organization of this chapter is the following. First, the simplified architecture on which the evaluation was performed is described. Then, the experimental setup in the form of a planning problem is presented. Various testing scenarios follow. They aim to investigate the system's ability to modify the operator's plan when a failure occurs. The metrics assessing the behaviour of the proposed systems focus on the difference from the operator's plan, the total delay of the modified plan from the operator's plan and an average delay of a cargo in the presented case study further. The first criterion evaluates the property for which the plan repair method was designed, and the other two metrics are investigating the properties valuable for industrial operations. Finally, the experimental results are interpreted and compared in the sense of these metrics.

■ 5.1 Testing architecture

The majority of the subsystems (coloured in grey) in Figure 8 are not finished yet, because Datavision is still developing them. Therefore, it is necessary to evaluate the proposed method on the simplified system for testing purposes. This simplified system is depicted in Figure 9.

First of all, an operator's plan is generated by a state of the art planner. Furthermore, this plan and failures occurring at a certain time are processed together in the Current Plan Generator. The output of this block is a current plan in which all actions have a status executed until the time when the failure appeared. Since that time instant, the plan is validated. The failure causes some actions to no longer be applicable. The status of these actions is set to aborted. Recall that this method relevant to preemptive replanning was already discussed in the Plan monitor section in 4.2.2.1.

The generated current plan and failures are input to the simplified Plan Monitor block and formed into an action message. The current plan and failures are reconstructed in the Plan Repair package, in which the current plan is repaired as described in section 4.1.2.1. The Problem Knowledge block provides objects, the initial state and goals from the original operator's problem through service to the Plan Repairer package. The original operator's problem is kept in the Problem Knowledge block and

5.2 GENERATION OF DATA

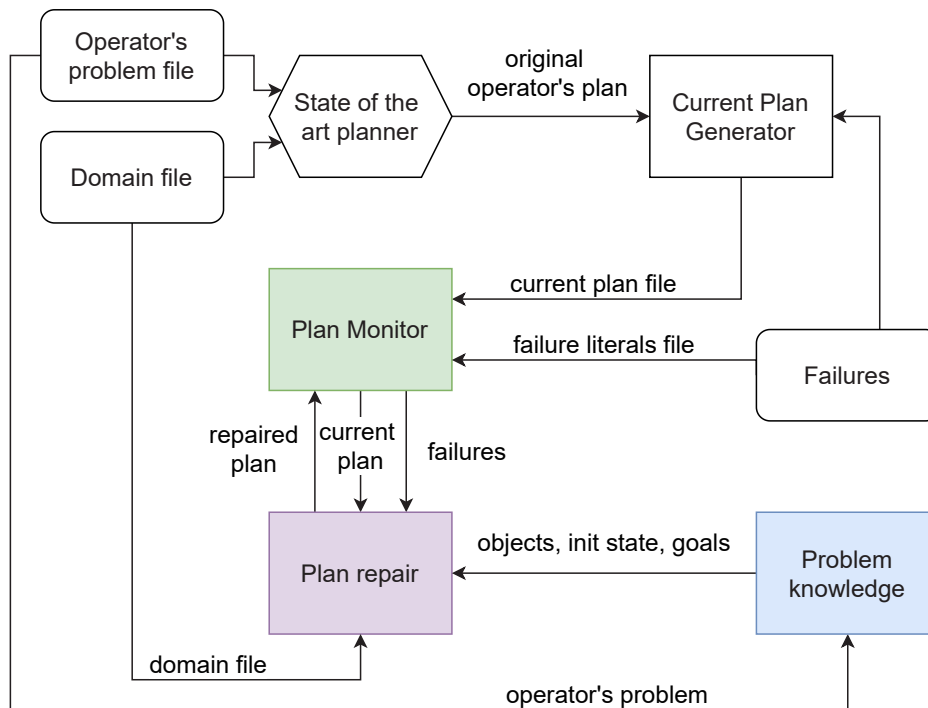


Figure 9: Simplified architecture of the system for testing purposes

initialized from the provided problem file. The generation of the operator's problem file and failures files is the subject of the following section.

■ 5.2 Generation of data

The input data simulating the real execution of a plan in an industrial environment were created to be able to test and evaluate the proposed systems. There are three input files that have to be provided into the system in the testing architecture. These files are the domain file, a problem file and a file containing failures.

The domain file modelling the world was already described in section 4.1.1.1. The problem file is a PDDL planning problem corresponding to the operator's plan. To create the operator's plan, the TFD planner is used, whose inputs are a planning domain and a planning problem. Therefore, the operator's problem file has to be created. The way this problem was created, its description and the resulting operator's plan are discussed in the first part of this section.

The second part of this section deals with the testing scenarios in which two types of failures occur. The time when they can occur and the way it was chosen are discussed in this part. The failure appearing at a particular time then corresponds to the input failures file.

■ 5.2.1 Operator's problem file

To be able to test the proposed methods, it is necessary to have a plan provided by an operator. The plan can be obtained by a state of the art planner, but then a planning problem has to be specified. Therefore, the aim was to create such a planning problem that would correspond to a real industrial environment as closely as possible. However, there are some limitations on the runtime and memory that constrain the task. The process of finding such a planning problem is described in the next section. In the following sections, the final planning problem and the corresponding operator's plan are described in more detail.

■ 5.2.1.1 Analysis of planning problem describing an industrial environment

Our first thought was to generate a warehouse-like environment with 8 AGVs transporting 30 cargos. The map had 78 waypoints, including locations for pickups, deliveries, docking and crossroad points. The problem was first run with the OPTIC planner on a computer with 16 GB RAM and four cores and then on a computer with 32 GB RAM and four cores. No solution was obtained on any of those computers as they ran out of memory.

The problem then was decreased to 3 waypoints - one for pickup, one for delivery and one for docking - to avoid the performance issues. This attempt was also unsuccessful because OPTIC did not provide any solution in 8 hours. Therefore, we decided to decrease the complexity of the problem to only 3 AGVs and 10 cargos so that the solution will definitely be provided. The aim was then to find the largest amount of waypoints for which a solution can be obtained in a reasonable time. This time limit was set to 30 minutes as it is usually used in planning competitions. Two planners were chosen to compare their performance - OPTIC and TFD.

Finding the largest amount of waypoints that can be solved depends a lot on the arrangement of waypoints and their interconnection. There are two possible extremes in how waypoints can be connected. Either, there are all possible connections, i.e. a bidirectional path between every two waypoints. This configuration of waypoints and paths is called *all* and represents a complete bidirectional graph, see Figure 10a. Or, there is a minimal amount of paths connecting waypoints in the way that an AGV can still drive to any possible location. Each waypoint has a maximum of two bidirectional connections so that paths are only between the consecutive locations. As a result, this configuration forms a line of interconnected waypoints as depicted in Figure 10b. Hence, this configuration is called *line*. Between these two extremes, a map for a real industrial environment can be designed. This map can have maximally the number of waypoints as in configuration *all* and the minimal number of edges as in configuration *line* to ensure that the solution of such a planning problem can be found.

I created a script to automatically generate a problem for a certain amount of AGVs, cargos and waypoints in these two configurations. In the problem, all AGVs start and

5.2 GENERATION OF DATA

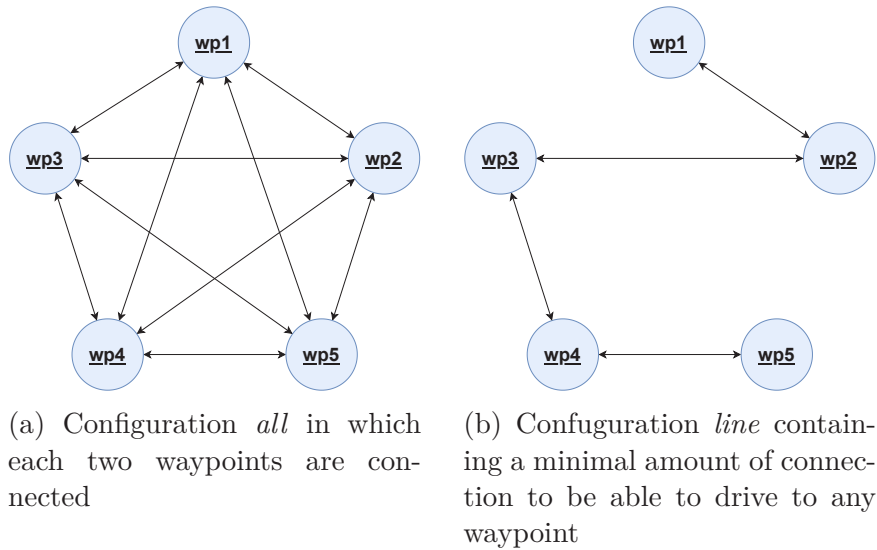


Figure 10: Two extreme configurations of waypoints

finish in the docking position at waypoint $wp0$. All cargos are in the pickup position at waypoint $wp1$ at the start, and their goal positions are equally distributed over all remaining waypoints. Such placement of delivery locations creates a dependence between the number of cargos and waypoints. If the number of waypoints is equal to the number of cargos plus two, i.e. there is exactly one cargo for each remaining waypoint, then there is no significant difference in generated problems for a higher amount of waypoints.

To find the biggest map describing an industrial environment, a set of problems was generated for a fixed number of 3 AGVs and 10 cargos. The amount of waypoints is increasing from 3 to 12 in configurations *all* and *line*. The minimal amount of waypoints is three - one for pickup, one for delivery and one for docking, and the maximum is equal to the number of cargos plus two. This set of problems was run on OPTIC and TFD, and the results are in Table 1.

In Table 1a, it can be seen that configuration *line* is problematic for OPTIC planner because the last provided solution was for 9 waypoints in this configuration. TFD planner solved the biggest problem in the set for both configurations, but it did not provide any solution for two problems, see Table 1b. The makespan is shown for the last problem that both planners solved, which means 12 waypoints for configuration *all* and 9 for configuration *line*. From these results, it can be seen that TFD provided shorter plans than OPTIC. The runtime is shown for the same problems as the makespan. It can be seen that TFD solved these problems in a significantly shorter time than OPTIC.

Because both planners successfully solved all problems in configuration *all*, one more experiment was carried out. The number of AGVs remained the same, and the number of cargos was increased to 20. Thus, the set of problems for two configurations was generated for waypoints going from 3 to 22. The results from experiments on both planners are shown in Table 2.

configuration	max waypoints solved	no solution found for waypoints	makespan	runtime [s]
all	12	-	136	72.44
line	9	10-12	376	1437.3

(a) Results for OPTIC planner

configuration	max waypoints solved	no solution found for waypoints	makespan	runtime [s]
all	12	-	130	0.8
line	12	6-7	300	0.5

(b) Results for TFD planner

Table 1: Results from a set of problems with 3 AGVs, 10 cargos and an increasing number of waypoints from 3 to 12 in configuration *all* and *line* run on OPTIC and TFD planners.

configuration	max waypoints solved	no solution found for waypoints	makespan (wp13)	runtime [s] (wp13)
all	14	12, 15-22	248	500.38
line	6	7-22	-	-

(a) Results for OPTIC planner

configuration	max waypoints solved	no solution found for waypoints	makespan (wp13)	runtime [s] (wp13)
all	22	5, 6-12, 14-17	270	1364.2
line	22	3-17	-	-

(b) Results for TFD planner

Table 2: Results from a set of problems with 3 AGVs, 20 cargos and an increasing number of waypoints from 3 to 22 in configuration *all* and *line* run on OPTIC and TFD planners.

