

Master Thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

Multi-agent Systems for Production Planning

Tomáš Staruch

Supervisor: Ing. Vojtěch Janů
January 2023

I. Personal and study details

Student's name: **Staruch Tomáš** Personal ID number: **465829**
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:

Multi-agent Systems for Production Planning

Master's thesis title in Czech:

Multiagentní systémy pro plánování výroby

Guidelines:

The goal of this thesis is to get acquainted with the theory of multi-agent systems and to design, implement, and experimentally verify an algorithm for multiagent planning of production. The following steps should be followed:

1. Get acquainted with the theory of multi-agent systems and write an overview of concepts that are important for the task of production planning.
2. Get acquainted with the existing non-complete solution for production planning that is used in Testbed for Industry 4.0.
3. Propose sound and complete algorithm for production planning of a single product.
4. Implement and validate the proposed algorithm on the Testbed production line by executing a complete production of the product.

Bibliography / sources:

- [1] Shoham, Y., & Leyton-Brown, K. (2008). Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations. Cambridge University Press. doi:10.1017/CBO9780511811654
[2] FIPA Communicative Act Library Specification. (2002). Retrieved 9 January 2022, from <http://www.fipa.org/specs/fipa00037/SC00037J.html>

Name and workplace of master's thesis supervisor:

Ing. Vojtěch Jan Testbed - CIIRK

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

Date of master's thesis assignment: **24.01.2022** Deadline for master's thesis submission: **10.01.2023**

Assignment valid until:

by the end of winter semester 2023/2024

Ing. Vojtěch Jan
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 his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt

Student's signature

Acknowledgements

I want to thank Ing. Vojtěch Janů for his outstanding support during this project. I would also like to thank the other members of the Testbed for Industry 4.0 group for their valuable advice, and I would like to thank Ing. Pavel Burget for the opportunity to work on this project at Testbed.

Declaration

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

Prague, 10. January 2023

Abstract

The aim of this thesis is to design, implement and experimentally verify an algorithm for multiagent production scheduling. The proposed algorithm uses the principle of chronological backtracking and is an extension of the existing plan, commit, and execute algorithm, which sometimes fails to find a solution. The implementation of the proposed algorithm is done in Java and uses the currently under-development multiagent platform for agent-to-agent communication, database, and logging. The functionality of the proposed algorithm is verified and compared with the existing algorithm in a simulation environment that forms a virtual twin of the Montrac testbed production line. The results are evaluated based on data from six test scenarios.

Keywords: production planning, multiagent system

Supervisor: Ing. Vojtěch Janů

Abstrakt

Cílem této práce je navrhnout, implementovat a experimentálně ověřit algoritmus pro multiagentní plánování výroby. Navržený algoritmus využívá principu chronologického backtrackingu a je rozšířením existujícího algoritmu plan, commit, execute, který v některých případech nedokáže najít řešení. Implementace navrženého algoritmu je provedena v Javě a využívá aktuálně vyvíjenou multiagentní platformu pro komunikaci mezi agenty, databázi a logování. Funkčnost navrženého algoritmu je ověřena a porovnána s existujícím algoritmem v simulačním prostředí, které tvoří virtuální dvojče testovací linky Montrac. Výsledky jsou vyhodnoceny na základě dat ze šesti příkladů.

Klíčová slova: plánování výroby, multiagentní systém

Překlad názvu: Multiagentní systémy pro plánování výroby

Contents

1 Introduction	1	1.6 Game Theory	12
1.1 State of the art	1	1.6.1 Noncooperative game theory	12
1.1.1 Multiagent systems area of use	1	1.6.2 Coalitional game theory	16
1.1.2 Multiagent systems in production planning	3	1.6.3 Multiagent resource allocation and Auctions	16
1.2 Multiagent systems history	3	2 Problem Description	19
1.3 Description of multiagent systems	4	2.1 Description of the Montrac test line	19
1.3.1 Agent description	4	2.2 Thesis problem description	22
1.3.2 Description of the environment surrounding the agent	5	2.3 MES = Description of actual solution concept	24
1.3.3 Intelligent agent description . .	6	2.4 MAS = Description of the developed version of the solution that uses multi-agent systems	25
1.4 Communication between agents . .	7	2.4.1 Description of agents	25
1.4.1 Speech act theory	7	2.4.2 Description of testing solution for production scheduling	25
1.4.2 Multiagent communication languages	8	2.4.3 Description of the proposal price calculation	26
1.5 Modeling techniques	11	2.4.4 Problems of current solution .	28
1.5.1 Distributed Constraint Satisfaction	11	3 Design of a new algorithm	29
1.5.2 Distributed optimization	11		

3.1 Design of communication automata	29	5.1.3 Production flow of the second product	49
3.2 Finding a suitable algorithm that is complete.....	31	5.1.4 Production flow of the third product	51
3.2.1 Parallel scheduling	31	5.1.5 Production flow of the fourth product	52
3.2.2 Schedulers	32	5.2 Use case 2	52
3.2.3 Chronological backtracking ..	33	5.2.1 Description of the use case ..	52
4 Implementation of the new algorithm	37	5.2.2 Production flow of the first product	53
4.1 Agents implementation without transport	38	5.2.3 Production flow of the second product	54
4.2 Agents implementation with product transport	38	5.3 Use case 3	55
4.3 Agents implementation with product and parts transport	41	5.3.1 Description of the use case ..	55
4.3.1 Agents implementation with backtracking	44	5.3.2 Production flow of the first product	57
5 Testing and comparison of algorithms	47	5.3.3 Production flow of the second product	58
5.1 Use case 1	47	5.3.4 Production flow of the third product	60
5.1.1 Description of the use case ..	47	5.4 Use case 4	61
5.1.2 Production flow of the first product	48	5.4.1 Description of the use case ..	61

5.4.2 Production flow of the first product	62
5.5 Use case 5	63
5.5.1 Description of the use case ..	63
5.5.2 Production flow of the first product	65
5.6 Use case 6	66
5.6.1 Description of the use case ..	66
5.6.2 Production flow for five product	67
5.7 Discussion	69
6 Conclusion	71
A Bibliography	73
B Additional tables	77

Figures

1.1 Behaviour of an agent in its environment, [1]	5
1.2 Example of simple communication exchange between two rational agents	9
1.3 Example of communication state machine between two rational agents, CFP - call for proposal, NU - not understood, P - propose, AP - accept proposal, RP - reject proposal, C - Confirm, D - disconfirm	10
1.4 The Sharing game described by extensive-form game representation	15
1.5 The Sharing game with imperfect information described by extensive-form game representation	15
2.1 Diagram of Montrac safety zones	22
2.2 Architecture of MES	24
2.3 Description of the price calculation for the operation, [2]	27
2.4 A production line with four robots, where each robot can perform an operation described by a number. Between the robots is a unidirectional conveyor belt, which is shown by arrows	28
3.1 state communication automaton for Request type messages	30
3.2 state communication automaton (plan-commit) for production operation or material import	31
3.3 Illustration of the algorithm with backtracking on an example with four robots	34
3.4 The transport links between the agents on the line - pruning example	35
3.5 DAG between strongly continuous components	35
3.6 Simplifying the example state space by pruning	36
4.1 Transport routes in system	38
4.2 Diagram describing computing of price for an operation	40
4.3 Diagram describing commit for an operation	40
4.4 Diagram describing computing of price for a parts transport	41
4.5 Diagram describing commit for a parts transport	42
4.6 Diagram describing computing of price with product and parts transport	43
4.7 Diagram describing backtracking algorithm	45

4.8 Diagram describing transport connection between agents in backtracking example	45	5.9 Diagram describing second flow through the Use case 3 for plan, commit, execute algorithm. Costs are in seconds.	58
4.9 Diagram describing flow through the backtracking algorithm on backtracking example. Cost are in ms.	46	5.10 Diagram describing second flow through the Use case 3 for backtracking algorithm. Costs are in seconds.	59
5.1 Diagram describing the transport between agents for Use case 1. Costs of transitions are in <i>s</i>	48	5.11 Diagram describing third flow through the Use case 3 for backtracking algorithm. Costs are in seconds.	60
5.2 Diagram describing first flow through the Use case 1 for both algorithms. Cost are in seconds. . .	49	5.12 Diagram describing the transport between agents for Use case 4. Costs of transitions are in <i>s</i>	61
5.3 Diagram describing second flow through the Use case 1 for both algorithms. Costs are in seconds. .	50	5.13 Diagram describing first flow through the Use case 4 for backtracking algorithm. Costs are in seconds.	62
5.4 Diagram describing third flow through the Use case 1 for both algorithms. Costs are in seconds. .	51	5.14 Diagram describing the transport between agents for Use case 5. Costs of transitions are in <i>s</i>	64
5.5 Diagram describing first flow through the Use case 2 for both algorithms. Costs are in seconds. .	53	5.15 Diagram describing first flow through the Use case 5 for Plan, commit, execute algorithm. Costs are in seconds.	65
5.6 Diagram describing second flow through the Use case 2 for both algorithms. Costs are in seconds. .	54	5.16 Diagram describing first flow through the Use case 5 for Backtracking algorithm. Costs are in seconds.	65
5.7 Diagram describing the transport between agents for Use case 3. Costs of transitions are in <i>s</i>	56	5.17 Diagram describing the transport between agents for Use case 6. Costs of transitions are in <i>s</i>	66
5.8 Diagram describing first flow through the Use case 3 for both algorithms. Costs are in seconds. .	57		

Tables

1.1 game of Rock Paper Scissors described using Normal Form Representation. The rows describe the actions of the first player and the columns describe the actions of the second player. The first value in parentheses always describes the first player's utility	13
1.2 Battle of sexes game described using Normal Form Representation	14
2.1 Diagram of Montrac production line	20
4.1 Price of paths between nodes in Figure 4.1	39
4.2 Agent capabilities with their cost and number of components for each agent	46
5.1 Agent capabilities with their cost [s] and number of parts needed for each operation in Use case 1.	48
5.2 The initial state of the agents' magazines in Use case 1.	48
5.3 Table showing the collected data from the first algorithm run in Use case 1. The bottom part of the table lists the number of messages received by each agent. P = producer agent, T = transport agent, D = Database	49
5.4 Table showing the collected data from the second algorithm run in Use case 1. The bottom part of the table lists the number of messages received by each agent. P = producer agent, T = transport agent, D = Database	51
5.5 Table showing the collected data from the third algorithm run in Use case 1. The bottom part of the table lists the number of messages received by each agent. P = producer agent, T = transport agent, D = Database	52
5.6 Agent capabilities with their cost [ms] and number of parts needed for each operation for Use case 2	53
5.7 The initial state of the agents' magazines in Use case 2.	53
5.8 Table showing the collected data from the first algorithm run in Use case 2. The bottom part of the table lists the number of messages received by each agent. P = producer agent, T = transport agent, D = Database	54
5.9 Table showing the collected data from the second algorithm run in Use case 2. The bottom part of the table lists the number of messages received by each agent. P = producer agent, T = transport agent, D = Database	55
5.10 Agent capabilities with their cost [s] and number of parts needed for each operation for Use case 3.	56
5.11 The initial state of the agents' magazines in Use case 3.	56

<p>5.12 Table showing the collected data from the first algorithm run in Use case 3. The bottom part of the table lists the number of messages received by each agent. P = producer agent, T = transport agent, D = Database 57</p> <p>5.13 Table showing the collected data from the second backtrack algorithm run in Use case 3. The bottom part of the table lists the number of messages received by each agent. P = producer agent, T = transport agent, D = Database 59</p> <p>5.14 Table showing the collected data from the third backtrack algorithm run in Use case 3. The bottom part of the table lists the number of messages received by each agent. P = producer agent, T = transport agent, D = Database 60</p> <p>5.15 Agent capabilities with their cost [s] and number of parts needed for each operation for Use case 4. 61</p> <p>5.16 The initial state of the agents' magazines in Use case 4. 62</p> <p>5.17 Table showing the collected data from the first backtrack algorithm run in Use case 4. The bottom part of the table lists the number of messages received by each agent. P = producer agent, T = transport agent, D = Database 63</p> <p>5.18 Agent capabilities with their cost [s] and number of parts needed for each operation for Use case 5. 64</p>	<p>5.19 The initial state of the agents' magazines in Use case 5. 64</p> <p>5.20 Agent capabilities with their cost [s] and number of parts needed for each operation for Use case 6. 67</p> <p>5.21 The initial state of the agents' magazines in Use case 6. 67</p> <p>5.22 The table describes the individual product production flows for use case 6. 68</p> <p>B.1 The table shows the list of ISO standards that were used to create the safety on the Montrac line 77</p> <p>B.2 Table showing the messages received by each agent of the first backtracking algorithm run in use case 6. The number of messages is the same for both algorithms. P = producer agent, T = transport agent, D = Database 78</p> <p>B.3 Table showing the messages received by each agent of the second backtracking algorithm run in use case 6. The number of messages is the same for both algorithms. P = producer agent, T = transport agent, D = Database 78</p> <p>B.4 Table showing the messages received by each agent of the third backtracking algorithm run in use case 6. The number of messages is the same for both algorithms. P = producer agent, T = transport agent, D = Database 78</p>
---	--

B.5 Table showing the messages received by each agent of the fourth backtracking algorithm run in use case 6. The number of messages is the same for both algorithms. P = producer agent, T = transport agent, D = Database 79

B.6 Table showing the messages received by each agent of the fifth backtracking algorithm run in use case 6. The number of messages is the same for both algorithms. P = producer agent, T = transport agent, D = Database 79




Chapter 1

Introduction

This work aims to design and implement an algorithm that uses a multiagent platform to remove current agent-based solutions' shortcomings. Chapter one provides a general description of agent systems, their current uses, agent description, communication between agents, game theory, and methods for modeling, reasoning, and creating multiagent systems. Chapter two describes the Montrac line, a description of the given problem, a description of the central planning solution, and a description of the agent solution. Chapter three describes a comparison of approaches and communication design for the new algorithm. Chapter four describes the implementation of the selected algorithm. Chapter five describes the use cases and the results of the algorithms based on the use cases.