In Table 2, it can be seen that OPTIC solved the maximal number of 14 waypoints in configuration *all* and 6 in configuration *line*. In contrast, TFD provided solution for the biggest problem in the set again. However, it also did not solve several problems in between. The makespan and runtime could be compared only for configuration *all* because the same problem was not solved by both planners in configuration *line*. From the obtained results, it can be seen that OPTIC provided a shorter plan than TFD in a much faster time for a problem with 13 waypoints in configuration *all*.

To sum it up, TFD seems to be good enough to solve interestingly big problems. Still, it has a significant drawback: it is not very deterministic because it did not solve some problems in the set. OPTIC is very deterministic, but it cannot solve big enough problems. Further, TFD showed a better performance than OPTIC for a chosen problem from the smaller set of problems when comparing the length of the plan and

5.2 GENERATION OF DATA

the runtime. However, the results for TFD was significantly worse for a chosen problem from the bigger set of problems than for OPTIC. More analysis would be necessary to properly compare which planner has a better performance for such a problem from the industrial environment. Nevertheless, the performance is not very relevant to the analysis aiming at finding the biggest map.

To conclude, the biggest map of the industrial environment solvable by both planners in 30 minutes can have maximally 14 waypoints, and the amount of connection can be minimally the same as it is for 6 waypoints in the configuration *line*, i.e. 5 bidirectional paths. The resulting map corresponding to an industrial environment was designed for 3 AGVs, 6 cargos and 9 waypoints in the end, see Figure 2. The additional requirements for the map were that it should be small enough so that a planner will provide a solution for it for sure and so that testing scenarios can be easily shown, imagined and demonstrated on it.

■ 5.2.1.2 Initial problem file

This section describes the final design of the operator’s problem representing the pickup-and-delivery problem in an industrial environment such as a factory concerning production and including logistics. This problem is described in the motivational example in section 1.2 in general. A detailed description of the problem in PDDL is provided in this section. The complete code is attached in Appendix A.2.

The map that is described by a PDDL problem is depicted in Figure 2. There are nine *waypoints*, wp_0 to wp_8 , representing locations. The storage center is in wp_0 , where *cargos* are picked up. Locations for deliveries of these *cargos* are marked by waypoints wp_2 to wp_7 . Each robot starts and ends in the docking centre, denoted by the waypoint wp_1 . Waypoint wp_8 is only a crossroad. The fleet consists of three *agvs* that deliver six *cargos* in this problem. To sum up, there are the following objects used in the problem - three AGVs, agv_0 to agv_2 , six cargos, $cargo_0$ to $cargo_5$, and nine waypoints, wp_0 to wp_8 .

In the initial state, the initial properties of these objects are defined. All three *agvs* are *alive* and *at* the docking center, i.e. wp_1 . All six *cargos* are *at* the storage center, i.e. wp_0 . The *paths* are bidirectional between each two *waypoints* according to the Figure 2. The weights of these edges are set as the *travel times* in both directions.

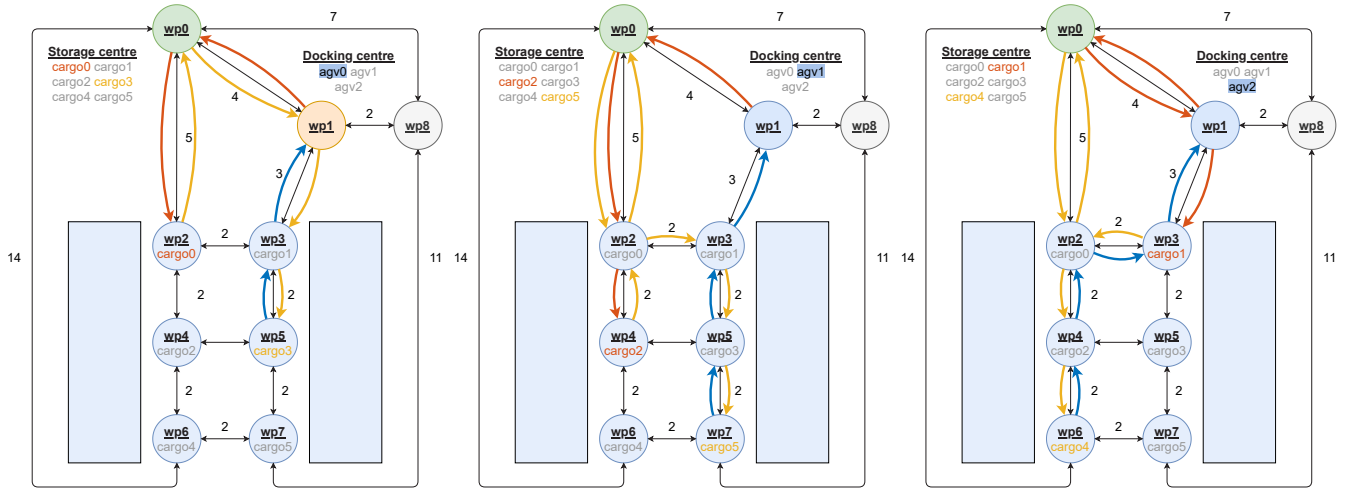
The goal state describes the state after this pickup-and-delivery problem is satisfied. This means that all *agvs* are back in the docking centre, i.e. wp_1 and all *cargos* are on the requested positions. The requested positions are marked by the green text in the Figure 2 and yields $cargo_0$ at wp_2 , $cargo_1$ at wp_3 , $cargo_2$ at wp_4 , $cargo_3$ at wp_5 , $cargo_4$ at wp_6 , $cargo_5$ at wp_7 .

■ 5.2.1.3 Initial PDDL plan

A state of the art planner, more specifically TFD, was used to obtain a plan from the described PDDL domain and corresponding initial PDDL problem. The plan is

attached in Appendix A.3.

The planner decided to assign $cargo_0$ and $cargo_3$ to be served by agv_0 , $cargo_2$ and $cargo_5$ by agv_1 , and $cargo_1$ and $cargo_4$ by agv_2 . The paths along which AGVs should drive are depicted in Figure 11. The first journey to serve the first assigned cargo is coloured in red, the second in yellow and the remaining blue colour denotes the trip back to the docking station.



(a) Path describing plan of agv_0 serving $cargo_0$, $cargo_3$ and return-
to dock (b) Path describing plan of agv_1 serving $cargo_2$, $cargo_5$ and return-
to dock (c) Path describing plan of agv_2 serving $cargo_1$, $cargo_4$ and return-
to dock

Figure 11: Visualization of plans of individual AGVs. The red colour marks paths required to deliver the first assigned cargo, the yellow colour to deliver the second assigned cargo and the blue colour denote paths necessary to get to the docking centre.

5.2.2 Testing scenarios

The proposed methods are tested with two types of failures - failure of an AGV and failure due to a blocked path. Any other types of failures are not in the scope of this thesis.

A failure of an AGV is represented by a literal (`not (alive ?agv)`), where $?agv$ is the name of the AGV which failed. The real reason for a failure of a robot occurred during the execution can be for example that a robot is out of the battery, a connection to a robot is lost, or a hardware component like motors, a computer, a lift, etc. are broken or stuck. All these cases are expected to get transformed into this literal.

A failure due to a blocked path can happen because of an obstacle on this path. Recall that it is assumed that all AGVs are smart enough to avoid each other on a path and that there is always enough space for such manoeuvres. They can also avoid small obstacles such as an AGV that failed on a path or a waypoint. It is also assumed that if there is some obstacle, it is always in both directions because the path is considered

5.2 GENERATION OF DATA

to be a single bidirectional line. Therefore, this failure is represented by two literals - $(\text{not } (\text{path } ?wp1 ?wp2))$ and $(\text{not } (\text{path } ?wp2 ?wp1))$, where $?wp1$ and $?wp2$ are some waypoints.

The testing scenarios are designed to test the proposed methods fixing these two types of failures occurring during the execution of the initial PDDL plan attached in Appendix A.3. The testing scenarios are divided into two parts, each relating to one type of failure. There are 42 different interesting scenarios in total in which a failure can occur and affect the plan. The discussion of how the testing scenarios were selected follows in the next two sections.

■ 5.2.2.1 Generation of AGV failure

There are three interesting time instants when a failure of an AGV can happen:

- before the start of the plan execution,
- before the execution of a load action and
- after the execution of an unload action.

Notice that failure occurring during the time when a cargo is in an AGV is not tested. The cargo that is in an AGV is stuck there and cannot be accessed. To fulfil the task of delivering this cargo, another cargo of the same category has to be transported to the delivery place. For example, suppose that an AGV will encounter a failure while transporting a box of car wheels. If there is another box of identical car wheels, then the task can still be fulfilled. To satisfy this task, another AGV has to pick up this box in the storage centre and transport it to the delivery place. As a result, this case is very similar to the situation when a failure of an AGV occur before the execution of a load action. In both cases, another AGV has to serve that cargo. Therefore, it is not necessary to also investigate the situation when a failure of AGV appears during the transportation of goods.

There are also various driving actions during which a failure of an AGV can occur. However, all these situations are very similar to each other, and they do not bring any further interesting information to the already selected three time instants.

Each AGV is assigned to serve various cargos to different locations. Hence, there is a slight difference in testing scenarios depending on an AGV that failed and depending on the number of cargos served. There are five interesting time instants for an AGV when its failure can occur:

- before the start of the plan execution denoted as *dead_agv_before_start*,
- before the execution of a load action of the first transported cargo, denoted as *dead_agv_before_1st_load*,

- after the execution of a unload action of the first transported cargo, denoted as *dead_agv_after_1st_unload*,
- before the execution of a load action of the second transported cargo, denoted as *dead_agv_before_2nd_load*, and
- after the execution of a unload action of the second transported cargo, denoted as *dead_agv_after_2nd_unload*.

There can be a failure of one AGV at any moment or two of them. If there is a failure of three AGVs, the task cannot be fulfilled. There are five interesting time instants when a single AGV can fail, and there are three AGVs to test on these cases, generating 15 testing scenarios in total.

Further scenarios should investigate the failure of two AGVs. These failures occur at the same time instant for simplicity. Failures are also set to appear in these five interesting time instants. However, both AGVs do not drive the same path at the same moment, so that these five time instants are not the same for both robots. Failures are simulated to occur in the way that they appear before a load action of the first of two investigated AGVs executing this action and after an unload action of the second of two investigated AGVs performing this action. As a result, none of these AGVs fails during the transportation of goods. There are three combinations of AGVs that can fail - *agv0* and *agv1*, *agv1* and *agv2*, *agv0* and *agv2*. For these three combinations and five time instants, 15 more testing scenarios are generated in total.

In total, there are 30 testing scenarios investigating a failure of an AGV in various time instants for different AGVs and their combinations, which should thoroughly examine the behaviour of the proposed methods.

■ 5.2.2.2 Generation of failure due to blocked path

Paths are selected to show the behaviour of the proposed system if one, two or three AGVs encounter a blocked path during the execution of their plans. From Figure 11, paths through which AGVs are driving during the execution of their plan were analysed, and the appropriate, interesting paths were selected based on this analysis. In Figure 11, it can be seen that only *agv2* is driving through the path between waypoints *wp4* and *wp6*. This situation is denoted with the prefix *path_1agv*. Further, it can be seen that *agv2* and *agv1* are both driving through the path between waypoints *wp2* and *wp4*. This case is named as *path_2agv*. Finally, all AGVs are driving through the path between waypoints *wp1* and *wp3*. This situation is called *path_3agv*. Blocking of these three bidirectional paths is tested to investigate the behaviour of the system when a failure due to a blocked path occurs.

There are also many time instants when this failure can appear. However, only three time instants are interesting to investigate because other cases are similar to them. Thus, the investigated time instants when a path becomes blocked are:

5.2 GENERATION OF DATA

- before the start of the plan execution denoted with the suffix *before_start*,
- before the execution of action drive through that path, named as *before_path*, and
- after the execution of action drive through that path, called *no_return*, because an AGV cannot return the same way back.

If more AGVs encounter a blocked path during the execution of their plans, these time instants are modified as follows. For two AGVs scheduled to drive through such a path, it can be further distinguished that the path is blocked:

- before the execution of action drive of the AGV that is scheduled earlier to drive through that path, denoted as *before_1st_path*, and
- after the execution of action drive of the AGV that is scheduled earlier to drive through that path, denoted as *after_1st_path*, in which *agv1* is encountering *no_return* scenario and *agv2* still *before_path* scenario.

No more time instants can be investigated for these two AGVs because then only one AGV would be scheduled to drive through a blocked path. There is only one possible different combination of AGVs (namely *agv0* and *agv1*) that could follow the same pattern for testing. However, this combination would not bring any additional interesting information about the behaviour of the system.

For three AGVs, the path is further blocked:

- after the execution of action drive of the AGV that is scheduled to drive through that path as the second one since the beginning denoted as *after_2nd_path*, and
- before the execution of action drive of the AGV that is scheduled to drive through that path as the third one since the beginning denoted as *before_3rd_path*

Further time instants would not test the behaviour of the system when three AGVs failed due to a blocked path. Moreover, only these two mentioned scenarios and the case *before_start* are investigated because other time instants are very similar to them and thus, do not bring any more information about the proposed system.

Finally, three more specific situations were created. First, all paths to the waypoint *wp4* are blocked before start and after *agv1* unloads a cargo there. No plan is provided by a state of the art planner for these two cases because the goals cannot be anyhow satisfied. Therefore, these two scenarios were not considered in the final evaluation. Second, there are three blocked bidirectional paths so that AGVs are forced to use the path between waypoint *wp0* and *wp6*. The failure occurs before the start of the execution of plans, and therefore, it is named *path_force_wp0_wp6_before_start*. The third situation is similar. There are also three blocked bidirectional paths before the start of the execution of plans. In this case, AGVs are forced to drive through the

waypoint *wp8*. These two last scenarios are designed to force AGVs to go around the conveyor belt, see Figure 2.

To conclude, there are 14 different scenarios in total investigating the failure due to a blocked path that should properly test the ability of the system to repair the plan. In total, there are 44 testing scenarios for both types of failures together. All of them are designed to examine the system in various situations and evaluate its behaviour according to the metrics described in the next section in more detail.

■ 5.3 Metrics description

Three metrics were chosen to compare the ability, behaviour and performance of the two proposed methods - plan repair and replanning from scratch. The most crucial measure for the evaluation is a *plan difference*, in which the modified plan is assessed how much different it is from the operator's plan. Further, the proposed methods are compared from the viewpoint of the *total plan delay*. The last metric is the *average cargo delivery delay*, in which the average delay of a cargo delivery in the plan is investigated.

The first metric examines the ability of the method to preserve as much of the operator's plan as possible to persuade the companies to trust a system with a minimal amount of AI, which can automatically handle the failure in the execution of a plan. In contrast, the two remaining measures evaluate the properties significant for operations in industrial environments. All three metrics are described in the following sections in more detail.

One more metric was considered to analyze - the *runtime*. This measure examines how much time is required for the execution of the plan when a failure occurs and is fixed by the modification of the plan. This metric could show a difference between the plan repair method that assumes the unaffected part of the plan to be finished and therefore, solves a smaller subset of the planning problem, and replanning from scratch solving the problem corresponding to the entire remaining plan. However, the runtime was around one second for both methods and all testing scenarios, so that this hypothesis cannot be tested on such small planning problems because the effect will not occur with respect to the makespan. The difference could be significant for much bigger problems, but there are difficulties to solve bigger problems by various planners as discussed in 5.2.1.1.

■ 5.3.1 Plan difference

The aim of this metric is to evaluate how the modified plan varies from the original operator's plan. This measure is taken from [34], where the metric is called *plan stability* and is computed as a difference between two plans.

This difference is obtained as the sum of *additional* and *missing* plans steps which penalizes the modified plan compared to the original plan. The *additional* plan steps

5.4 EXPERIMENTAL RESULTS

are the actions that are present in the modified plan but not in the operator's plan. The *missing* plan steps are the actions that are contained in the operator's plan but not in the modified plan, and thus, they are missing in the modified plan.

Both these parts of the sum are important. For example, suppose that *agv2* is forced to drive through a path between waypoints *wp0* and *wp6* that is one driving action instead of three driving actions as in the original plan to deliver the *cargo4* as visualised in Figure 11c. If only *additional* actions are counted, the difference of these plans is only one action. But in reality, the difference is bigger because three plan steps are not contained in the modified plan. Therefore, also the number of *missing* actions is crucial for the evaluation of the difference between the two plans.

■ 5.3.2 Total plan delay

This metric evaluates the delay of the entire modified plan compared to the original operator's plan. This delay is computed as the difference of makespans of the modified and operator's plans. The delay is then expressed as a percentage of the makespan of the operator's plan so that it is possible to evaluate by what percentage the modified plan is longer than the operator's plan.

■ 5.3.3 Average cargo delivery delay

This metric investigates the average delay of a cargo. The process of how it is measured is explained further.

Each cargo is found in the modified plan and paired with the corresponding cargo in the operator's plan. The delivery time of a cargo is computed as the sum of the scheduled time of the action unload and its duration. The delay of the delivery of a cargo is a difference between the delivery time in the modified plan and in the operator's plan. This delay is computed for each cargo in the plan. From these delays, the mean is obtained. Finally, this average is expressed as a percentage of the makespan of the operator's plan so that it is possible to investigate by what percentage a cargo will be delivered later in average in the modified plan than in the operator's plan.

■ 5.4 Experimental results

The proposed system is evaluated on 42 various testing scenarios by three measures. Six interesting testing scenarios are selected from this set to discuss the results of the measures in more detail. Discussion for each metric follows in the next sections. Statistical values such as mean and standard deviation are computed for each metric on the set of all testing scenarios.

Two sets of experiments were performed. In the first one, the plan repair method was used to fix the execution of the plan when a failure occurred, and in the second

one, the replanning from scratch method was used. In these experiments, the TFD planner was chosen as a state of the art planner providing plans for the affected part of the plan in case of the plan repair method and the remaining part of the plan in case of replanning from scratch. This planner was chosen because it provided plans with a shorter makespan in a shorter runtime, as analyzed in 5.2.1.1.

■ 5.4.1 Plan difference

The results of the measure *plan difference* for 6 selected interesting testing scenarios are depicted in Figure 12. Three scenarios are picked for each type of failure. For the failure of an AGV, there is a scenario *06_dead_agv1_before_start* in which a plan from replanning from scratch differ significantly from the operator’s plan, whereas a plan from plan repair does not. In the next scenario *08_dead_agv1_after_1st_unload*, there is only a slight difference between the proposed methods, and in the last scenario *25_dead_agv1_agv2_after_2nd_unload*, there is no difference between them. For the failure due to a blocked path, there is a scenario *32_path_1agv_before_path* in which also no difference between the methods can be observed. In the next scenario *40_path_3agv_before_3rd_path*, there is a slight difference between them. In the last scenario *44_path_force_wp8_before_start*, the plan from the plan repair method is more different from the operator’s plan than the plan from replanning from scratch method. These results are revisited and interpreted in the following six paragraphs.

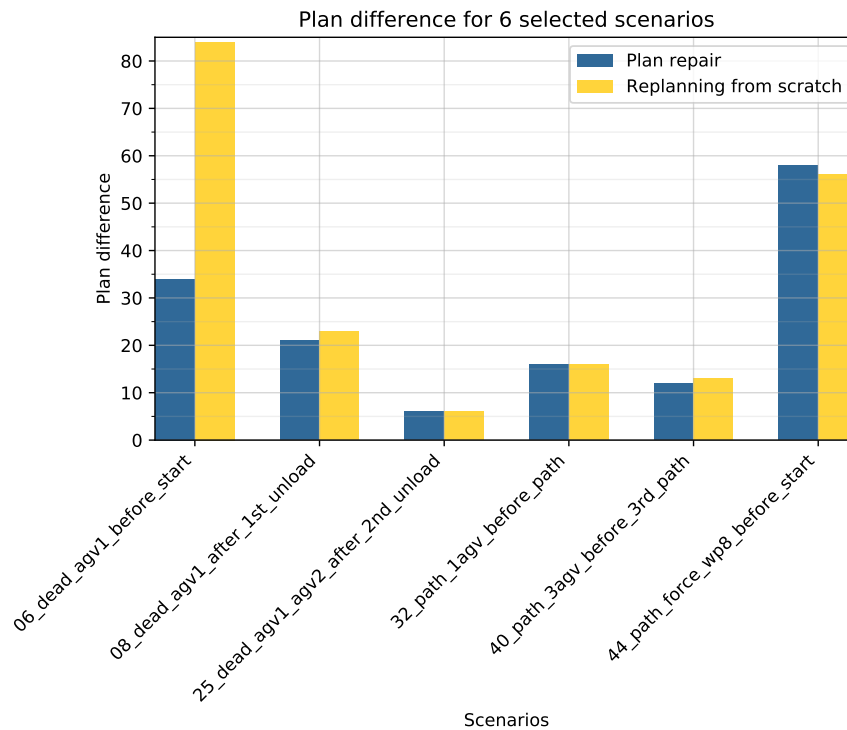


Figure 12: Plan difference

5.4 EXPERIMENTAL RESULTS

In *06_dead_agv1_before_start* scenario, the plan repair method is much better than the replanning from scratch because a plan from plan repair method is significantly less different from the operator's plan than a plan from replanning from scratch. In this scenario, the failure of *agv1* occurs before the start of the execution of a plan. The plan repair method assumes that plans of *agv0* and *agv1* are finished and then solves only the task of transporting cargos that *agv1* should have served. Whereas replanning from scratch method solves the task of transporting all cargos by the two remaining alive AGVs and the planner is optimizing the length of the plan. Thus, this plan can differ a lot, and it does. Therefore, the results make sense and are expected.