1.1 State of the art



1.1.1 Multiagent systems area of use

Multiagent systems (MAS) have gained much attention from researchers in various fields due to their ability to solve complex problems that can be broken down into smaller tasks. A subtask is assigned to an autonomous agent, which a distributed algorithm can solve. Although MAS have wide applicability, there are still several problems such as agent coordination, task assignment, and security. MAS are used in many different areas. The areas

in which they are most commonly used are described below [3], [4].

MAS are mostly used in computer networks. The complexity of these networks continues to increase due to new technologies and the increasing number of devices connected to the network. Agents are used to overcoming this complexity. Given the wide range of applications of MAS in networks, we further divide them into four subcategories:

Cloud computing

Cloud computing provides access to configurable system resources and computing services. Cloud computing often uses virtualization, meaning that a single physical machine is shared among multiple customers as multiple virtual machines. The complexity of cloud computing is primarily due to the management of cloud resources, communication, and accounting of resources and services used by individual users. These problems can be solved by using MAS [5], [6]. Other uses of MAS in cloud computing include resource monitoring, security, and resource discovery.

Social networking

The complexity of these networks lies in their dynamic nature. It means that many users join or leave the network in a short time. MAS can be a potential solution to overcome this complexity. Gatti [7] proposed an agent-based model to predict user behavior. The agents collect a dataset of user behavior with which they create a profile. This profile is then used to predict the future behavior of the user.

Security

The main use of MAS in network security is in the form of an agent-based Intrusion Detection System (IDS). IDS was proposed in [8]. The proposed IDS consists of five agents that work together to find suspicious behavior in the network and take appropriate mitigation actions when such behavior is detected.

Routing

Routing is finding a path for packets between source and destination with certain metrics. Using agents for routing was one of the first applications of MAS. Agent-based routing inspired by ants behavior was described in [9].

Another area where MAS are commonly used is mobile robotics. Here, agents are used to cooperate and coordinate the robots and to plan their trajectories. For example, in this paper [10] they have used MAS for collision avoidance for mobile robots. In recent years, MAS have also been used in large

cities for transport systems and traffic management or for freight distribution planning. Another use is in building management where, for example, agents manage heating.

The last thing MAS are often used for is modelling complex systems. Agents are used here for their flexibility and autonomy, which greatly facilitates the modeling of such systems. The main advantages of using agents, according to D. Helbing [11]: 1) the possibility of aggregation and combination with other modelling methods, 2) flexibility in assumptions for MAS modelling, 3) flexibility in predefined knowledge, as agents can acquire knowledge by learning from the environment, 4) the possibility to explore emergent behaviour due to the proactivity of agents.

1.1.2 Multiagent systems in production planning

Frequent changes in customer needs and requirements require reconfigurable and adaptive production systems. In the last decade, extensive research has developed on deploying multi-agent systems in several industrial environments.

Using MAS for production planning is similar to modeling a complex system. An example of such a system is the newspaper industry, where production planning using multiagent systems has been applied in Germany. This planning had better results than static centralized optimization.[12]

Currently, the use of MAS in production planning is common in supply chain management. One example is the use of agents to allocate orders efficiently according to production line capabilities [13]. Another example is the use of MAS for the collaborative forecasting and replenishment planning process between trading partners [14]. A more detailed overview of the use of MAS for supply chain management has been written in this paper[15].

1.2 Multiagent systems history

The evolution of multiagent systems in their field started simultaneously with the distributed computational environment approximately in 1980. At that time, distributed computing began to be used in LANs for expert systems, systems that are supposed to provide advice or recommend solutions to a

given situation. The multiagent systems got widespread recognition in the mid of the 1990s, when local networks were connected to the Internet. The rapid growth of multi-agent systems began when massive open distributed systems became more common. The growth is due to the belief that agents are a proper approach to exploiting these systems. A multiagent system is a natural tool for understanding and designing an artificial social system, with each agent representing an object in a larger system. [16]

■ 1.3 Description of multiagent systems

■ 1.3.1 Agent description

Multiagent systems combine multiple computational entities, known as agents. Various authors have proposed different definitions of agents. The following definition is taken from [1, p. 4]: "An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to achieve its delegated objectives." An agent is autonomous if it can decide actions for itself without the intervention of other systems or humans. An example of such behavior is depicted in Figure 1.1, an agent takes sensory input from the environment and, based on the input, produces an action that affects the environment. Usually, an agent has a set of actions that could have some preconditions. The interaction with the environment is usually an ongoing, non-terminating one. An agent usually does not have complete control over its environment. It means that an action can fail.

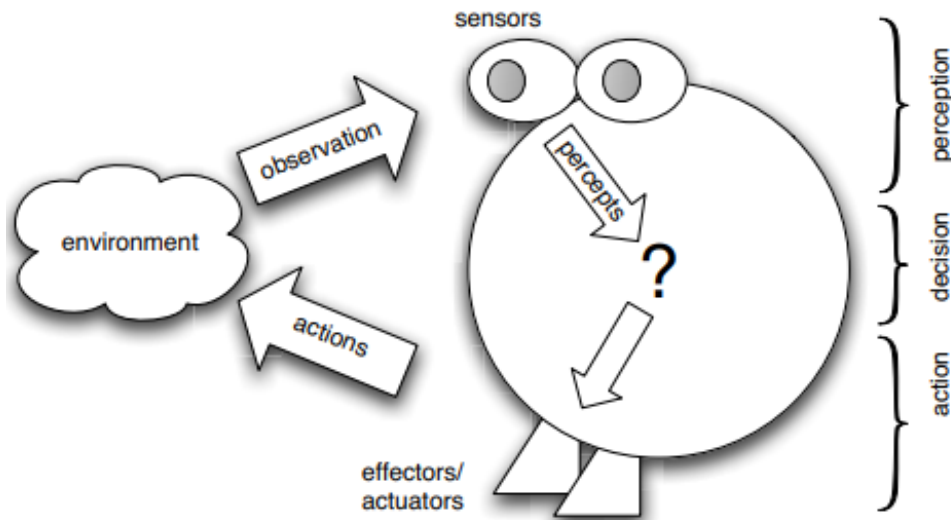


Figure 1.1: Behaviour of an agent in its environment, [1]

1.3.2 Description of the environment surrounding the agent

By definition Agent description, an agent is situated in some environment. Russell and Norvig, in their book [17, p. 40], classified environment properties by five attributes. These attributes are described below.

Fully observable vs. partially observable

In a fully observable environment, an agent has access to complete information about his surroundings at any given time. An environment can be partially observable because a sensors are inaccurate, or some parts of the environment state are not observed or cannot be observable at all. For example, in city traffic, one car cannot predict the actions of other cars with certainty. Most real-world environments are partially observable. The more observable environment is, the easier it is to define a capable agent.

Deterministic vs. stochastic

An environment is deterministic if any action has one guaranteed effect. There is no uncertainty about the state that will result from performing an action. The non-deterministic environment is stochastic. It is more challenging to define an agent in a stochastic environment. If the environment is partially observable, it may appear stochastic to the agent; this is especially true if the environment is complex. Therefore, it is better to consider this property of the environment from the agent's perspective.

Episodic vs. sequential

In an episodic environment, the agent is working in discrete episodes. The episodes are independent of each other, which means that agent actions

in the episode depend only on the episode itself. Many classification tasks are episodic, for example, product classification on the production line or detection of defective parts on an assembly line. In some cases agent decision can affect future decisions in a sequential environment. Typical representatives are games like chess or real-life environments like city traffic. Episodic environments are easier for agent development because the agent makes decisions based only on the current episode.

Static vs dynamic

A static environment does not change over time. The static environment is simpler because an agent does not need to worry about time passing and the change in his surroundings. In a dynamic environment, other processes continuously change the environment. An agent in this environment is periodically asked what it is going to do; if it hasn't decided yet, it counts as doing nothing. An example of a dynamic environment is again city traffic. If the environment is not changing with time, but the agent utility function (a function that describes agent performance) is, then the environment is called semidynamic. An example of this environment could be chess with a clock.

Discrete vs continuous

An environment is discrete if there is a finite number of percepts and actions of the agent. An example of a discrete environment is chess. An example of a continuous environment is city traffic. The speed and position of cars are continuous values.

■ 1.3.3 Intelligent agent description

By the agent definition mentioned above Agent description, an agent can be any simple control system, and still, all aspects of the definition will be valid. An example of such a control system often used in the literature is a thermostat. A temperature sensor is situated in a room, and if the output from the sensor is below the set threshold, then the heating of the room is triggered. From our perspective, we do not view these simple systems as agents, so we add more concepts to the agent definition to define "intelligent agent". The following list of concepts was suggested in [18].

Reactivity

Reactive intelligent agents can perceive their environment and react appropriately in a timely fashion to changes that occur in it to satisfy their delegated objectives.

Proactivity

Proactive intelligent agents can exhibit goal-directed behavior by taking the initiative to satisfy their design objectives.

Social ability

Social able intelligent agents are capable of interacting with other agents and possibly humans to satisfy their design objectives.

Furthermore, in this thesis, we assume that "intelligent agents" are rational [19]. A rational agent behaves in such a way that he tries to maximize the expected utility value given the information at his disposal.

1.4 Communication between agents

1.4.1 Speech act theory

Interaction in the social ability concept is not only the ability to exchange bitstreams of information - an agent has to be able to interact in the human sense. An agent must be able to understand and reason about the goals of others in order to be able to negotiate to achieve its designed goal, for example. Linguistics and the philosophy of language are generally concerned with describing the communication. Of interest to us in this field is the notion of speech act, on which communication between agents is based. The speech act is an act performed by a speaker that not only carries the information but also performs an action. [20] For example, the sentence "I like tea, could you please make me some?" is a speech act because the sentence conveys not only information but also a request to make tea. Commonly included types of speech acts are apologies, complaints, compliments, responses, greetings, invitations, refusals, requests, and thanks. [21]

According to J. L. Austin [20], a speech act consists of three different components:

locutionary act

An locutionary act is to pronounce a sentence. The meaning of this sentence is all the verbal, social, and rhetorical meanings of the sentence, which are based on its semantics, syntax, and verbal aspect.

shared and reusable [24]. Then Cohen and Levesque offered a formal semantics for KQML in [25]. KQML is a communication language and is designed as a universal standard for agent-to-agent communication. KQML provides an extensible set of performatives that specify the types of communication that agents can have with each other. In addition, KQML contains some set of policies that specify a legal sequence of communication acts. KQML is predecessor of FIPA-ACL.