In *08_dead_agv1_after_1st_unload* scenario, the plan repair method is slightly better than replanning from scratch. In this scenario, the failure of *agv1* occurs after it unloads the first transported cargo. The remaining AGVs also managed to serve the first transported cargo. There are not many plan steps till the end of the plan, and AGVs are in different positions so that the planner can provide a very similar plan to the operator's plan even when replanning from scratch. There can be only two significant differences - to which AGV the second cargo of *agv1* is assigned, and whether this AGV has to visit the docking place before serving this cargo. These two cases explain why the results for the proposed methods are only slightly different.

In *25_dead_agv1_agv2_after_2nd_unload* scenario, the plan difference is the same for both proposed methods. In this scenario, the failure of *agv1* and *agv2* occurs after all AGVs transported all cargos. Moreover, *agv0* is already back in the docking centre. Thus, there are no more tasks to be planned. Therefore, there is no difference between the plan repair and replanning from scratch. The plan difference is equal to the number of *missing* actions which correspond to the plan steps necessary for *agv1* and *agv2* to drive back to the docking centre.

In *32_path_1agv_before_path* scenario, the proposed methods has the same plan difference again. In this scenario, the failure due to a blocked path between waypoints *wp4* and *wp6* occurs right before *agv2* will drive through it. Because *agv2* is transporting already the second cargo, and *agv0* and *agv1* have already begun to transport their second cargo, there are not many possibilities of various plans to finish the task. The planner will provide the same rerouting of *agv2* through the same paths. Therefore, there is no difference between the plan repair and replanning from scratch.

In *40_path_3agv_before_3rd_path* scenario, the plan repair method is only slightly better than replanning from scratch. In this scenario, the failure due to a blocked path between waypoints *wp1* and *wp3* occurs before third AGV, which is *agv0*, is scheduled to drive through this path. In this case, *agv0* already managed to serve all its cargos, *agv1* and *agv2* are almost finished with the transportation of their second cargo. All AGVs are affected by the blocked path because they are scheduled to drive through it to the docking centre. There are also not many possibilities of various plans, but still, a plan from replanning from scratch differs more than a plan from plan repair. In plan repair, AGVs are executing the same plan as the operator's one until the last possible path, and then another path is used to avoid the blocked one. Whereas in replanning from scratch, the blocked path is avoided earlier because the planner is optimizing the length

of the plan. Because of this small detail, plan repair differs less from the operator’s plan than replanning from scratch.

In *44_path_force_wp8_before_start* scenario, a plan from plan repair method differs more than a plan from replanning from scratch. In this scenario, paths between waypoints *wp1* and *wp3*, *wp2* and *wp3*, *wp6* and *wp7* are blocked before the start of the execution of the plan so that all AGVs are forced to drive through waypoint *wp8*. The plans from the proposed methods are almost the same, but they differ in two plan steps. In the plan repair method, the plan of *agv1* is reused till it loads *cargo2* and drive from *wp0* to *wp2*. But then TFD finds the plan in which *agv1* should first unload *cargo2* at *wp2* and drive back to *wp0*, where *cargo4* is loaded first. In contrast, a plan from replanning from scratch is coincidentally the same in the beginning (meaning *cargo2* is served by *agv1*), but then it immediately loads *cargo4* at *wp0*. The plan difference then varies by two extra actions that are reverted in the plan from plan repair compared to the plan from replanning from scratch.

The ability to preserve as much as possible from the operator’s plan is also investigated throughout the whole set for both proposed methods. The statistical values describing the set are shown in Table 3, and the graph depicting all results is attached in Appendix B in Figure 15. It can be seen that plan repair has a much smaller mean and standard deviation than replanning from scratch. These results indicate that majority of data are in a smaller range for plan repair than for replanning from scratch, and thus, the behaviour of the plan repair method is more expectable than the behaviour of replanning from scratch, which seems to be more random. This fact is also supported by the smaller range of extreme values for plan repair than for replanning from scratch.

Table 3: Results of statistical values for *plan difference* for all testing scenarios

	plan repair	replanning from scratch
mean	30.262	46.119
std	19.076	28.043
min	2.000	2.000
max	72.000	84.000

To sum it up, while there are cases in which plan repair method might be worse than replanning from scratch or they can be equally good from the viewpoint of the plan difference metric, there are the majority of cases in which plan repair method is better than replanning from scratch, i.e. a plan from the plan repair method differs from the original operator’s plan less than a plan from replanning from scratch.

■ 5.4.2 Total plan delay

The same six selected scenarios that were already discussed for plan difference are also evaluated from the viewpoint of total plan delay, and their results are shown in Figure 13. The total delay of the plan is interpreted for each scenario in separate paragraphs.

5.4 EXPERIMENTAL RESULTS

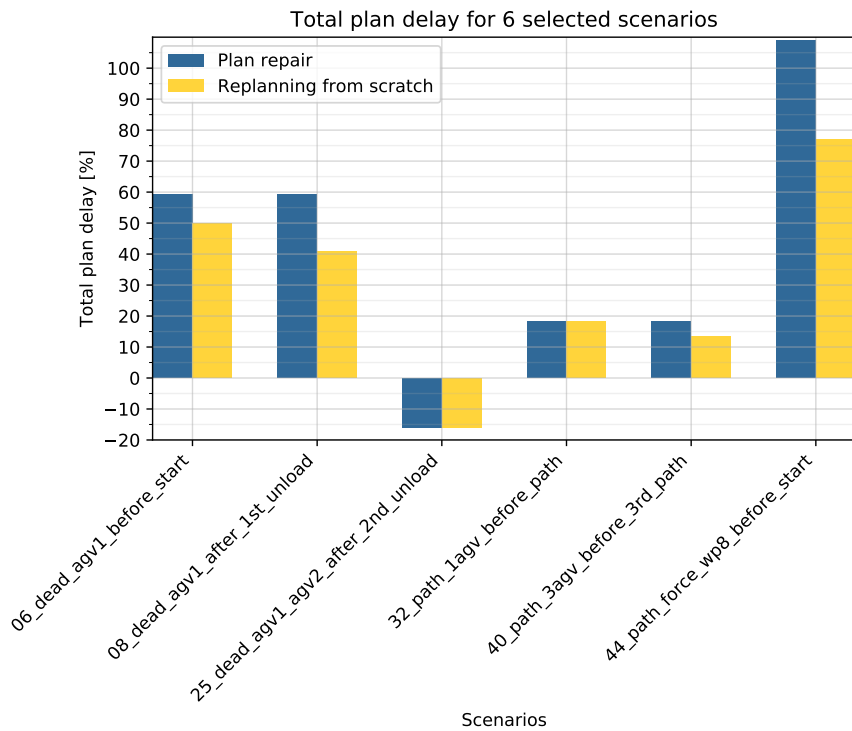


Figure 13: Total plan delay

In *06_dead_agv1_before_start* scenario, plan from plan repair method is longer than plan from replanning from scratch. The difference between these methods is caused by reusing the operator’s plan in which an AGV is driving into the docking centre at the end of the plan. Thus, *agv2* has to go to the dock first and then it can serve one of the remaining cargos of failed *agv1*. The actions necessary to drive to the dock first causes a bigger plan delay than for replanning from scratch method, which can optimize the makespan for the whole plan in this case. For both methods, the delay corresponds to the number of actions connected with cargos that have to be served by remaining alive AGVs.

In *08_dead_agv1_after_1st_unload* scenario, the difference between the total plan delay for both method is bigger. In this scenario, TFD is assigning the remaining cargo of *agv1* to the rest of the fleet. In the plan repair method, this cargo is assigned to *agv2* without any additional information because it is assumed that both AGVs start at the docking centre as they already managed to transport all their cargos. Whereas in replanning from scratch, the makespan is optimized for the rest of the plan so that the cargo is assigned to *agv0* because the planner computed that *agv0* can fulfil it earlier than *agv2*. This decision causes a much bigger delay of the plan from plan repair than of the plan from replanning from scratch.

In *25_dead_agv1_agv2_after_2nd_unload* scenario, the length of both plans is shorter than the operator’s plan approximately by 15%, and thus, the value is negative in this case. The reason for such a value is that all tasks are already fulfilled,

and there is nothing left to plan for. Both plans have a smaller makespan than the operator’s plan because *agv1* and *agv2* cannot drive back to the docking centre due to their failure. These aborted actions correspond to approximately 15% of the length of the operator’s plan.

In *32_path_1agv_before_path* scenario, both methods modified the plan in the same way by rerouting *agv2* through identical paths as discussed already for plan difference metric. Therefore, there is no difference between these methods. The delay of the plan corresponds to the rerouted plan of *agv2* which represents a delay by nearly 20% of the operator’s plan.

In *40_path_3agv_before_3rd_path* scenario, plan repair is reusing the operator’s plan until the last possible path. In contrast, replanning from scratch can start avoiding the blocked path earlier and thus, the planner has provided a plan with optimized makespan in this case. Therefore, the delay of the plan from plan repair is slightly higher than the delay of the plan from replanning from scratch. The difference between the proposed method is small because the remaining plan to be finished is short, and there are not many possibilities of various ways how the operator’s plan can be modified.

In *44_path_force_wp8_before_start* scenario, both methods modify the operator’s plan in almost the same way. The difference for the plan repair method is in two extra plan steps of *agv1* whose plan is reusing the operator’s plan until the last possible action. These two actions are then immediately reverted, resulting in four redundant actions in total. Therefore, the makespan of the plan from plan repair is much longer than the makespan of the plan from replanning from scratch. Both plans have significant delays compared to the operator’s plan because AGVs are forced to drive through the path between waypoints *wp8* and *wp7* which takes more time than if they could drive through blocked paths.

The total delay of the modified plan was also investigated for all testing scenarios, and the corresponding graph with results is attached in Appendix B in Figure 16. Statistical values describing the total plan delay metric on this set are shown in Table 4. It can be seen from the table that the mean is higher for plan repair than for replanning from scratch. This result is expectable because replanning from scratch can optimize the remaining part of the plan. In contrast, plan repair is reusing the operator’s plan until the last possible action, which does not have to be the fastest solution. Nevertheless, the values do not vary much for both methods. The standard deviation is similar for both approaches, and it is relatively high, indicating that data are distributed in a quite wide range. Therefore, the behaviour of both systems seems to be rather random than expectable when considering the makespan of modified plans. The range of extreme values is slightly bigger for the plan repair method, which can correspond to the somewhat higher mean value.

To summarize the results for this metric, there are more cases in which plan repair produces a longer plan compared to the operator’s plan than replanning from scratch. Still, there are also cases in which both methods provide equally delayed plans. This behaviour is supported by the fact that replanning from scratch can optimize the entire rest of the plan. In contrast, plan repair is reusing the operator’s plan until the last

5.4 EXPERIMENTAL RESULTS

Table 4: Results of statistical values for *total plan delay* for all testing scenarios

	plan repair	replanning from scratch
mean	67.333	61.143
std	56.772	56.799
min	-15.924	-15.924
max	181.753	172.639

possible action so that it does not have to provide the shortest plan. On the contrary, the statistical investigation across all testing scenarios does not indicate that this behaviour could be expectable.

5.4.3 Average cargo delivery delay

In this section, the six selected scenarios are investigated from the viewpoint of the deliveries of cargos. The average delay of a cargo delivery for these scenarios is depicted in Figure 14 and discussed in the following paragraphs in more detail.

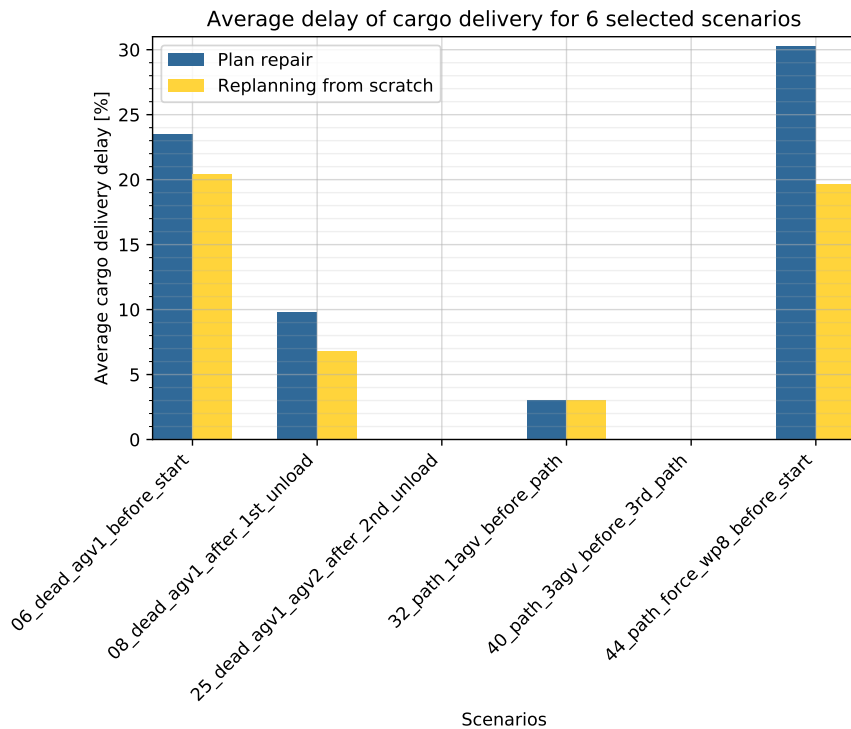


Figure 14: Average cargo delivery delay

In *06_dead_agv1_before_start* scenario, *agv1* should have transported two cargos - *cargo2* and *cargo5*. Only these two cargos are delayed in the plan from the plan repair

method, and all others are delivered as scheduled in the operator’s plan. In the plan from replanning from scratch method, four cargos are delayed, but because the planner could optimize the makespan for the whole plan, the late deliveries are spread across the entire plan and yield a shorter delay of a cargo delivery on average.

In *08_dead_agv1_after_1st_unload* scenario, plans from both methods differ only in the AGV that is assigned to serve the remaining cargo after the failure of *agv1*. Because replanning from scratch is optimizing the entire rest of the plan, this cargo is assigned to *agv0* that can serve it earlier than *agv2*. As a result, the delay of this cargo is shorter than in the plan from plan repair. Because this is the only difference from the operator’s plan, it also corresponds to the average cargo delivery delay.

In *25_dead_agv1_agv2_after_2nd_unload* scenario, all cargos were already transported and almost the entire plan executed. Thus, the delay of all cargo deliveries is zero in this case. Therefore, the average delay of a cargo delivery is also zero.

In *32_path_1agv_before_path* scenario, *agv2* is affected by the blocked path during the transportation of its second cargo and hence, is rerouted to fulfil this task. Both proposed methods modified the operator’s plan in the same way. Only the second cargo of *agv2* is delayed because of the failure, and all other deliveries are performed as scheduled in the operator’s plan. Therefore, the delay of this cargo corresponds to the average delay of a cargo delivery in this scenario and is approximately only 3% of the length of the operator’s plan.

In *40_path_3agv_before_3rd_path* scenario, the path become blocked during the transportation of the second cargo of *agv1* and *agv2*, *agv0* already served both its cargos. The blocked path does not influence the transportation but only returning back to the docking centre. Therefore, there is no delay in cargo deliveries, and thus, the average delay of a cargo delivery is zero, too.

In *44_path_force_wp8_before_start* scenario, the plan repair method is reusing the operator’s plan so that it gives two extra actions compared to the replanning from scratch. These two actions are then reverted in order to serve *cargo4* in the plan repair method. Because of these four redundant actions in plan repair, *cargo4* is delayed. In contrast, replanning from scratch coincidentally managed to deliver this cargo on time. This delay in the plan repair method is cumulated into the delay of *cargo5* that is delivered by the same AGV afterwards. The blocked path affects the plan of *agv1* and *agv2*, resulting in the delay of all their transported cargos. Therefore, the average delay of a cargo delivery is relatively high. The value is smaller for replanning from scratch than for plan repair because the latter is additionally delayed by four redundant actions compared to the former one.

The analysis of an average delay of a cargo delivery was examined for all testing scenarios, and the results are depicted in Figure 17 in Appendix B. The statistical values describing this set of results from the viewpoint of average cargo delivery delay are shown in Table 5. The table shows that mean values and standard deviations are very similar for both proposed methods, and they are slightly higher for the plan repair method. The standard deviation is not very high for both approaches indicating that

5.5 SUMMARY AND DISCUSSION

majority of data is in a rather small range, and thus, the behaviour of the proposed methods is quite expectable, as for average cargo delivery delay. The difference between the extreme values is more significant when considering both methods. The higher maximum of the plan repair method can correspond to the fact the as much as possible of the operator's plan is reused, which might cause some local delays of deliveries. Compared to replanning from scratch, this method can optimize the rest of the plan and thus, the delay of cargo deliveries might be smaller.

Table 5: Results of statistical values for *average cargo delivery delay* for all testing scenarios

	plan repair	replanning from scratch
mean	24.810	21.343
std	28.963	26.373
min	0.000	-0.021
max	92.401	80.252

To sum it up, there are more cases in which the average delay of a cargo delivery is higher for plan repair than for replanning from scratch. There are also cases in which the average delay is equal for both methods or none in the considered set. While the statistical investigation did not reveal very convincing results on which method performs better, it is more expectable that a cargo will be delivered later on average for the plan repair method than for replanning from scratch. The reason for this is that replanning from scratch can optimize the makespan of the remaining part of the plan, and thus, cargos are optimized to be fulfilled at the earliest time. In comparison, plan repair is reusing as much as possible from the operator's plan, which might cause longer delays in deliveries.

■ 5.5 Summary and discussion


This section discusses the results obtained from two performed experiments and evaluated on three metrics. To assess the abilities and behaviour of both proposed methods, all three metrics have to be considered together. First, the plan repair method is summarized, then replanning from scratch and at the end, there is a comparison and final evaluation of both approaches.

Plan repair method modifies the plan in the way that it differs less from the original operator's plan than the second investigated approach. This smaller *plan difference* means that this method can preserve more from the operator's plan than the second method. From the statistical investigation, this result is more expectable than for the second method. This good performance is redeemed by the length of the plan. The aim to minimize the difference from the operator's plan reduces the operation room for the rest of the plan, which cannot be optimized so effectively due to this. As a result,

total plan delay against the operator's plan can be bigger than in the second method. However, this rule is not highly supported by statistical values describing the set of all tested scenarios. The high standard deviation indicates that data are distributed in a broader range, and thus, the behaviour of the system related to the plan delay is rather random than expected. Analysis of *average cargo delivery delay* revealed that a cargo could be more delayed from the operator's schedule on average than in the second method. This fact is closely related to the two previous metrics. The plan repair method reuses as much as possible of the operator's plan so that it can be longer due to this, and it can also deliver cargos later. The reason is still the same - there is not much of the rest of the plan left to optimize in time. However, statistical values describing the entire set are similar for both methods, and thus, this rule is not very convincing. Nevertheless, the standard deviation indicates that this behaviour is rather expectable.

Replanning from scratch is not designed to reuse the majority of the operator's plan. It provides a whole new plan for the remaining goals of the operator's plan from the current state. Therefore, the modified plan obtained from this method differs from the operator's plan more than the first proposed method. Moreover, the *plan difference* investigated on the set of all tested scenarios has a significantly higher mean value with a small standard deviation from it, indicating that majority of data are expected in this range and thus, to differ more from the operator's plan. The advantage of this method is reflected in *total plan delay*. The plans modified by this method can be optimized in time so that the makespan of the plan from this method can be shorter than for the first one. However, the standard deviation from the mean value on the set of all testing scenarios is high, and thus, this behaviour of the system related to the plan delay is rather random than expected. For the same reason, the *average cargo delivery delay* can also be smaller than for the first method. While the standard deviation from the mean value on the set of all tested scenarios is small and thus, the behaviour of the system from the viewpoint of this metric is more expectable than random, the statistical values are similar for both methods, and hence, the performance is difficult to assess.

The evaluation of both methods confirmed that plan repair could better modify the plan in the way that as many as possible plan steps are preserved from the operator's plan than in replanning from scratch. Thus, the plan repair method is more suitable to persuade companies to trust and use such a system with a minimal amount of AI capable of repairing a plan when a failure in its execution occurs. However, it is necessary to keep in mind that this method does not provide the best results when considering the delay of the entire plan or an average delay of a cargo delivery. Nevertheless, the main criterion is the difference from the original operator's plan because it is required by industrial companies.



Chapter 6

Conclusion

The goal of this thesis was to propose a system that would amend the input plan specified by an operator in reaction to failures during its execution. Nowadays, companies are reluctant to use AI in their manufacturing processes because they are afraid of its behaviour which can comprise unexpected solutions. Therefore, this thesis aims to design a system with a minimal amount of AI responsible for resolving failures that will modify the operator's plan as little as possible so that these changes are straightforward and well-understand by human beings.

While working on this thesis, I needed to familiarise myself with the following topics.

First, I needed to study Vehicle Routing Problems in which a fleet of vehicles should visit a set of places so that some objectives are minimized. A specific group of these problems is called the Pickup and Delivery problem, which focuses specifically on the task of transporting goods. The Pickup and Delivery Problem with Time Windows (PDP-TW) is a group of problems in which goods transportation is scheduled. The task of delivering goods in industrial environments can be the most realistically described as this problem. Therefore, the group of these problems is reviewed.

Second, I reviewed failures that can occur during the execution of schedules addressing PDP-TW in industrial environments such as production and logistics in factories.