FIPA - ACL is a standard for agent communication developed by the Foundation for Intelligent Physical Agents (FIPA). The principles of this language are based on KQML, but it tries to define semantics and communication protocols better. Again, we have a set of messages that are fundamentally similar to performatives and are called communication acts [26]. FIPA - ACL, unlike KQML, has a closed set of acts, so new communication acts can only be created by combining basic acts. The components of the communication act model characterize both the reasons for which the act is chosen (rational effect, or RE) and the conditions that must be met for the act to be planned (feasibility preconditions, or FPs). An example of communication using the FIPA protocol is shown and described below:

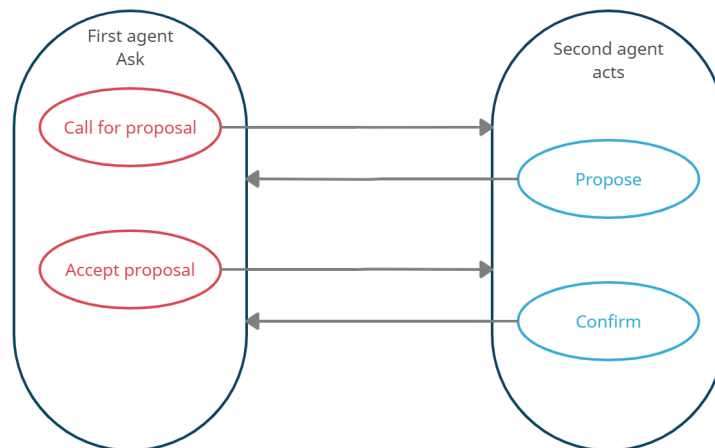


Figure 1.2: Example of simple communication exchange between two rational agents

In the Figure 1.2, we assume that agents behave rationally, so they try to achieve their goal by sending messages (acts). The first act sent from agent one is a **call for proposal**, which starts the negotiation process. This act may include a simple query about whether the other agent can perform an action, such as whether he can put wheels on a car. Or this act may contain some query, expecting to receive a proposal, for example, at what price the other agent is able to assemble the car. To this the second agent replies with a **propose** act, as already mentioned this is a response to a query from the

first agent. The agent also uses this act to communicate its intention to take action as soon as the preconditions are met. The first agent responds with **Accept Proposal**, confirming that it agrees to the proposal and may send additional information with this act, such as exactly when the action is to be performed. The second agent may additionally reply **Confirm** to this act, confirming to the first agent its desire to perform the action.

The following Figure 1.3 shows a state machine that represents the possible evolution of communication between two agents. This state machine is only used for a simple description of the communication, in real communication it can theoretically be infinite. One ongoing communication (message exchange) cannot give rise to another, in other words, there can be no branching, only one branch is always selected.

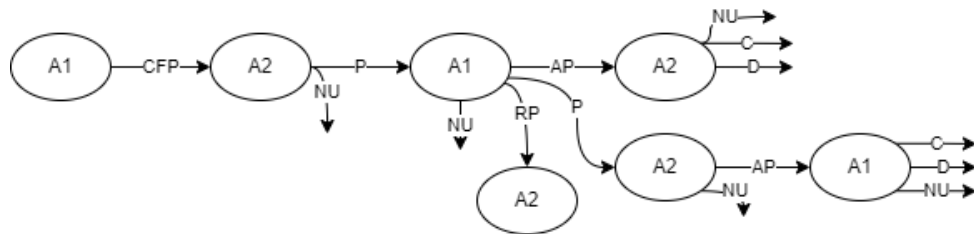


Figure 1.3: Example of communication state machine between two rational agents, CFP - call for proposal, NU - not understood, P - propose, AP - accept proposal, RP - reject proposal, C - Confirm, D - disconfirm

The appearance of the state machine depends on the implementation of the agents in a given system, for example, communication may terminate after an act propose. Agent one will again initiate communication by asking for a proposal. Agent two send propose or may send not understand if it cannot parse the message or, for example, expects different act content. Agent one can reject the proposal and thus end the entire communication, agree with the proposal or make a counter proposal. Depending on the agents' decisions, the communication continues. Disconfirm can be sent if for some reason the agent is no longer able to fulfill what was agreed upon.

Agents using these ACLs may only inform other agents of their actions or may perform more complex message exchanges, for example, for coordination, negotiation, or cooperation purposes. Coordination is a communication process between agents that results in rational behavior of the system as a whole with the aim of achieving an outcome in an optimal way. Negotiation is a communication process used to reach an agreement between agents. In other words, it is a negotiation of how to achieve the goals of all the agents involved. Cooperation is a communication process in which agents try to agree on a common solution to a given problem.

1.5 Modeling techniques

We have explained what an agent is, in what environment it can be situated, and have described the principle of communication between agents. We now outline various techniques for modeling, reasoning, and building multiagent systems. These techniques include knowledge of logic, probability theory, game theory, and optimization. The following information on the various techniques is taken from a book on multiagent systems by Shoham and Leyton-Brown [27].

1.5.1 Distributed Constraint Satisfaction

One of these techniques is Distributed Constraint Satisfaction. A constraint satisfaction problem (CSP) is defined by a set of variables and domains for each of the variables. Furthermore, the problem is defined by constraints on the values that the variables can take simultaneously. Agents work cooperatively to solve such a problem. The agents try to find the values of the variables that satisfy all constraints or decide that there is no solution. One well-known problem that belongs to CSP is graph coloring.

1.5.2 Distributed optimization

Distributed optimization is another technique that builds on the previous one by adding a global objective function that we want to optimize. This technique can be further divided into:

Distributed dynamic programming for path planning

Path planning aims to find the cheapest route using weighted links between the two states in the graph. For example, this approach can be used in internet networks or for car navigation where states represent the cities, links are available roads between cities, and the link weight is the time needed for crossing the road.

Distributed solutions to Markov Decision Problems

The optimal policy in Markov Decision Problems for a single agent is described by mutually recursive Bellman equations. For multiple agents, the global action is replaced by a vector of local actions of all agents. A more detailed description of Markov Decision Problems

matrix, where for each state of the world, described by the combination of all players' actions, the player's utility is determined.

Environments where the state of the world also depends on chance (Bayesian games) can also be described in this form. Example of this representation, described by a matrix, for the game Rock Paper Scissors is shown below in Table 1.1

	R	P	S
R	(0,0)	(-1,1)	(1,-1)
P	(1,-1)	(0,0)	(-1,1)
S	(-1,1)	(1,-1)	(0,0)

Table 1.1: game of Rock Paper Scissors described using Normal Form Representation. The rows describe the actions of the first player and the columns describe the actions of the second player. The first value in parentheses always describes the first player's utility

The choices that players make in the game are called pure strategy. One strategy can consist of multiple actions. The set of strategies for each player is called a strategy profile. Not all games can be described by a pure strategy, so there is also a mixed strategy, where a player chooses a pure strategy based on a probability distribution.

■ Solution concepts

If we are to choose the optimal strategy in a single-agent environment, we aim to maximize the payoff of this agent. The situation is much more complex in a multi-agent environment where each agent aims to maximize his profit. The choice of one agent's best strategy depends on the other agents' strategies. This problem is solved by finding specific subsets of outcomes that are somehow significant. These subsets are called solution concepts.

Pareto optimality

An outcome **a** is pareto optimal outcome, if that there is no other outcome **b** where one player would be better off and all other players have at least the utility as in **a**.

Social welfare

An outcome that maximizes the sum of all players utilities.

Nash equilibrium

Nash equilibrium is a concept where the optimal outcome of a game is

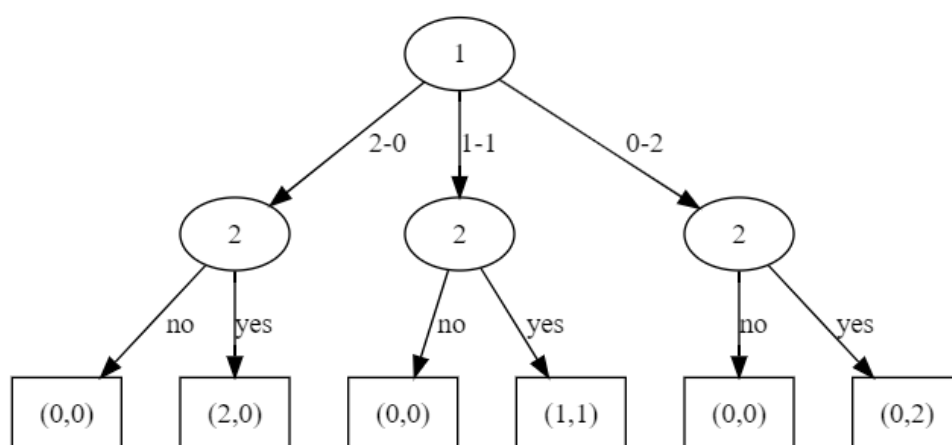


Figure 1.4: The Sharing game described by extensive-form game representation

Games with imperfect information in extensive form are represented in the same way as perfect games, and contain information sets (infosets) in addition. If the actions of another player are unobservable, then the states following these actions are indistinguishable and belong to the same info set.

Every finite game can be represented as an extensive form game with imperfect information. If we did not know the action of the first player in the mentioned Sharing game, then the tree would contain one info set. The new tree is shown in the Figure 1.5.

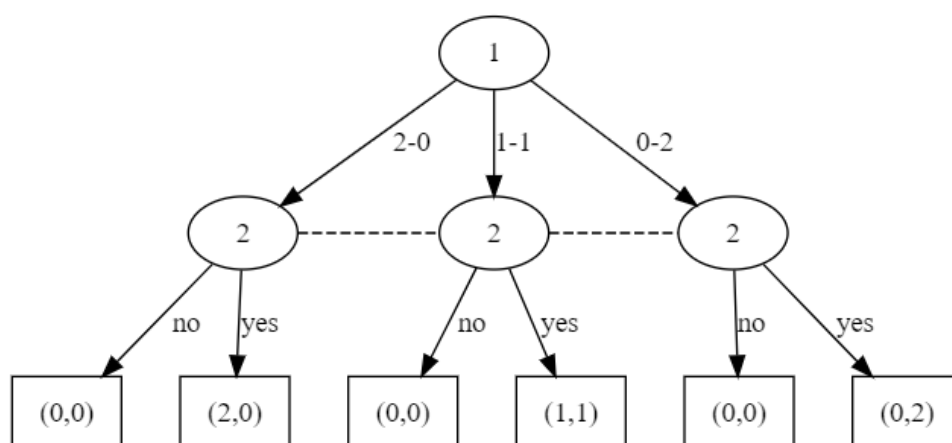


Figure 1.5: The Sharing game with imperfect information described by extensive-form game representation

■ 1.6.2 Coalitional game theory

As mentioned above, coalition games are about forming coalitions between agents. A coalition is a set of agents who try to maximize the gain of the whole coalition. Finding the optimal coalition structure is an NP-hard problem, since the number of all possible coalitions grows exponentially as the number of players increases.

Each agent gains some worth from the total coalition profit independently of the actions of the other players. The solution to the game is a set of worth allocations for all players. We describe here two basic solution concepts.

Shapley value

Shapley value describes each player's contribution to the coalition. This value is fair, meaning that players with the same contribution receive the same amount and a player who has no contribution receives zero.

Core

Core "is a set of efficient allocations upon which no coalition can improve" [28]. Core is stable and can be considered analogous to the Nash equilibrium in non-cooperative games. The problem with the core is that it is not always unique, it can be empty, and it is also not easy to compute.

■ 1.6.3 Multiagent resource allocation and Auctions

■ Multiagent resource allocation

Multiagent resource allocation is a process in which we try to distribute a certain number of objects among agents. Each agent has some own object rating, a preference.

The allocation of objects among agents is measured by social welfare. Allocation can be either efficient, where we give the object to the one who values it the most, or fair, where the goal is to make the agent who is worst off the best off.

■ Auctions

An auction is a protocol that allows agents to express an interest in a resource, and this interest is used to make decisions about resource allocation and agent payments [27]. Auctions can be held for one item (single-good), for multiple identical items (multi-unit) or for multiple different items (multi-item). Agents can also buy, sell, or both (exchange).

We describe here the most common example of auctions, agents buying one or more items. We start with the simple case of a single item. Auctions can be divided into five categories, depending on how they are conducted:

English

The most famous is the English auction. The auctioneer starts the bidding at the starting price. The bidders then report their rising bids, and when no one else raises, the last bidder is the winner and gets the item for that price.

Japanese

In the Japanese auction, the auctioneer starts at the starting price and all bidders stand. The auctioneer gradually raises the price and when the price exceeds the amount the bidder is willing to pay, he sits down. The last one standing is the winner and gets the item for that price.

Dutch

In a Dutch auction, the price of the item starts high and gradually decreases. At some point, the bidder reports "mine" and gets the item for the actual price. There is a problem in this auction because the agent does not know the valuations of the other agents. There is a trade-off between the probability of winning and the amount the agent pays. There is no dominant strategy in this type of auction.

First price sealed bid

In this auction, bidders write their valuation on a piece of paper. The auctioneer will then sell the item to the highest bidder at the price they wrote. This type of auction is similar to a Dutch auction in terms of agent behavior.

Second price sealed bid

In this auction, bidders write their valuation on a piece of paper. The auctioneer will then sell the item to the highest bidder at the price written by the second highest bidder. This type of auction is similar to English and Japanese auctions in terms of agent behavior. In this auction, writing the agent's valuation as the final price is the best option, this way the agent will achieve his best result.

In sealed bid auctions, we can introduce a reserve price below which the item will not sell.

If we are going to auction for several different items, we can use the Vickrey-Clarke-Groves (VCG) auction. This auction is a sealed bid auction, so each bidder will report their valuation for the items without knowing the valuations of the others. The auctioneer then allocates the items in a socially optimal way and charges each bidder the damage he caused to the others. It is similar to a second price sealed bid auction and therefore the best option for the agent is a true valuation of the items.



Chapter 2

Problem Description

This chapter describes the Montrac production line, a description of the task and its problems, a description of the currently used MES solution and a description of the first multiagent solution.



2.1 Description of the Montrac test line

Montrac is the name of the testing line located in the Testbed for industry 4.0 at the Czech Institute of Informatics, Robotics and Cybernetics Czech Technical University. This line is a smaller version of a real industrial production line and is used to test new technologies and principles. For this reason, the resulting algorithm was tested on this line. The Figure 2.1 shows a layout of the line.

Five industrial robots of three different types from KUKA are on the line:

- The R1-R3 robots are *KUKA KR 10 Agilus-2* - High-speed industrial 6-axis robots programmed in KRL. These robots are used on the production line for quick pick and place operations. It is possible to automatically change the tools used or manually attach an electric screwdriver. The manual is available here [29].
- R10 is *KUKA LBR iiwa 14* - A modern cooperative 7-axis robot that is programmed in Java. This robot is used on the production line for pick and place operations. It can be used for more complex operations thanks to its 7 axes. At the same time, this robot can work in collaborative mode with human. The manual is available here [30].
- R20 is the *KUKA Cybertech KR 8 R1620* - A fast industrial 6-axis robot with a higher payload capacity programmed in KRL. This robot is used on the production line for pick and place operations. It has a camera mounted on it for more accurate localization of parts. This robot is the only one that can unload new parts from the stock shelf. The manual is available here [31].

The Montrac monorail transportation system connects the robots. It consists of rails called tracks, track curves, track switches (letter "C" in the figure), and positioning units (letter "S" in the figure) that ensure the exact position of shuttles in working cells. The Montrac monorail on the production line is used to transport parts or product between robots. The parts are placed in shelves that are transported by Automated Guided Vehicles (AGV). Imported parts are unloaded onto the shuttles using the R20 and distributed between the individual robots.

These robots communicate with Siemens PLC via Profinet. This communication includes safety variables, control variables, and the acquisition of diagnostic data from the robots. The PLC uses MQTT protocol to send diagnostic data and OPCUA protocol to communicate with the control system (scheduler/planner). An HMI panel is used to monitor the current status of the production line.

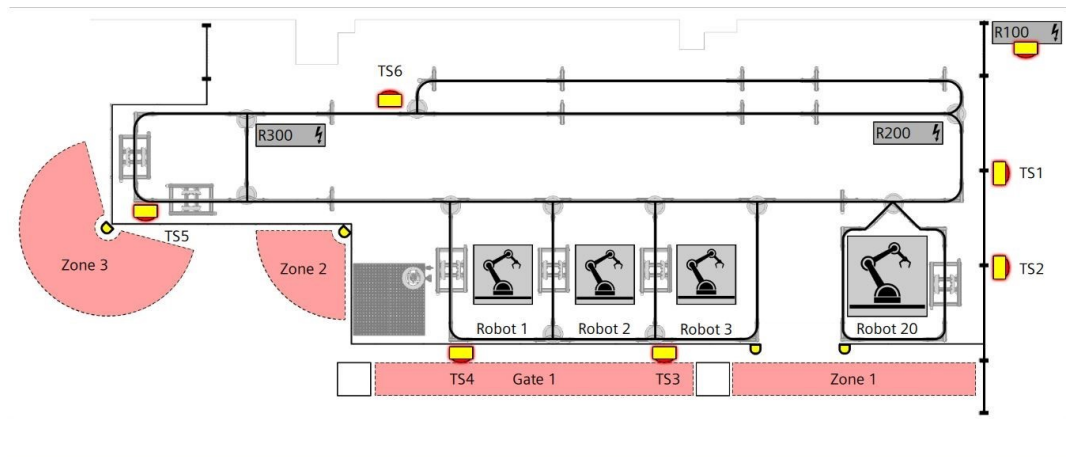


Figure 2.1: Diagram of Montrac safety zones

Safety features that meet ISO standards for safe industrial environment (the ISO standards used are mentioned in the Table B.1 in the Appendix B) are placed around the production line. There are seven total stops around the production line, which immediately stops all robots and the conveyor. A fence encloses part of the line to prevent access by people and safety devices from SICK monitors the rest of the line. If one of these devices detects an object, the PLC safety program stops all robots. The devices have their detection zones; see Figure 2.1.

- Zone 1 is secured by two micro scanners
- Zones 2 and 3 are secured by a safety scanner S3000
- Zone Gate 1 is secured by a C4000 light curtain

This simulation production line contains industrial robots, communicates using industrial protocols, and includes safety features, so it can be considered a faithful copy of a typical industrial line.

■ 2.2 Thesis problem description

This work aims to design, implement and test an algorithm that schedules the production of one product on any production line if such schedule exists. The principles of multi-agent systems are used and will be described later. Another goal is to deploy the algorithm on a real production line. Thus, it

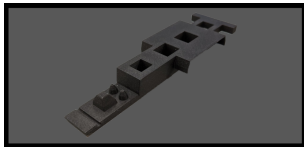
will be possible to process a product order, schedule production and execute it.

The production line contains robots that have specific capabilities and thus can perform some sub-operations, sometimes different to each other. At the same time, each robot has a magazine where it can store parts. A conveyor connects the robots, but there may only be a connection between some of the robots. The connection between robots can be unidirectional.

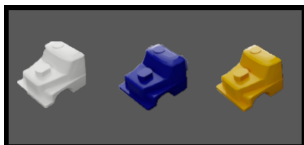
Exact sequence of sub-operations is known. An example of such an operation may be fitting a car hood. Each sub-operation needs some components to be successfully performed. For example the hood mentioned above.

An example of such production is the current demonstration of a demo car on the Montrac line. The demo car consists of three parts:

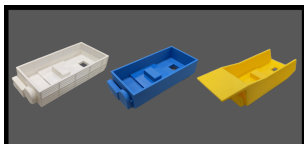
- chassis



- cabin, which can be of different colors.



- body, which has different type variations and colors



The manufacturing process consists of three steps: chassis placement, cabin placement and body placement.

2.3 MES = Description of actual solution concept

Production on the Montrac line is now controlled by a manufacturing execution system (MES). This system consists of a planner, a digital twin, a scheduler, and a plan executor. See Figure 2.2.

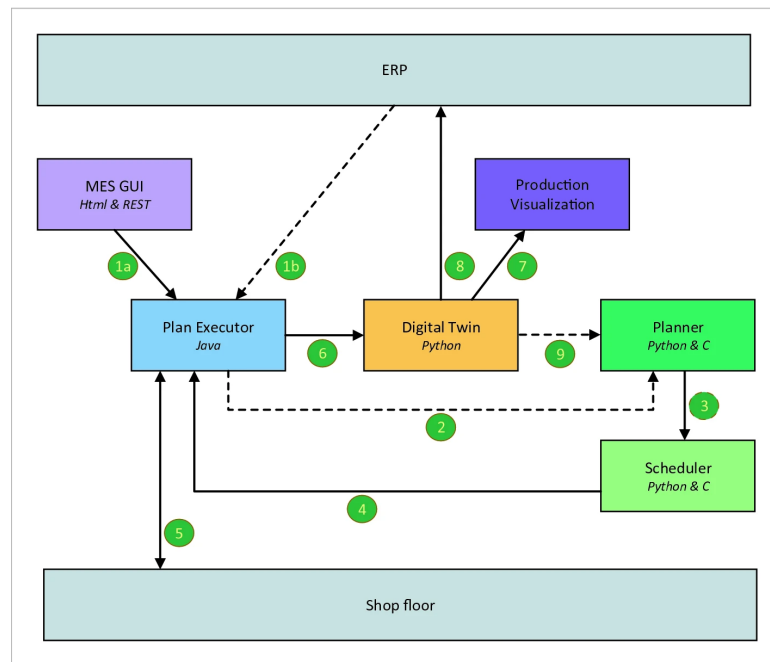


Figure 2.2: Architecture of MES

The plan executor receives orders, which it passes on to the planner, which calculates the plan. This plan is passed to the scheduler, which enriches it with schedule information and hands it over back to the plan executor. The plan executor now assigns tasks to individual robots with the completed plan and updates the digital twin whenever an operation starts or finishes. The digital twin, therefore, contains a detailed history of the production.

A more detailed description of this system is given in [32].

Although improvements are still being made to the mentioned MES there is an effort to develop an alternative solution. The computational complexity of MES planning increases exponentially as the complexity of the production line increases. The goal is to propose an alternative solution that will not have such computational demands due to the simplification of the problem, and we will be able to compare the two solutions.

■ 2.4 MAS = Description of the developed version of the solution that uses multi-agent systems

This section describes a solution that uses a multiagent approach. It describes how production scheduling is done, how the price of the operation is calculated by each agent and what problems this solution has. This solution is an alternative approach to the problem using agent-based systems, as opposed to an MES system (2.3) that uses central planning.

■ 2.4.1 Description of agents

The main reason for using a multiagent system is the flexibility of the entire production line. With this approach, a factory can easily modify the number of robots on the production line or their capabilities. Each robot or transport system is represented by an autonomous agent who knows its capabilities and can determine the time required for each operation. The determination of the time depends on the current state of the robot (time utilization, condition of parts in the magazine).

All agents are part of the multiagent platform that uses RabbitMQ for messaging and provides a database and logging. This platform defines the agent interfaces, but the actual functionality (implementation) of the agent can be different. This approach allows us to take the definition of an agent, for example, from a robot supplier and integrate it into a production line with ease. This platform is still under development and will be published in a paper later this year.

Agents communicate with each other using messages, referring to the FIPA communication protocol. (Section: Multiagent communication languages)

■ 2.4.2 Description of testing solution for production scheduling

The existing solution for production scheduling is based on an immediate reactive approach with a plan, commit, and execute negotiation protocol introduced by Petr Kadera and Pavel Tichy [33]. In conjunction with the agent system, this approach performs the same four steps for each operation

from the production workflow. These steps are query, plan, commit, and execute.

When a customer order is received, a new production agent is created to represent that order. The order includes a production steps from which the agent takes each operation in sequence and finds a solution for the execution.

- The first step is query - in this step, the production agent queries a database to determine which robots can perform the operation, that is, have the necessary capability. The database responds with a list of agents.
- The second step is plan (scheduling) - in this step, a *call for proposal* message is sent to all agents in the list, which means asking whether the agent will perform the operation and, if so, at what cost. Each agent checks for itself whether it can perform the operation. There may be a situation where the agent needs the cooperation of other agents in the system to perform the operation, for example, for importing material, in which case it initiates communication with the agent again using query and then the *call for proposal* message and waits for a price proposal. If it can perform the operation, it calculates the price (adds up all the subprices) and replies to the production agent. The production agent therefore receives a price proposal or refusal of the operation from each agent.
- The third step is commit - in this step, the production agent chooses the proposal and sends an acknowledgment of this proposal to the agent from whom it received the proposal. The agent checks its status to see if it can still perform the operation at the given price and, if so, sends an acknowledgment. The other agents who sent a price proposal will receive a rejection from the order agent.
- The last step is execute - in this step, the production agent sends a request to the agent to start executing the operation. The agent sends back information about the start of the operation (*operation started*) and the successful completion of the operation (*operation finished*).

■ 2.4.3 Description of the proposal price calculation

The calculation of the total price of the operation that is sent to the production agent consists of several components. For clarity, we describe the price calculation in Figure 2.3.

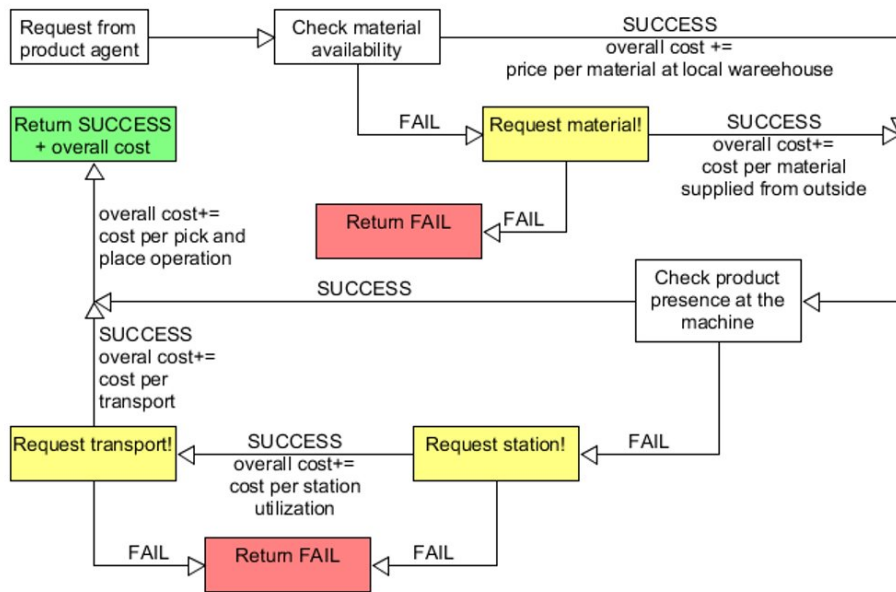


Figure 2.3: Description of the price calculation for the operation, [2]

The agent first checks the availability of the required materials in its warehouse. If it does not have any material, it arranges for its import. The material owner (warehouse or another robot) and the conveyor are involved in the import of the material. The import price is based on both the owner's and the conveyor's workload. This may result in several different offers from different agents for the import of one type of material. So, from the offers received, the agent selects the cheapest offer and includes it in the price calculation. There may be more than one type of material missing, so the total price for importing the material is the sum of these prices.