Third, I deepened my knowledge about the planning language PDDL that is used to model the industrial environment and describe the task. The planning languages PDDL 2.1 and PDDL 3 are reviewed, and their properties and possibilities are discussed with respect to the PDP-TW. Both versions can be used for modelling. In the proposed system, PDDL 2.1 is used because this version introduces durative actions necessary to model schedules and thus, it is a sufficient version for addressing PDP-TW. Moreover, using version 2.1 has the advantage that plenty of state of the art planners supports this version in contrast to PDDL 3. Hence, we were able to test more planners and pick the most suitable one.

Fourth and the last, planners that can solve problems described in the form of PDDL such as TFD and OPTIC were reviewed. Both planners are reviewed. The TFD planner was then used in the proposed system because it seems to be more suitable for the tackled task than OPTIC, especially due to its computational speed.

The core of the thesis then proposes two approaches that can modify the operator's

plan when a failure during the execution occurs. First, the plan repair method aims to preserve as many plan steps of the operator's plan as possible. To be able to reuse the majority of the operator's plan, it is assumed that the execution finished with the failure appeared. The operator's schedule is then repaired with the use of a plan that the TFD planner provides only for the part of the operator's plan affected by a failure. The second proposed method is replanning from scratch. This method uses a plan provided by the TFD planner since the time when a failure occurred to amend the operator's schedule. The first approach is novel, while the second is a state of the art solution, and it is used in order to evaluate the performance of the plan repair method.

Finally, the proposed systems were evaluated on a set of testing scenarios. These scenarios describe an industrial environment and various time instants in which two frequent types of failures, a failure of a robot and failure due to a blocked path, can occur. The behaviour of both methods is assessed according to three criteria. These criteria were chosen in order to evaluate the system's ability to provide minor changes to the operator's plan when modifying its schedule and properties valuable for industrial operations, such as a delay of the modified plan or an average delay of a cargo delivery. Experimental results confirmed that plan repair provides plans with a small difference from the operator's plan than replanning from scratch. In contrast, plans provided by replanning from scratch can be less delayed when compared to the operator's plan and can have a smaller delay of a cargo delivery on average. These findings can be summarized in two most important conclusions:

- Prefer plan repair over replanning from scratch if the most critical requirement is a small difference from the operator's plan.
- Prefer replanning from scratch over plan repair when interested in performance in industrial operations.

The advantage of the proposed system worth emphasizing is its modularity. Thanks to that property, any state of the art planner that can deal with at least durative actions can be used and thus, the performance of this system can be continuously improved with novel and better-performing planners.

■ 6.1 Future work

Further work can be divided into three directions. The first direction addresses the improvements of the system's properties. The second direction focuses on the analysis of further abilities of the system. The practical part of the working system is targeted in the third direction.

Improvements of the system's properties

During the work on this thesis, several problems were revealed. These problems are addressed in this section, and possible solutions to them are suggested.

6.1 FUTURE WORK

The plan repair method does not perform very well from the viewpoint of the total plan delay and an average cargo delivery delay. These two abilities could be improved by **auctioning algorithms** in which a remaining cargo to be served would be assigned to an AGV who finishes the entire plan including this cargo at the earliest time.

Another drawback of the plan repair method can be seen in performance in industrial operations when investigating the plan difference. In some cases, an AGV is forced to reuse the operator's plan until the last possible action so that the AGV drive back to the dock first and then perform the extra assigned tasks. To overcome this disadvantage, a **merging algorithm** based on the approach similar to [37] could be developed. Such an algorithm could temporarily violate the reusing of the operator's plan in the case when an AGV is driving to the docking centre. This approach should not influence the plan difference metric much because the driving into the dock is satisfied in the end, but it could improve the total plan delay and average cargo delay metrics. This method could be a compromise between plan repair and replanning from scratch by combining both approaches and taking their advantages.

The last property that could be handled is a violation of **time windows** and planning based on preferences. These properties would need to be modelled using PDDL 3, and a planner supporting all those features would need to be available for public use.

Further evaluation

It would be interesting to evaluate the behaviour of the system on a huge problem and also investigate its **scalability** in terms of AGVs, cargos and waypoints. The difference between plan repair and replanning from scratch could be analysed in terms of the **runtime**. Though, to be able to cope with these goals, a broader analysis of the state of the art planners similar to that one done in section 5.2.1.1 would be necessary.

One of the most relevant algorithms to this work are presented in [22]. A **comparison** with similar algorithms could reveal more information about the proposed system.

Execution

The further steps can be to integrate the system into REX and verify its functionality in the real industrial operations as well as in simulated environments such as in Gazebo¹⁵. The repeatability and robustness of plan modifications could be then analysed.

¹⁵http://gazebosim.org/tutorials?tut=ros2_overview

References

- [1] Datavision s.r.o. Project presentation, appendix number 1, 2020. Project Guidance and Localization upgrade creating Autonomous Mobile Robots. Technology Agency of the Czech Republic. TREND Programme FW03010020.
- [2] Jonathan L. Gross and Jay Yellen. *Handbook of graph theory*. CRC press, 2003.
- [3] Pictorial History of the TSP, May 2005. [Online; accessed 17. Apr. 2021]. Available from <http://www.math.uwaterloo.ca/tsp/history/pictorial/pictorial.html>.
- [4] Ministry of Industry and Trade. Industry 4.0 initiative, 2016.
- [5] Vladimír Mařík et al. Industry 4.0 national initiative, 2015. Ministry of Industry and Trade.
- [6] Ángel Corberán and Gilbert Laporte. *Arc routing: problems, methods, and applications*. SIAM, 2015.
- [7] Paolo Toth and Daniele Vigo. *Vehicle routing: problems, methods, and applications*. SIAM, 2014.
- [8] Jan Karel Lenstra and AHG Rinnooy Kan. On general routing problems. *Networks*, 6(3):273–280, 1976.
- [9] Leonhard Euler. The seven bridges of Königsberg. *The world of mathematics*, 1:573–580, 1956.
- [10] Jack Edmonds and Ellis L. Johnson. Matching, euler tours and the chinese postman. *Mathematical programming*, 5(1):88–124, 1973.
- [11] Bernhard H. Korte, Jens Vygen, B. Korte, and J. Vygen. *Combinatorial optimization*, volume 1. Springer, 2011.
- [12] David Guichard. An introduction to combinatorics and graph theory. *Whitman College-Creative Commons*, 2017.
- [13] George Dantzig, Ray Fulkerson, and Selmer Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4):393–410, 1954.

- [14] David L. Applegate, Robert E. Bixby, Vasek Chvatal, and William J. Cook. *The traveling salesman problem: a computational study*. Princeton university press, 2006.
- [15] David L. Applegate, Robert E. Bixby, Vašek Chvátal, William Cook, Daniel G. Espinoza, Marcos Goycoolea, and Keld Helsgaun. Certification of an optimal tsp tour through 85,900 cities. *Operations Research Letters*, 37(1):11–15, 2009.
- [16] George B. Dantzig and John H. Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, 1959.
- [17] Andrea Mor and Maria Grazia Speranza. Vehicle routing problems over time: a survey. *4OR*, pages 1–21, 2020.
- [18] Salma Naccache, Jean-François Côté, and Leandro C. Coelho. The multi-pickup and delivery problem with time windows. *European Journal of Operational Research*, 269(1):353–362, 2018.
- [19] Farshid Azadian, Alper Murat, and Ratna Babu Chinnam. An unpaired pickup and delivery problem with time dependent assignment costs: Application in air cargo transportation. *European Journal of Operational Research*, 263(1):188–202, 2017.
- [20] Sungbum Jun, Seokcheon Lee, and Yuehwern Yih. Pickup and delivery problem with recharging for material handling systems utilising autonomous mobile robots. *European Journal of Operational Research*, 289(3):1153–1168, 2021.
- [21] Ramin Raeesi and Konstantinos G. Zografos. The electric vehicle routing problem with time windows and synchronised mobile battery swapping. *Transportation Research Part B: Methodological*, 140:101–129, 2020.
- [22] Antonín Komenda, Peter Novák, and Michal Pěchouček. Domain-independent multi-agent plan repair. *Journal of Network and Computer Applications*, 37:76–88, 2014.
- [23] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.
- [24] Constructions Aeronautiques, Adele Howe, Craig Knoblock, ISI Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, David Wilkins SRI, Anthony Barrett, Dave Christianson, et al. Pddl| the planning domain definition language. Technical report, Technical Report, 1998.
- [25] Maria Fox and Derek Long. Pddl2. 1: An extension to pddl for expressing temporal planning domains. *Journal of artificial intelligence research*, 20:61–124, 2003.
- [26] Stefan Edelkamp and Jörg Hoffmann. Pddl2. 2: The language for the classical part of the 4th international planning competition. Technical report, Technical Report 195, University of Freiburg, 2004.

- [27] Alfonso Gerevini and Derek Long. Plan constraints and preferences in pddl3. Technical report, Technical Report 2005-08-07, Department of Electronics for Automation, 2005.
- [28] Patrick Eyerich, Robert Mattmüller, and Gabriele Röger. Using the context-enhanced additive heuristic for temporal and numeric planning. In *Nineteenth International Conference on Automated Planning and Scheduling*, 2009.
- [29] J. Benton, Amanda Coles, and Andrew Coles. Temporal planning with preferences and time-dependent continuous costs. In *Twenty-Second International Conference on Automated Planning and Scheduling*, 2012.
- [30] Amanda Coles, Andrew Coles, Maria Fox, and Derek Long. Forward-chaining partial-order planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 20, 2010.
- [31] Subbarao Kambhampati and James A. Hendler. A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence*, 55(2-3):193–258, 1992.
- [32] Roman Van Der Krogt and Mathijs De Weerdt. Plan repair as an extension of planning. In *ICAPS*, volume 5, pages 161–170, 2005.
- [33] Tsz-Chiu Au, Héctor Munoz-Avila, and Dana S. Nau. On the complexity of plan adaptation by derivational analogy in a universal classical planning framework. In *European Conference on Case-Based Reasoning*, pages 13–27. Springer, 2002.
- [34] Maria Fox, Alfonso Gerevini, Derek Long, and Ivan Serina. Plan stability: Re-planning versus plan repair. In *ICAPS*, volume 6, pages 212–221, 2006.
- [35] Bernhard Nebel and Jana Koehler. Plan reuse versus plan generation: A theoretical and empirical analysis. *Artificial intelligence*, 76(1-2):427–454, 1995.
- [36] Subbarao Kambhampati, Craig A. Knoblock, and Qiang Yang. Planning as refinement search: A unified framework for evaluating design tradeoffs in partial-order planning. *Artificial Intelligence*, 76(1-2):167–238, 1995.
- [37] Lenka Mudrová, Bruno Lacerda, and Nick Hawes. Partial order temporal plan merging for mobile robot tasks. *ECAI*, 2016.
- [38] Nerea Luis, Susana Fernández, and Daniel Borrajo. Plan merging by reuse for multi-agent planning. *Applied Intelligence*, 50(2):365–396, 2020.
- [39] Jaume Jordán, Alejandro Torreno, Mathijs De Weerdt, and Eva Onaindia. A better-response strategy for self-interested planning agents. *Applied Intelligence*, 48(4):1020–1040, 2018.
- [40] Catherine Harris and Richard Dearden. Contingency planning for long-duration auv missions. In *2012 IEEE/OES Autonomous Underwater Vehicles (AUV)*, pages 1–6. IEEE, 2012.