The next step is to check whether the order in progress is in the agent's workspace. If not, it is necessary to arrange delivery of the work-in-progress order to the agent. From these steps, another part of the total price is created.

The last step is the execution of the operation itself. This step may, for example, consist of changing the tool and performing the operation itself.

The total price is the sum of the cost of importing the material, the price of importing the work-in-progress order, and the operation price.

2.4.4 Problems of current solution

One of the problems of the current solution is the algorithm used. This algorithm uses a greedy approach and only produce good results for some production lines. When applying the algorithm to a line where the transport graph is complete, no problem arises. But a simple example can show that this algorithm will not find a solution even if it exists.

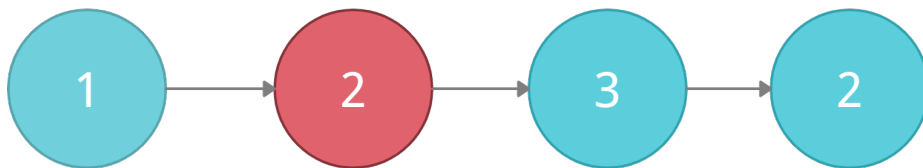


Figure 2.4: A production line with four robots, where each robot can perform an operation described by a number. Between the robots is a unidirectional conveyor belt, which is shown by arrows

There are four robots in the Figure 2.4, where each robot knows one elementary operation described by a number. The conveyor is unidirectional and therefore there is no possibility of return. To produce a product, operations 1,2 and 3 need to be performed in that order. The product just produced has completed the first operation and the plan for the second operation is being planned. The robot shown by the red circle will return a higher price than the last robot. The product then moves to the last robot from which there is no path back; therefore, the product cannot be completed. In this example, there is a solution; the robot in the red circle would perform the second step with the higher price.

The second problem is of an implementation nature. This solution is written in pure Java as a test solution of the algorithm. Since then, the platform mentioned above has been rewritten using opensource production ready tools and frameworks. The advantage of this approach is the long-term sustainability and maintainability of the platform. The new algorithm will be built on this new platform.

Chapter 3

Design of a new algorithm

This chapter describes the design of the necessary components and a new algorithm that is complete for single product scheduling on any production line.

3.1 Design of communication automata

The multi-agent platform provides communication between agents but does not manage the content of the communication. I designed communication state machines to make communication between agents more understandable from a logging perspective. The content type/performative of each message corresponds to the FIPA protocol (Section: Multiagent communication languages), and it is clearly defined what response to this message can occur. Each message contains a recipient, message ID, performative, conversation (state machine) ID, and content.

In our case, three types of communication between agents can occur.

- informing about a state change - for example, completing an operation or updating parts in the agent's stack
- a request to perform some one-time action - for example, a query about an agent's capabilities or a request to start executing an operation

- a plan-commit - Asking about the price of the operation and then possibly confirming it. The operation could be a pick or import of parts, for example

For the first case (inform), a communication state machine consists of one node representing the message inform.

For the second case (request), the state automaton looks like this (Figure 3.1):

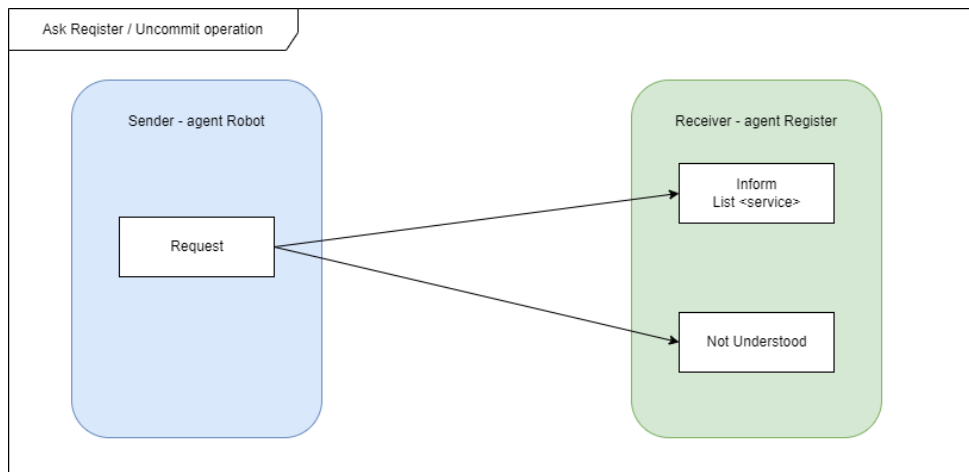


Figure 3.1: state communication automaton for Request type messages

For the third case (plan-commit), the state automaton looks like this (Figure 3.2):

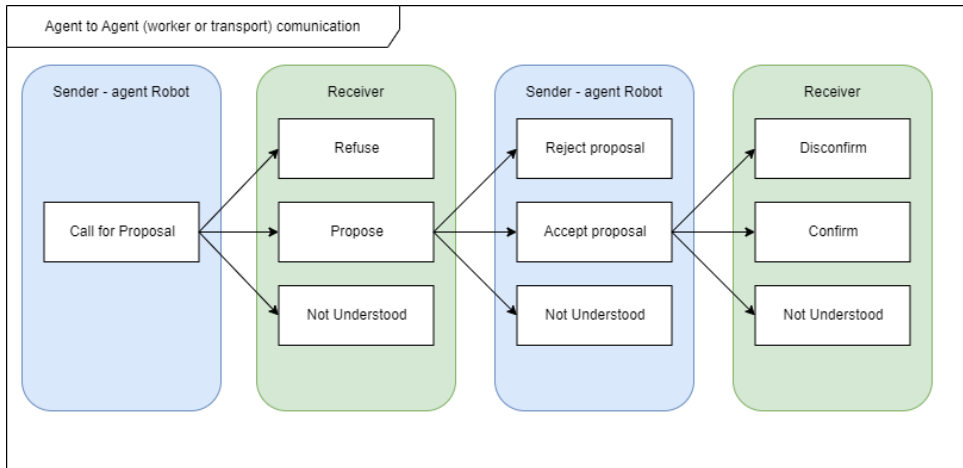


Figure 3.2: state communication automaton (plan-commit) for production operation or material import

The agent starts the conversation by sending a *call for proposal* message. This message defines the action (operation execution, material import). If the receiver does not understand the action, it replies with a *Not Understood* message. If it understands the message, it calculates the price of the action and replies with a proposal. If the receiver cannot perform the action, for example, there is no way to transport the product or insufficient material, it replies with a *Refuse* message. The Sender either accepts the proposal with *Accept proposal*, thus performing a commit operation, or rejects it with *Reject proposal*. The receiver checks if the cost of the operation has changed and responds accordingly.

3.2 Finding a suitable algorithm that is complete

This section describes the solution search. An attempt has been made to take advantage of common production line scheduling solutions.

3.2.1 Parallel scheduling

The first effort to find a complete algorithm to solve our problem was with the help of parallel schedulers. So that, all production steps would be scheduled at once, and in case of a problem with the sequence of steps, for example, with the transportation of the product, the following steps would be rescheduled. An

the price of an operation to the scheduler since the price may depend on the system's current state and may change. For this reason, we do not use schedulers, otherwise commonly used in industry, to solve the problem.

3.2.3 Chronological backtracking

The proposed and implemented algorithm is based on chronological backtracking. This algorithm can be very slow for larger state spaces. We perform scheduling only for operations from the product workflow, and the agents themselves handle sub-operations. For this reason, we can use this algorithm because the state space will not be so large as to affect the speed of the computation noticeably. At the same time, this algorithm can be extended further to speed up the search solutions and to find more optimal solutions.

The algorithm is very similar to the current solution that uses agents (Description of testing solution for production scheduling), and uses the same steps as *query*, *plan*, *commit* and *execute*. The first two steps, *query* and *plan*, are identical. Thus, a call for proposal is sent out to capable agents, and we get responses from them.

During the *commit* step, the lowest cost agent is selected, and a confirmation is sent to it. Nothing is sent to the other agents who responded positively. If no positive response is received during the plan step, backtracking occurs. For the last committed operation, the reverse operation is performed - UnCommit. All resources are unreserved. Another agent that responded positively in the previous step is selected, and scheduling continues. Backtracking is performed to the next level if there is no such agent. If the solution is found, the *product agent* starts sending execution requests (performing *execute* step) in the same order in which we booked the operations for each agent.

The solution of the example Figure 2.4 on which the current algorithm did not find a solution would be as follows.

For the first operation, we only get a response from R1 and therefore commit to this robot. For the second operation, we get a response from R2 and R4. R4 has a lower total cost, so we commit to him. For operation three, there is no solution from the current state. We perform an uncommit on R4 and select another agent that has responded positively. In our case, it is only R2. So we commit to R2 and continue with the planning. R3 can now perform the last operation. Finally, we send an execute to R1, R2, and R3.

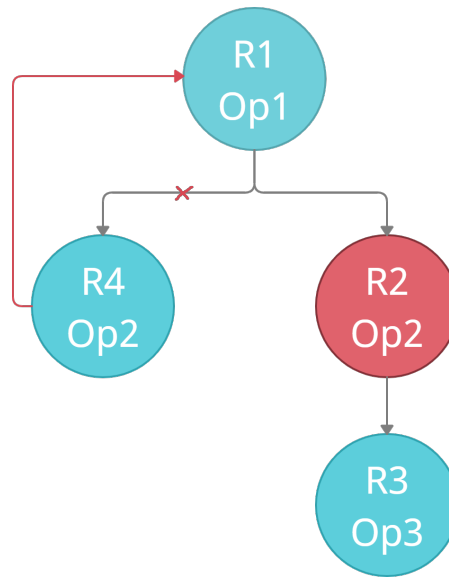


Figure 3.3: Illustration of the algorithm with backtracking on an example with four robots

Backtracking can occur in two cases. In the first case when there is no possibility of transferring the work in progress to any agent with the skills for the next production step. The second case occurs if we can transfer the product, but none of the agents to whom I can transfer the product has the necessary parts to perform the operation, and no one can deliver them to them.

■ Graph pruning

The first case can be avoided by performing graph pruning before the search begins. We know what transport exists on the production line between the machines, and therefore we know the context of the graph. The first step is to partition the graph into strongly connected components. The next step is to create a directed acyclic graph (DAG) between the components created in the first step. Next, we know the product's production process and each agent's capabilities. The last step is state space pruning. We proceed from the last operation. If there is no path from a component to even one agent that can perform the operation, we can discard that component because we can never solve it. We can exclude in advance paths that cannot reach a solution and thus speed up the algorithm.

An example of pruning is shown in the following example. The production

line consists of five agents, and the product manufacturing process consists of 4 different operations. The capabilities of the agents are divided as follows:

- Operation 1 -> Agent 1, Agent 4
- Operation 2 -> Agent 2, Agent 5
- Operation 3 -> Agent 1, Agent 3, Agent 4
- Operation 4 -> Agent 4, Agent 5

The transport link between the agents is shown in the Figure 3.4.

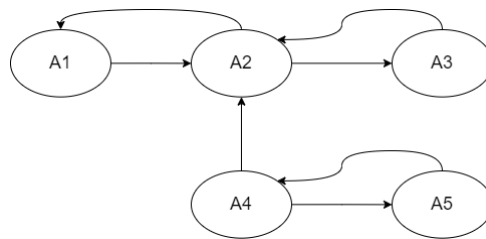


Figure 3.4: The transport links between the agents on the line - pruning example

After splitting into strongly continuous components, we are left with two nodes. Now we create a DAG between the components, see Figure 3.5

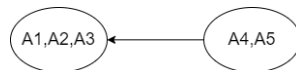


Figure 3.5: DAG between strongly continuous components

The last operation can only be performed by agents that are in component [A4, A5]. So we can exclude component [A1, A2, A3] from the state space. This will make the state space for this product very simple, see Figure 3.6.

Pruning hasn't been implemented yet, because the test line in Testbed is one highly contiguous component and therefore would have no effect. If this solution is found to be beneficial, pruning will be implemented and the optimization of the solution will be addressed as next.

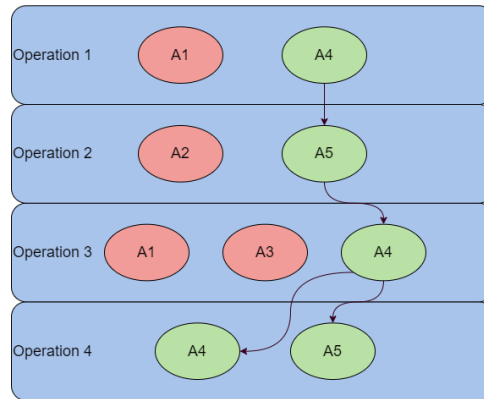


Figure 3.6: Simplifying the example state space by pruning

■ **time complexity of the algorithm**

The time complexity of the algorithm depends on the number of agents in the system, the number of operations needed to produce the product and the possible transport between agents. The behaviour of the algorithm corresponds to a deep tree search.

Worst-case time complexity occurs if there is a connection between all agents and the product cannot be produced. In this case, the entire space is searched and the time complexity corresponds to this equation 3.1. Where N is the number of operations required to produce the product and K is the number of agents in the system.

$$O(N \cdot K + (N - 1) \cdot K^2) \tag{3.1}$$

The time complexity of the algorithm can be reduced by implementing the aforementioned pruning, which can significantly reduce the state space or by checking the sufficiency of the necessary components before running the algorithm.



Chapter 4

Implementation of the new algorithm

This chapter describes the actual implementation of the selected algorithm - chronological backtracking.

The algorithm was implemented in Java. I used the following technologies. **Spring Boot**, which allows you to create stand-alone, production-grade Spring-based Applications that you can "just run". **Lombok** is a popular and widely used Java library used to minimize or remove the boilerplate code. It saves time and effort. **Spring AMQP**, which provides a "template" as a high-level abstraction for sending and receiving messages. **Spring Data JPA**, Its purpose is to unify and easily access the different persistence stores, both relational database systems and NoSQL data stores. **Jackson-core**, which defines public API for writing JSON content.

I implemented one version of the code for all agents. According to the *service name*, the agents are divided into *order agents*, *database*, and *work agents* (robots, transport).

I developed the algorithm on a new multiagent platform, starting from scratch. The first step was implementing the original agent solution and then adding the backtracking extension.

4.1 Agents implementation without transport

The first step was a proof of concept. I tested the functionality of the multiagent platform by producing a simple product consisting of four parts. I did not address inter-agent transport or code extensibility. After the functionality was verified, code refactoring was needed. I created a class to represent the agent (robot) capabilities and a database that contains the capabilities of each agent in the system, resource status in the system, and products-in-progress information. The agent represented the database because the platform did not contain its own database at the time of implementation. After these steps, I tested the production of the mentioned product again using the database.

4.2 Agents implementation with product transport

The next step was to add transport between agents. In the first step, I only handled the transport of the product in progress. I know the transport edges and their prices between nodes (agents). I assume that the end of one edge can be connected to the beginning of the next edge if it ends at the same point. A conveyor can be connected physically, or an agent (robot) at a given node can transfer the product to the other conveyor. Based on this information, all possible paths can be found by Dijkstra's algorithm. This algorithm was taken from [36] and then slightly modified. Each of these paths is the shortest possible. The following Figure 4.1 describes the transport in the system using each edge. The Table 4.1 below lists the paths found and their prices using Dijkstra's algorithm.

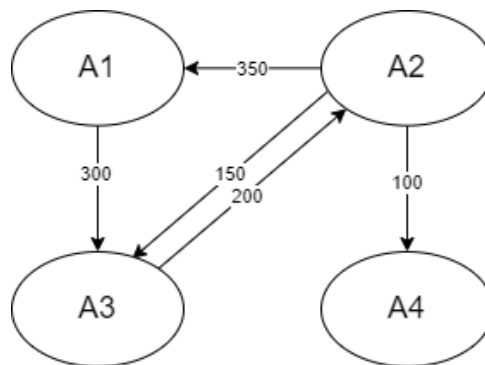


Figure 4.1: Transport routes in system

A single agent represents all the transportation in the system. Under this

From	To	Cost	From	To	Cost
A1	A2	500	A2	A1	350
A1	A3	300	A2	A3	150
A1	A4	600	A2	A4	100
A3	A1	550	A4	A1	None
A3	A2	200	A4	A2	None
A3	A4	300	A4	A3	None

Table 4.1: Price of paths between nodes in Figure 4.1

agent, there may be other agents that will represent different conveyors. For simplicity of testing, I use only the parent agent, i.e., one type of transport.

After adding a product transport, I cannot respond to the *production agent* immediately, and communication with the *transport agent* must occur. Busy wait can't be used because the program runs single-threaded, and there would be no processing of the new incoming messages. Therefore, I created a new class that maintains and updates the state of the pricing conversation.

This class provides three main functions: computing the price for an operation (Figure 4.2), commit the computed operation (Figure 4.3), and executing the operation. The execution of the operation is straightforward; the sub-operations start executing sequentially. If this operation is handled by another agent (transport), a message is sent, and a response is awaited. After the operation is executed, a message is sent to the database to update the agent's parts status.

When a *CFP message* is received from a *production agent*, an instance of this object is created and stored. It is addressable by *conversation ID*. When an *accept proposal* or *execute request* is received, this instance is accessed. The instance is deleted if the step is successfully executed, the production agent rejects the operation, or when there is no possibility of completing the operation (no transport path, not enough components).

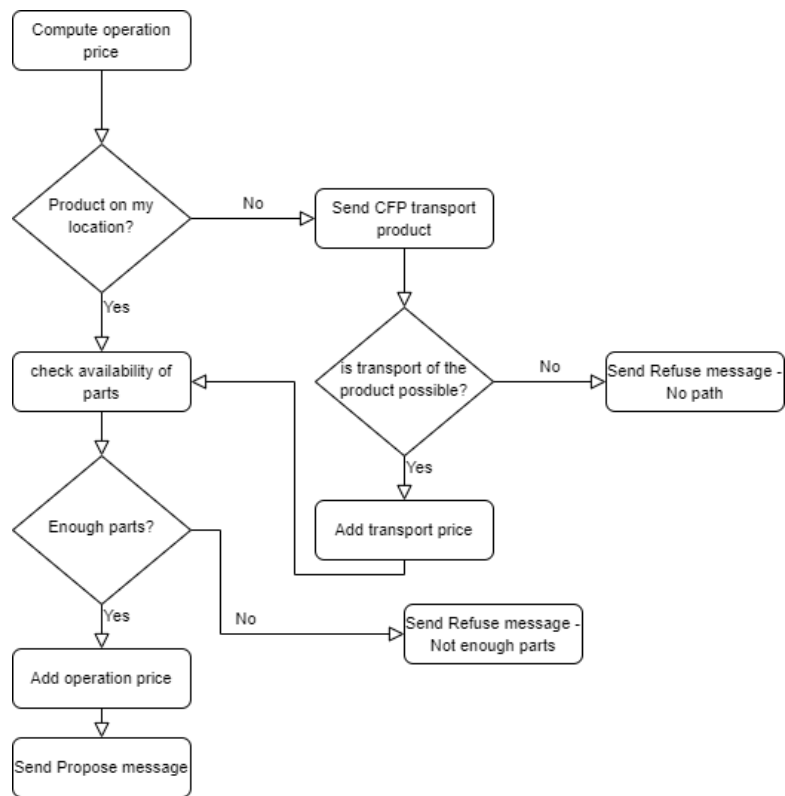


Figure 4.2: Diagram describing computing of price for an operation

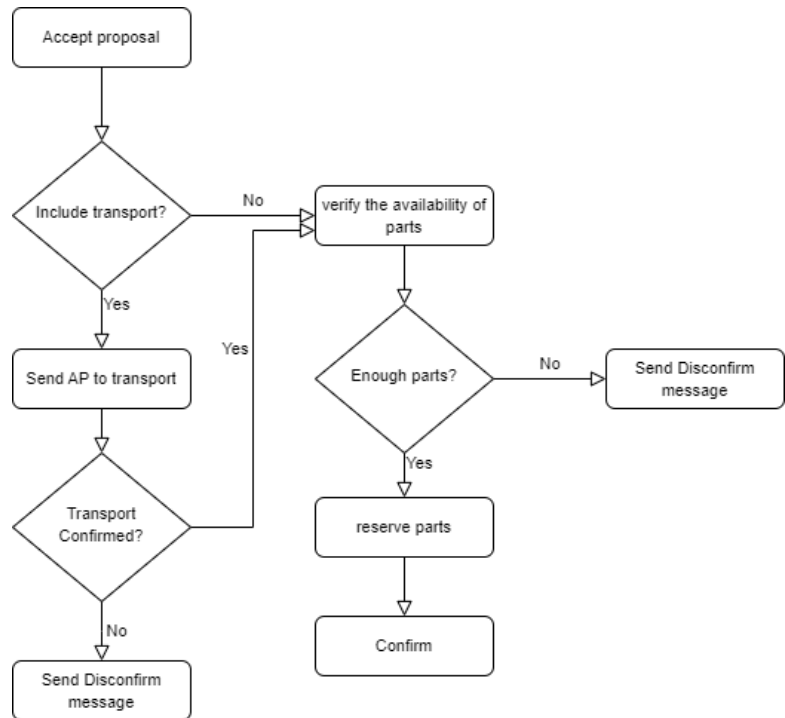


Figure 4.3: Diagram describing commit for an operation

4.3 Agents implementation with product and parts transport

After verifying the code's functionality, I proceeded to the last implementation step to deploy the original agent solution. It was necessary to provide transportation of parts between agents in case of shortages. As in the previous step, I had to maintain information about the state of the conversation between the agents. In this case, two more significant conversations between agents could have arisen - arranging the transport of parts and determining the price of an agent's operation. I duplicated the class I created and modified it for both cases.

The class that maintains the state of the transport of parts has been simplified. The following figures show the state machines for determining the price of transporting a part (Figure 4.4) and for booking an order (commit) (Figure 4.5). The executing step performs the operations in the given order. Also, the agent updates the status of the parts for itself and sends a message to the database.

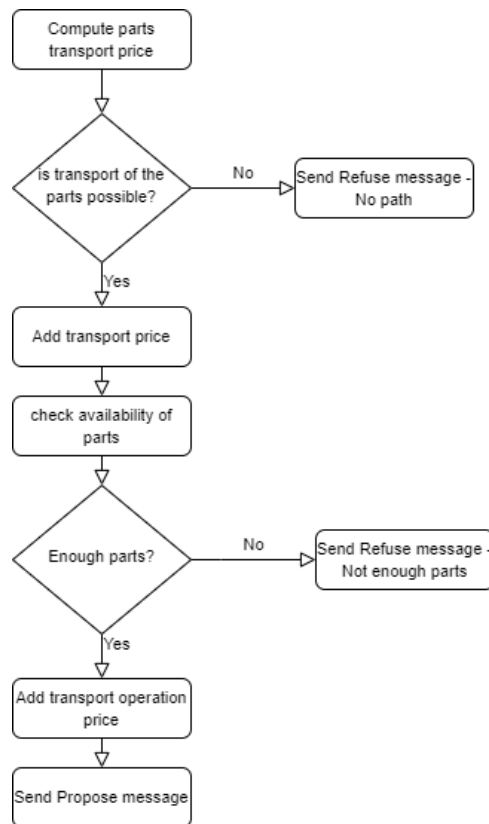


Figure 4.4: Diagram describing computing of price for a parts transport

For simplicity, we can ship any number of parts of one type at a time. On a real production line, there is a limited capacity for the transport cart and a limited size of the parts bins. If multiple parts bins need to be moved, the cost of the robot transport operation will increase. And if the bins would not fit on one transport cart, the transport price would increase. In both cases, this does not change the solvability of the problem.

At the same time, since we are scheduling production for only one product, the state of the parts at the agent cannot change during the scheduling event. Checking the state of the parts at the agent is redundant in this case, but this step is necessary if we would like to plan multiple orders at once.

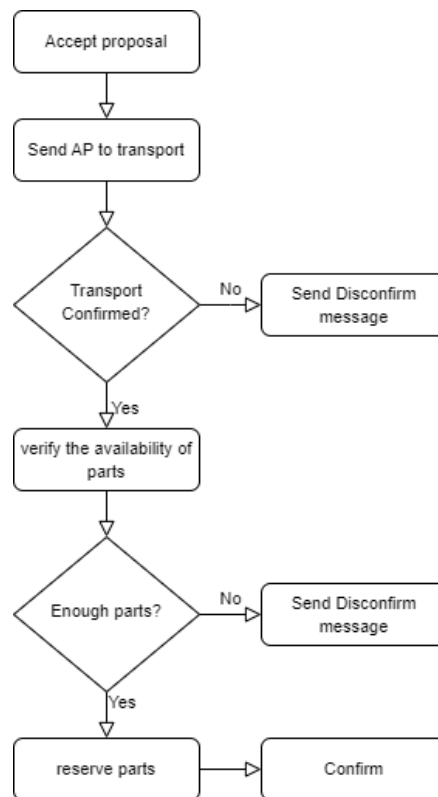


Figure 4.5: Diagram describing commit for a parts transport

Also, there can be no rejection during confirmation of the operation, as production is now planned for only one product. The future work should be able to schedule multiple products at once, and confirmation might be needed; therefore, this step is included.