Appendix A

Attached code

The domain and problem files are attached in the following sections in this appendix.

A.1 PDDL domain

The full code demonstrating the PDDL domain is in Listing 5.

Listing 5: PDDL domain

```
1 (define (domain agvtransportsimplefunctions)
2   (:requirements :typing :durative-actions :fluents)
3   (:types
4     waypoint locatable - object
5     agv cargo - locatable
6   )
7
8   (:predicates
9     (at ?obj - locatable ?wp - waypoint)
10    (in ?cargo - cargo ?agv - agv)
11    (path ?from - waypoint ?to - waypoint)
12    (empty ?agv - agv)
13    (full ?agv - agv)
14    (alive ?agv - agv)
15  )
16
17  (:functions
18    (travel_time ?wp1 - waypoint ?wp2 - waypoint)
19  )
20
21  (:durative-action load
22    :parameters (?agv - agv ?cargo - cargo ?wp - waypoint)
23    :duration (= ?duration 2)
24    :condition (and
25      (over all (alive ?agv))
26      (over all (at ?agv ?wp))
27      (at start (at ?cargo ?wp))
28      (at start (empty ?agv))
29    )
30    :effect (and
31      (at start (not (at ?cargo ?wp)))
```

```

32     (at end (in ?cargo ?agv))
33     (at start (not (empty ?agv)))
34     (at end (full ?agv))
35   )
36 )
37
38 (:durative-action unload
39   :parameters (?agv - agv ?cargo - cargo ?wp - waypoint)
40   :duration (= ?duration 2)
41   :condition (and
42     (over all (alive ?agv))
43     (over all (at ?agv ?wp))
44     (at start (in ?cargo ?agv))
45     (at start (full ?agv))
46   )
47   :effect (and
48     (at start (not (in ?cargo ?agv)))
49     (at end (at ?cargo ?wp))
50     (at start (not (full ?agv)))
51     (at end (empty ?agv))
52   )
53 )
54
55 (:durative-action drive
56   :parameters (?agv - agv ?from - waypoint ?to - waypoint)
57   :duration (= ?duration (travel_time ?from ?to))
58   :condition (and
59     (over all (alive ?agv))
60     (at start (at ?agv ?from))
61     (over all (path ?from ?to))
62   )
63   :effect (and
64     (at start (not (at ?agv ?from)))
65     (at end (at ?agv ?to))
66   )
67 )
68
69 )

```

■ A.2 Initial PDDL problem

See Listing 6 for the whole PDDL problem file.

Listing 6: Initial PDDL problem

```

1 (define (problem prob_wp8_c6_a3)
2   (:domain agvtransportsimplefunctions)
3
4   (:objects
5     agv0 - agv
6     agv1 - agv

```

A.2 INITIAL PDDL PROBLEM

```
7         agv2 – agv
8         cargo0 – cargo
9         cargo1 – cargo
10        cargo2 – cargo
11        cargo3 – cargo
12        cargo4 – cargo
13        cargo5 – cargo
14        wp0 – waypoint
15        wp1 – waypoint
16        wp2 – waypoint
17        wp3 – waypoint
18        wp4 – waypoint
19        wp5 – waypoint
20        wp6 – waypoint
21        wp7 – waypoint
22        wp8 – waypoint
23    )
24
25    (:init
26        (at agv0 wp1)
27        (empty agv0)
28        (alive agv0)
29        (at agv1 wp1)
30        (empty agv1)
31        (alive agv1)
32        (at agv2 wp1)
33        (empty agv2)
34        (alive agv2)
35        (at cargo0 wp0)
36        (at cargo1 wp0)
37        (at cargo2 wp0)
38        (at cargo3 wp0)
39        (at cargo4 wp0)
40        (at cargo5 wp0)
41        (path wp0 wp1)
42        (path wp0 wp2)
43        (path wp0 wp6)
44        (path wp0 wp8)
45        (path wp1 wp0)
46        (path wp1 wp2)
47        (path wp1 wp3)
48        (path wp1 wp7)
49        (path wp1 wp8)
50        (path wp2 wp0)
51        (path wp2 wp1)
52        (path wp2 wp3)
53        (path wp2 wp4)
54        (path wp3 wp1)
55        (path wp3 wp2)
56        (path wp3 wp5)
57        (path wp4 wp2)
58        (path wp4 wp6)
59        (path wp5 wp3)
```

```

60         (path wp5 wp7)
61         (path wp6 wp0)
62         (path wp6 wp4)
63         (path wp6 wp7)
64         (path wp7 wp1)
65         (path wp7 wp5)
66         (path wp7 wp6)
67         (path wp7 wp8)
68         (path wp8 wp0)
69         (path wp8 wp1)
70         (path wp8 wp7)
71
72         (= (travel_time wp0 wp1) 4)
73         (= (travel_time wp0 wp2) 5)
74         (= (travel_time wp0 wp6) 14)
75         (= (travel_time wp0 wp8) 7)
76         (= (travel_time wp1 wp0) 4)
77         (= (travel_time wp1 wp3) 3)
78         (= (travel_time wp1 wp8) 2)
79         (= (travel_time wp2 wp0) 5)
80         (= (travel_time wp2 wp3) 2)
81         (= (travel_time wp2 wp4) 2)
82         (= (travel_time wp3 wp1) 3)
83         (= (travel_time wp3 wp2) 2)
84         (= (travel_time wp3 wp5) 2)
85         (= (travel_time wp4 wp2) 2)
86         (= (travel_time wp4 wp6) 2)
87         (= (travel_time wp5 wp3) 2)
88         (= (travel_time wp5 wp7) 2)
89         (= (travel_time wp6 wp4) 2)
90         (= (travel_time wp6 wp7) 2)
91         (= (travel_time wp6 wp0) 14)
92         (= (travel_time wp7 wp5) 2)
93         (= (travel_time wp7 wp6) 2)
94         (= (travel_time wp7 wp8) 11)
95         (= (travel_time wp8 wp0) 7)
96         (= (travel_time wp8 wp1) 2)
97         (= (travel_time wp8 wp7) 11)
98     )
99
100    (:goal
101      (and
102        (at agv0 wp1)
103        (at agv1 wp1)
104        (at agv2 wp1)
105        (at cargo0 wp2)
106        (at cargo1 wp3)
107        (at cargo2 wp4)
108        (at cargo3 wp5)
109        (at cargo4 wp6)
110        (at cargo5 wp7)
111      )
112    )

```

A.3 INITIAL PDDL PLAN

```
113
114     (:metric minimize
115      (total-time)
116     )
117
118 )
```

A.3 Initial PDDL plan

The initial PDDL plan is provided in Listing 7.

Listing 7: Initial PDDL plan

```
1 0.00100000: (drive agv0 wp1 wp0) [4.00000000]
2 4.01100000: (load agv0 cargo0 wp0) [2.00000000]
3 6.02200000: (drive agv0 wp0 wp2) [5.00000000]
4 11.03300000: (unload agv0 cargo0 wp2) [2.00000000]
5 0.00100000: (drive agv2 wp1 wp0) [4.00000000]
6 4.01200000: (load agv2 cargo1 wp0) [2.00000000]
7 6.02300000: (drive agv2 wp0 wp1) [4.00000000]
8 10.03300000: (drive agv2 wp1 wp3) [3.00000000]
9 13.04400000: (unload agv2 cargo1 wp3) [2.00000000]
10 0.00100000: (drive agv1 wp1 wp0) [4.00000000]
11 4.01200000: (load agv1 cargo2 wp0) [2.00000000]
12 6.02300000: (drive agv1 wp0 wp2) [5.00000000]
13 11.03400000: (drive agv1 wp2 wp4) [2.00000000]
14 13.04500000: (unload agv1 cargo2 wp4) [2.00000000]
15 13.04500000: (drive agv0 wp2 wp0) [5.00000000]
16 18.05700000: (load agv0 cargo3 wp0) [2.00000000]
17 20.06700000: (drive agv0 wp0 wp1) [4.00000000]
18 24.07800000: (drive agv0 wp1 wp3) [3.00000000]
19 27.08900000: (drive agv0 wp3 wp5) [2.00000000]
20 29.10000000: (unload agv0 cargo3 wp5) [2.00000000]
21 15.05500000: (drive agv2 wp3 wp2) [2.00000000]
22 17.06600000: (drive agv2 wp2 wp0) [5.00000000]
23 22.07700000: (load agv2 cargo4 wp0) [2.00000000]
24 24.08800000: (drive agv2 wp0 wp2) [5.00000000]
25 29.09900000: (drive agv2 wp2 wp4) [2.00000000]
26 31.11000000: (drive agv2 wp4 wp6) [2.00000000]
27 33.12100000: (unload agv2 cargo4 wp6) [2.00000000]
28 15.05600000: (drive agv1 wp4 wp2) [2.00000000]
29 17.06700000: (drive agv1 wp2 wp0) [5.00000000]
30 22.07800000: (load agv1 cargo5 wp0) [2.00000000]
31 24.08900000: (drive agv1 wp0 wp2) [5.00000000]
32 29.10000000: (drive agv1 wp2 wp3) [2.00000000]
33 31.11100000: (drive agv1 wp3 wp5) [2.00000000]
34 33.12200000: (drive agv1 wp5 wp7) [2.00000000]
35 35.13200000: (unload agv1 cargo5 wp7) [2.00000000]
36 31.11100000: (drive agv0 wp5 wp3) [2.00000000]
37 33.12200000: (drive agv0 wp3 wp1) [3.00000000]
38 37.14200000: (drive agv1 wp7 wp5) [2.00000000]
```

```
39 39.15300000: (drive agv1 wp5 wp3) [2.00000000]
40 41.16400000: (drive agv1 wp3 wp1) [3.00000000]
41 35.13200000: (drive agv2 wp6 wp4) [2.00000000]
42 37.14300000: (drive agv2 wp4 wp2) [2.00000000]
43 39.15400000: (drive agv2 wp2 wp3) [2.00000000]
44 41.16500000: (drive agv2 wp3 wp1) [3.00000000]
```



Appendix B

Graphs

Graphs depicting results for 42 scenarios are attached in this appendix. Figure 15 shows the *plan difference*, Figure 16 presents the *total plan delay*, and an *average cargo delivery delay* is in Figure 17.