The state machine for calculating the price of an operation is more complex, see Figure 4.6. The first step is to check the product's location and possibly arrange its transport. The second step is to arrange transport of the missing parts. Multiple types and numbers of parts may be needed for a given

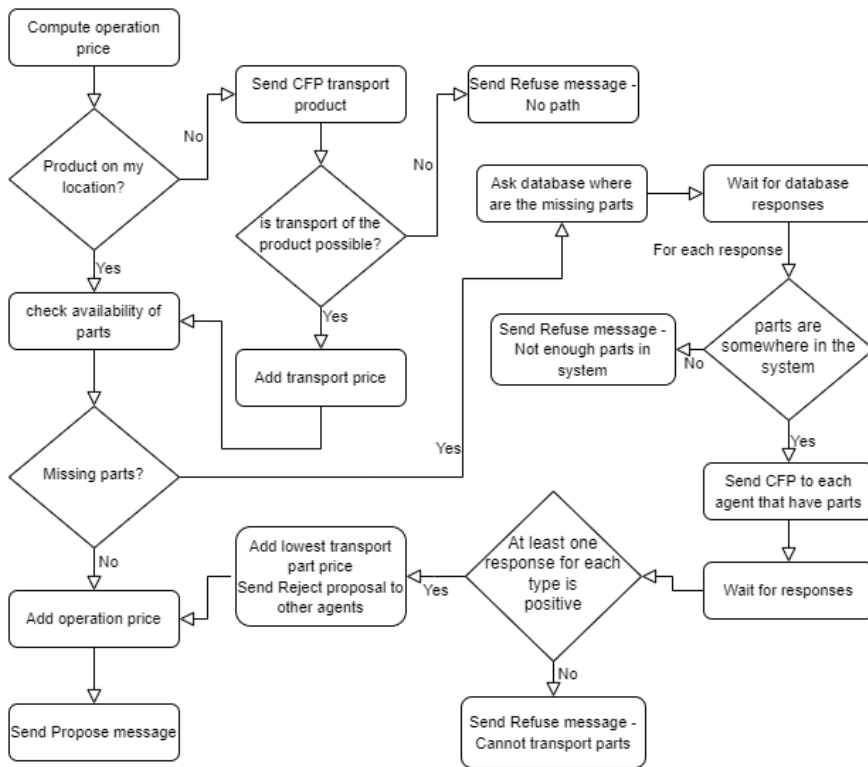


Figure 4.6: Diagram describing computing of price with product and parts transport

operation. The agent checks its current stock of parts and accordingly queries the database to determine which agent has the exact or higher number of missing parts. It makes one query for each type. This approach may have a problem if there are enough parts in the system for a given operation, but they are spread among the agents. This problem is addressed in the Use case 5. In this case, the database will return an empty field, and the agent will return the Refuse message. An alternative approach is to query for one missing part at a time and reserve them one at a time, which could put a strain on the communication network. Another approach is to query the database for who has what number of parts, and the agent would decide for itself from whom and how many it wants to import. The complexity of this approach would increase considerably if any agent refused it. Then the import distribution would need to be changed by the type of rejection, and some previous queries would need to be cancelled and new ones sent out. The chosen approach is most appropriate for a typical industrial operation where parts are imported regularly.

The database will return a list of agents that have the required number of parts. The agent sends a *CFP message* to calculate the cost of importing parts to all agents and waits for responses. It chooses the best possible one from the responses, adds its price, and rejects the other offers. Finally, it

adds the price of the operation itself and sends the price proposal to the *production agent*.

The operation and booking of resources are confirmed sequentially according to the previous price calculation (product transport, component transport/s, agent operation). The confirmation message is sent to the *production agent* if all agents confirm the operation. Again, in single product planning, there can be no non-confirmation.

The algorithm implemented in this way behaved similarly to the original agent solution. In this step, backtracking was integrated into the algorithm.

■ 4.3.1 Agents implementation with backtracking

As described above (Description of testing solution for production scheduling), the test ("greedy") algorithm executes each process step as soon as it is scheduled. To perform backtracking, it is not possible to execute the steps immediately. The whole solution has to be created first, and then the execution can start. This is a major change in the implementation described below. Also, for backtracking, the ability to UnCommit operations is needed. That is a method that releases the reserved resources that are related to the operation. This method has been added to the classes maintaining state about the operation. Thus, the class contains methods for pricing the operation, reserving resources, unreserving resources, and executing the operation.

I will describe the algorithm flow using the following example. The transport connections between the agents are described by the Figure 4.8. The product consists of 4 parts - chassis, tires, cabin, and body. Agent operations are named after the component and each operation requires one component of the same name, except for the tires operation which requires four tires. The capabilities of agents and placement of the components are described in Table 4.2. The cost of the operation to load the parts onto the conveyor is zero.

The flow through the algorithm is described in the Figure 4.9. In the first step, the product does not exist, so we do not have to deal with its transport. The first operation is the chassis, and it is possible operation for agents *A1* and *A3*. We ask for the price of the operation and get the prices from the agents. We choose the cheaper agent, make a commit and proceed to the second step. This step will proceed similarly to the first step. The third step follows. The operation is cheaper from agent *A4*, a commit is made to

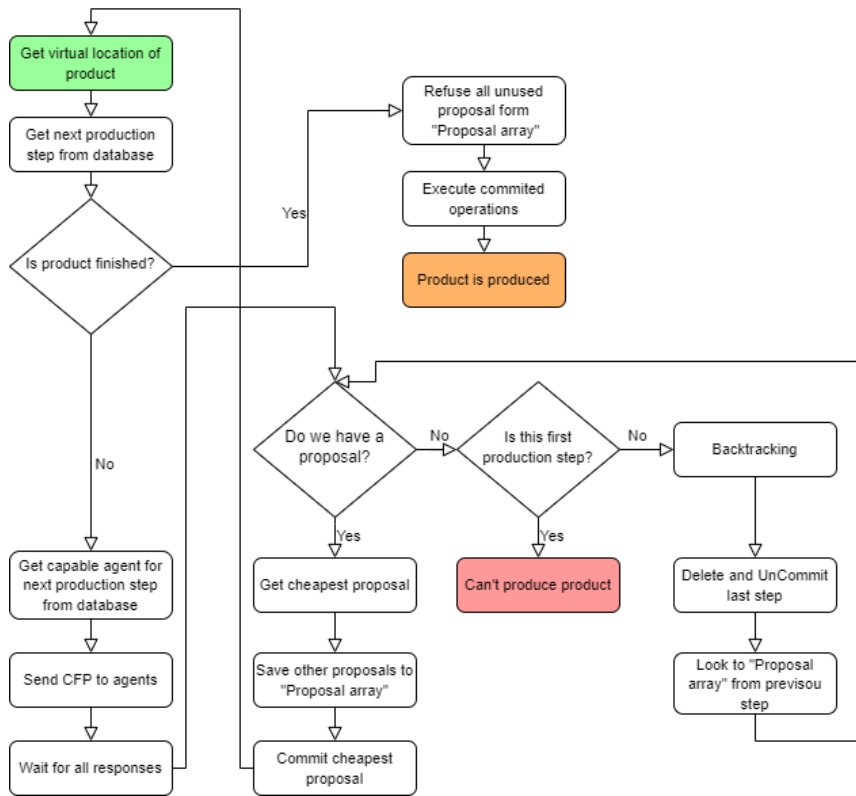


Figure 4.7: Diagram describing backtracking algorithm

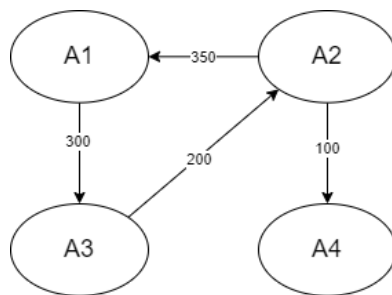


Figure 4.8: Diagram describing transport connection between agents in backtracking example

this agent and the proposal from agent *A1* is stored in the array, as in the previous steps. In the fourth step, we get no proposal because there is no path from agent *A4*. So we go back one step, perform the UnCommit operation and take the proposals from the array in this step. The only proposal is from agent *A1*, so we commit it and continue to step four. Now we get proposals from agents *A2* and *A4*. Agent *A2* has a better price for the operation even with the import of material, we commit it, and thus we have found a solution. Now we start executing the operations one by one.

Agent	chassis	tires	cabin	body	
A1	1500	2600	2000	-	cost of capability [ms]
	1	4	1	0	number of parts [-]
A2	-	2000	-	2000	cost of capability [ms]
	0	4	0	0	number of parts [-]
A3	1800	-	-	2500	cost of capability [ms]
	1	0	0	2	number of parts [-]
A4	-	-	2200	-	cost of capability [ms]
	0	0	1	0	number of parts [-]

Table 4.2: Agent capabilities with their cost and number of components for each agent

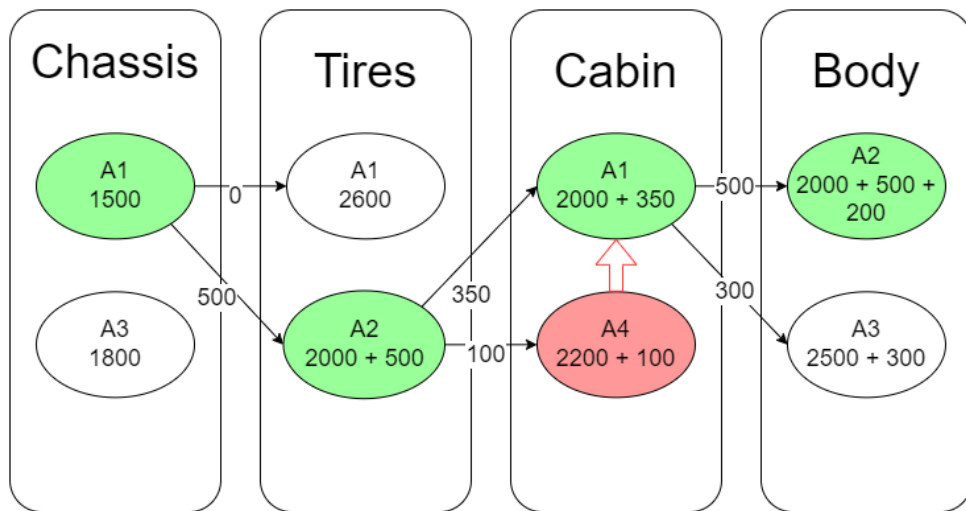


Figure 4.9: Diagram describing flow through the backtracking algorithm on backtracking example. Cost are in ms.

In some cases, this solution could schedule multiple products simultaneously if there are enough components in the system and they do not block each other. This issue could be solved by reimplementing the behavior of the production agent if resources are not reserved by commit, for example, to perform the scheduling of a given step again. At the same time, agents currently do not have an internal operations scheduler, so the total production time would often not match the expected time. Furthermore, there could be a dining philosopher problem. The plan is to develop this algorithm further and solve all these problems.

Chapter 5

Testing and comparison of algorithms

In this chapter, I describe the results of two algorithms on six use cases. The algorithms used for testing are the plan, commit, execute algorithm (section: Description of testing solution for production scheduling, implementation description in section: Agents implementation with product and parts transport) and the chronological backtracking algorithm (section: Chronological backtracking, implementation description in section: Agents implementation with backtracking). The use cases were tested in a simulation environment running on a single computer. The communication platform was running in Docker.

5.1 Use case 1

In this use case, the transport of parts between robots and the determination of prices for individual operations is tested.

5.1.1 Description of the use case

The transport between robots forms a complete graph so that deadlock cannot occur, see Figure 5.1. The Table 5.1 lists the agents' capabilities and their costs. The product manufacturing process corresponds to the operations in

the table from top to bottom, except for the transport parts operation. The Table 5.2 describes the initial magazine status of each agent.

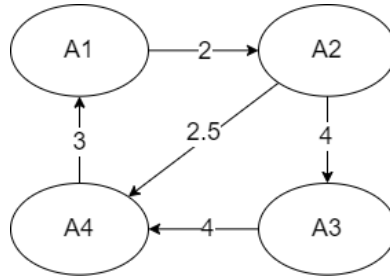


Figure 5.1: Diagram describing the transport between agents for Use case 1. Costs of transitions are in s .

	Parts quantity	A1	A2	A3	A4
Operation 1	1 x Part 1	2	-	-	1.8
Operation 2	1 x Part 2	-	4	4	-
Operation 3	1 x Part 3	-	3	4	3
Operation 4	1 x Part 4	2.5	-	3	-
Transport parts	-	1	1	1	1

Table 5.1: Agent capabilities with their cost [s] and number of parts needed for each operation in Use case 1.