A.3 INITIAL PDDL PLAN

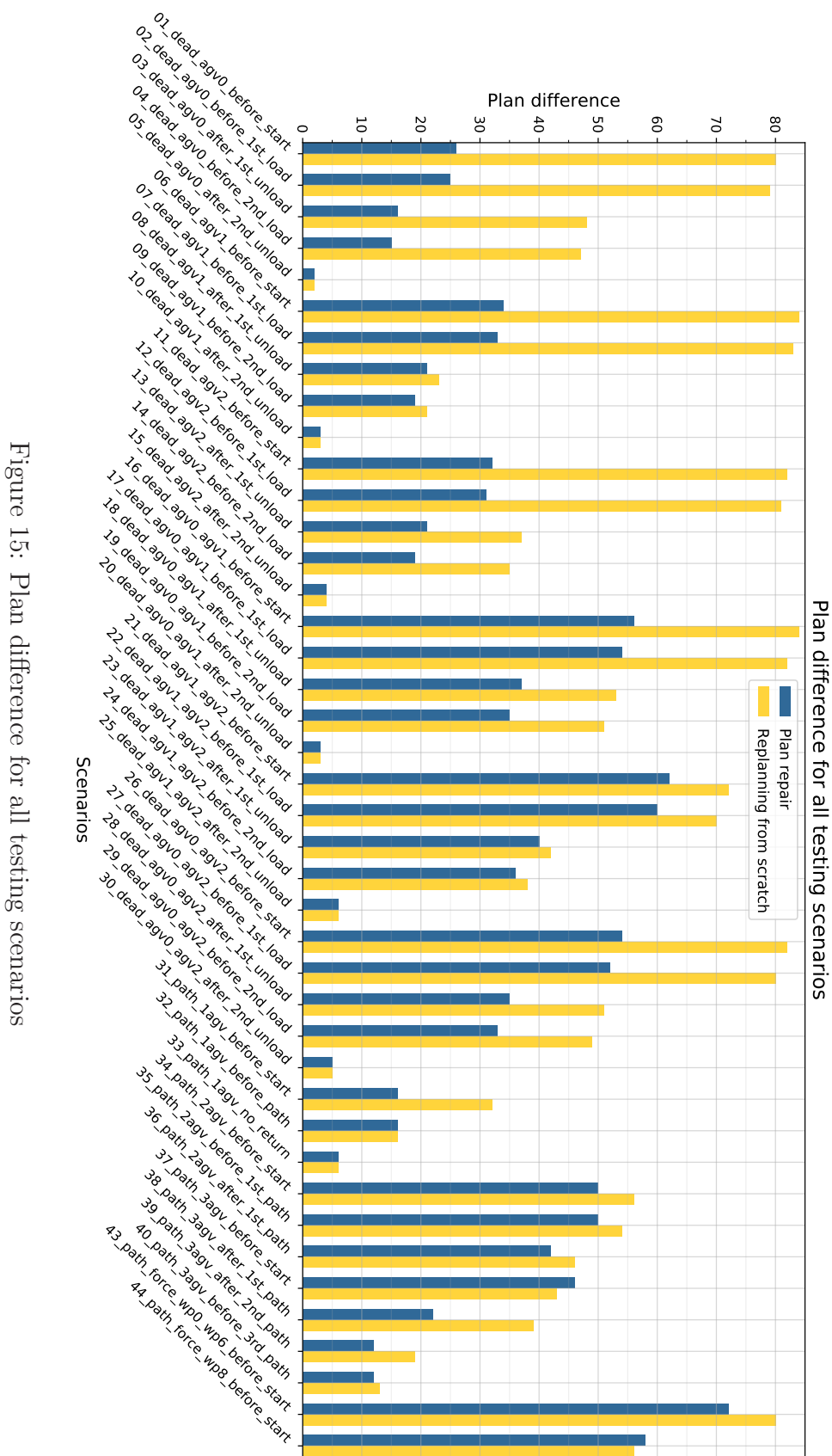


Figure 15: Plan difference for all testing scenarios

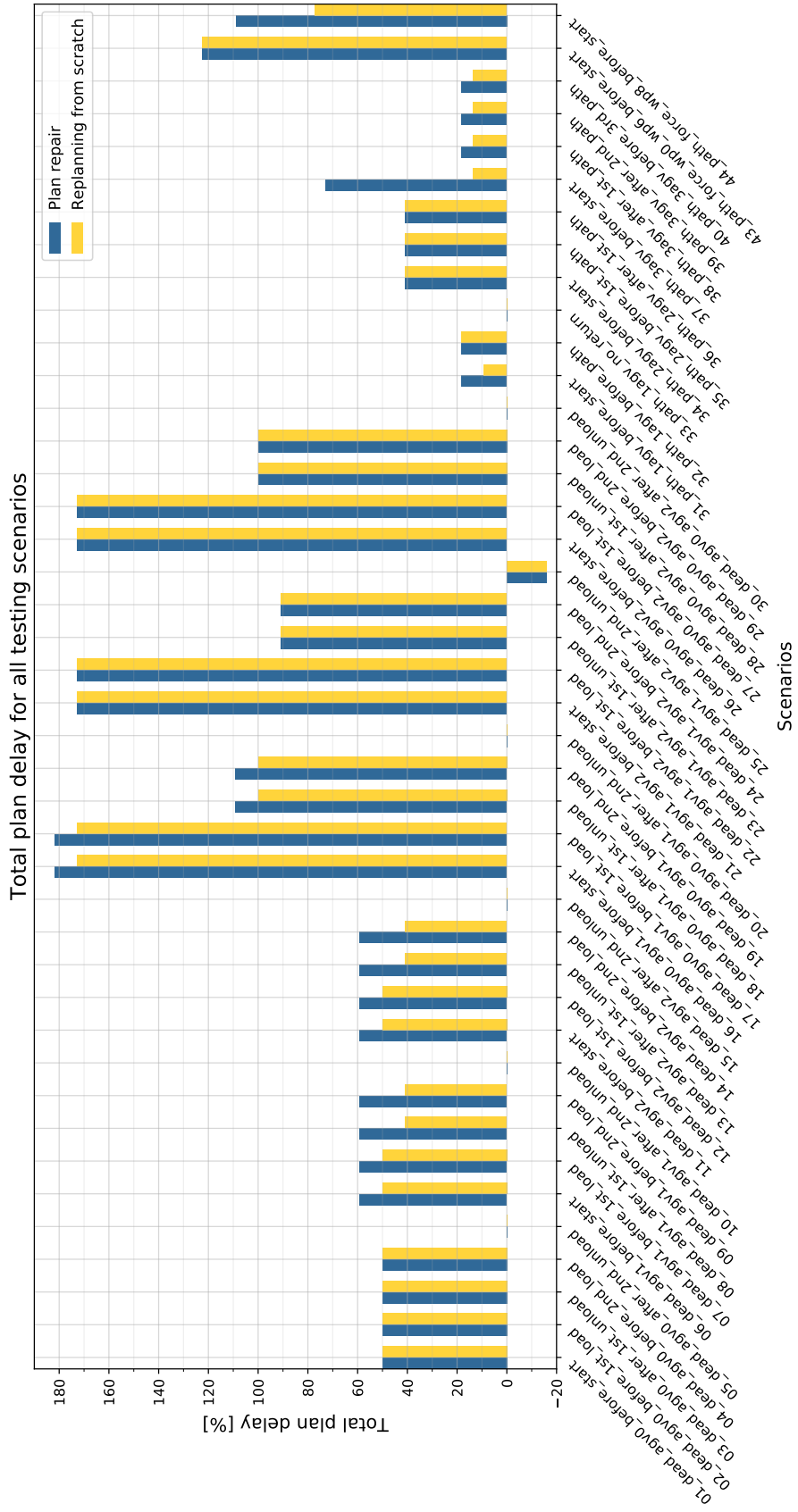


Figure 16: Total plan delay for all testing scenarios

A.3 INITIAL PDDL PLAN

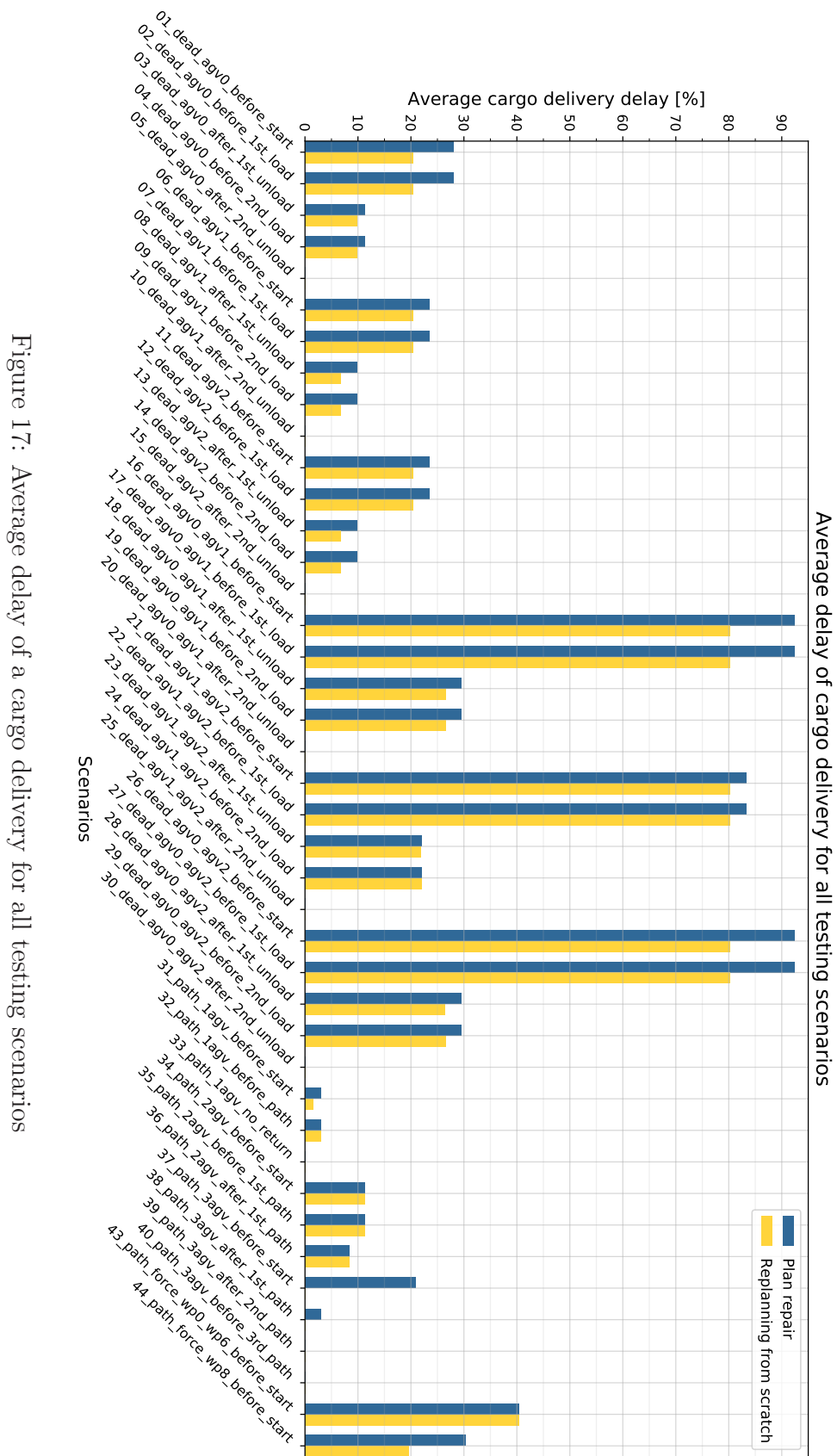


Figure 17: Average delay of a cargo delivery for all testing scenarios