	A1	A2	A3	A4
Part 1	2	0	0	1
Part 2	0	1	2	0
Part 3	0	2	1	0
Part 4	1	0	2	0

Table 5.2: The initial state of the agents' magazines in Use case 1.

5.1.2 Production flow of the first product

Both algorithms had the same output, which is shown in the Figure 5.2. The rows show the responses from each agent for a given operation. The total cost of the operation is written in parentheses, and the nodes show the cost of the operation plus any cost of importing materials. Operations confirmed to agents and then executed are shown in green. The Table 5.3 shows the values obtained during the run of the algorithms. Both algorithms behave identically unless backtracking is required. For this reason, the number of messages received by each agent is identical, and the total time is very similar. The total time describes the time from receiving an order to completing it or detecting a failure.

	Plan, commit execute algorithm							Backtracking algorithm						
Successfully completed	Yes							Yes						
Total time	23.758 [s]							23.774 [s]						
	P	A1	A2	A3	A4	T	D	P	A1	A2	A3	A4	T	D
Call for proposal	0	2	3	4	2	8	0	0	2	3	4	2	8	0
Propose	9	1	2	4	3	0	0	9	1	2	4	3	0	0
Refuse	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Not understood	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reject proposal	0	2	1	3	1	0	0	0	2	1	3	1	0	0
Accept proposal	0	0	2	1	1	2	0	0	0	2	1	1	2	0
Confirm	4	0	1	1	0	0	0	4	0	1	1	0	0	0
Disconfirm	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Request	0	1	3	2	2	3	11	0	1	3	2	2	3	11
Inform	18	0	2	2	1	0	14	18	0	2	2	1	0	14

Table 5.3: Table showing the collected data from the first algorithm run in Use case 1. The bottom part of the table lists the number of messages received by each agent. P = producer agent, T = transport agent, D = Database

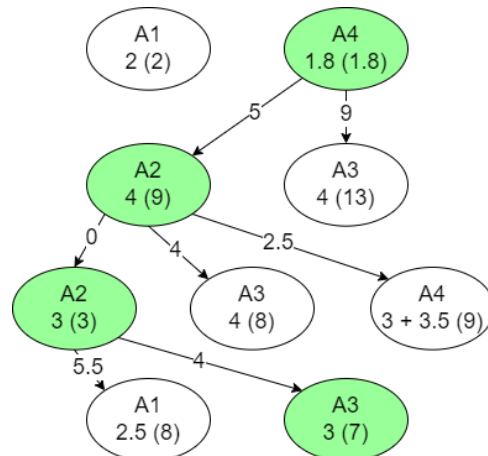


Figure 5.2: Diagram describing first flow through the Use case 1 for both algorithms. Cost are in seconds.

5.1.3 Production flow of the second product

The Figure 5.3 describes the second pass in use case 1. The pass is different because the state of the magazines of each agent has changed and therefore the cost of the operations has changed. The Table 5.4 describes the data collected during the run of the algorithm. The table has been simplified because the number of messages sent is the same for both algorithms.

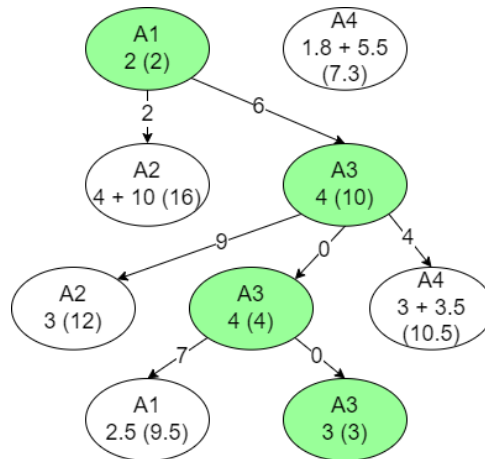


Figure 5.3: Diagram describing second flow through the Use case 1 for both algorithms. Costs are in seconds.

The number of messages sent between agents depends on the flow of the algorithm through the state space. At the same time, there is a dependency between some types of messages. The total number of *Call for proposal* messages is the same as the number of *Propose* and *Refuse* messages. The *Reject proposal* message is sent to reject an operation or import of parts. This message is not sent to the transport agent to avoid communication overhead. The number of *Accept proposal* messages is the same as the number of material or product imports and the number of operations performed. The producing agent gets *Confirm* message for each agent operation, and other agents get *Confirm* message for each material or product import. Agent *A3* has received a confirmation message for the product import in this run. *Request* messages are sent as a request, for example, to perform an operation. *Inform* messages include a change of status on the production line.

	Plan, commit execute algorithm				Backtracking algorithm		
Successfully completed	Yes				Yes		
Total time	21.783 [s]				21.683 [s]		
	P	A1	A2	A3	A4	T	D
Call for proposal	0	3	3	5	2	9	0
Propose	9	2	4	3	4	0	0
Refuse	0	0	0	0	0	0	0
Not understood	0	0	0	0	0	0	0
Reject proposal	0	2	3	2	2	0	0
Accept proposal	0	1	0	3	0	1	0
Confirm	4	0	0	1	0	0	0
Disconfirm	0	0	0	0	0	0	0
Request	0	1	0	3	0	1	13
Inform	18	0	1	2	2	0	9

Table 5.4: Table showing the collected data from the second algorithm run in Use case 1. The bottom part of the table lists the number of messages received by each agent. P = producer agent, T = transport agent, D = Database

5.1.4 Production flow of the third product

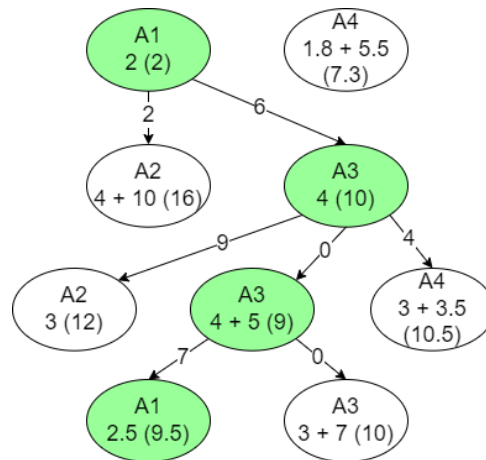


Figure 5.4: Diagram describing third flow through the Use case 1 for both algorithms. Costs are in seconds.

	Plan, commit execute algorithm				Backtracking algorithm		
Successfully completed	Yes				Yes		
Total time	33.188 [s]				33.191 [s]		
	P	A1	A2	A3	A4	T	D
Call for proposal	0	4	4	4	2	10	0
Propose	9	3	5	4	3	0	0
Refuse	0	0	0	0	0	0	0
Not understood	0	0	0	0	0	0	0
Reject proposal	0	2	3	2	2	0	0
Accept proposal	0	2	1	2	0	3	0
Confirm	4	1	1	2	0	0	0
Disconfirm	0	0	0	0	0	0	0
Request	0	2	1	2	0	3	15
Inform	18	2	3	6	2	0	10

Table 5.5: Table showing the collected data from the third algorithm run in Use case 1. The bottom part of the table lists the number of messages received by each agent. P = producer agent, T = transport agent, D = Database

5.1.5 Production flow of the fourth product

When trying to produce a fourth product for which there were no parts in the system, the algorithms terminated at the first operation with the error "Not enough parts for operation in production line".

5.2 Use case 2

This use case tested the transportation of multiple different parts between robots and the pricing of each operation.

5.2.1 Description of the use case

The distribution of agents on the production line and the transport between them is the same as in use case 1, see Figure 5.1. The Table 5.6 lists the agents' capabilities and their costs. The product manufacturing process corresponds

to the operations in the table from top to bottom, except for the transport parts operation. The capabilities and their costs are identical to the use case 1. The change is in the components needed for the operations. Multiple components of one or more types are required. The Table 5.7 describes the initial magazine status of each agent.

	Parts quantity	A1	A2	A3	A4
Operation 1	1 x Part 1	2	-	-	1.8
Operation 2	4 x Part 2, 16 x Part 3	-	4	4	-
Operation 3	8 x Part 3, 1 x Part 4	-	3	4	3
Operation 4	3 x Part 5	2.5	-	3	-
Transport parts	-	1	1	1	1

Table 5.6: Agent capabilities with their cost [ms] and number of parts needed for each operation for Use case 2

	A1	A2	A3	A4
Part 1	1	0	0	2
Part 2	0	6	6	0
Part 3	0	20	30	22
Part 4	0	1	0	2
Part 5	5	0	4	0

Table 5.7: The initial state of the agents' magazines in Use case 2.

5.2.2 Production flow of the first product

The Figure 5.5 describes the first pass in use case 2. The Table 5.8 describes the data collected during the run of the algorithm. The table shows a slight increase in the total number of messages sent compared to the first pass in use case 1. This is due to more queries for importing parts.

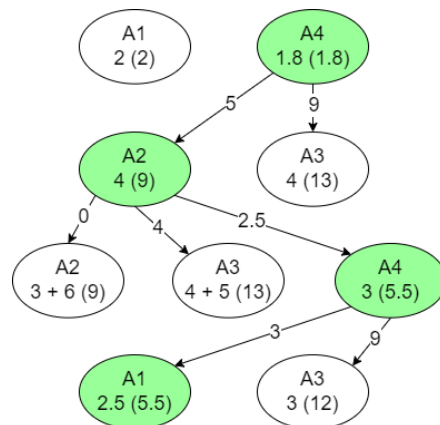


Figure 5.5: Diagram describing first flow through the Use case 2 for both algorithms. Costs are in seconds.

	Plan, commit execute algorithm				Backtracking algorithm		
Successfully completed	Yes				Yes		
Total time	24.828 [s]				24.882 [s]		
	P	A1	A2	A3	A4	T	D
Call for proposal	0	2	3	4	4	10	0
Propose	9	1	4	6	3	0	0
Refuse	0	0	0	0	0	0	0
Not understood	0	0	0	0	0	0	0
Reject proposal	0	1	2	4	2	0	0
Accept proposal	0	1	1	0	2	3	0
Confirm	4	1	1	0	1	0	0
Disconfirm	0	0	0	0	0	0	0
Request	0	2	2	1	3	4	12
Inform	18	2	3	1	2	0	16

Table 5.8: Table showing the collected data from the first algorithm run in Use case 2. The bottom part of the table lists the number of messages received by each agent. P = producer agent, T = transport agent, D = Database

5.2.3 Production flow of the second product

The Figure 5.6 describes the second pass in use case 2. For *operation 2*, *agent 2* returned a significantly higher price compared to *agent A3* because he needed to import 2 types of parts. The Table 5.9 describes the data collected during the run of the algorithm. Again, we see an increase in messages sent between agents, due to more frequent queries about importing parts.

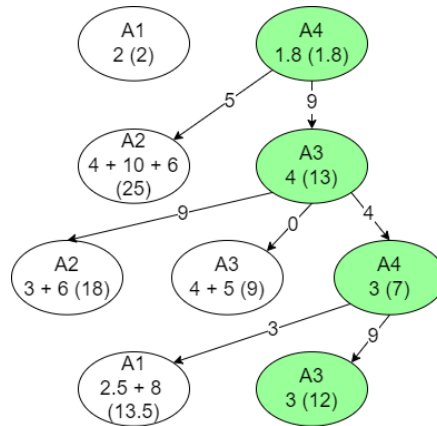


Figure 5.6: Diagram describing second flow through the Use case 2 for both algorithms. Costs are in seconds.

	Plan, commit execute algorithm				Backtracking algorithm		
Successfully completed	Yes				Yes		
Total time	36.482 [s]				36.605 [s]		
	P	A1	A2	A3	A4	T	D
Call for proposal	0	2	3	7	5	14	0
Propose	9	2	8	8	4	0	0
Refuse	0	0	0	0	0	0	0
Not understood	0	0	0	0	0	0	0
Reject proposal	0	2	3	5	3	0	0
Accept proposal	0	0	0	2	2	3	0
Confirm	4	0	0	2	1	0	0
Disconfirm	0	0	0	0	0	0	0
Request	0	0	0	2	2	3	15
Inform	18	1	3	5	2	0	11

Table 5.9: Table showing the collected data from the second algorithm run in Use case 2. The bottom part of the table lists the number of messages received by each agent. P = producer agent, T = transport agent, D = Database

■ 5.3 Use case 3

This use case tested the backtracking algorithm and its ability to use backtrack and find a solution.

■ 5.3.1 Description of the use case

The transport between robots does not form a complete graph, there is an agent from which there is no path back, see Figure 5.7. The Table 5.10 lists the capabilities of the agents and their costs. The product manufacturing process corresponds to the operations in the table from top to bottom, except for the transport parts operation. The Table 5.11 describes the initial inventory state of each agent.

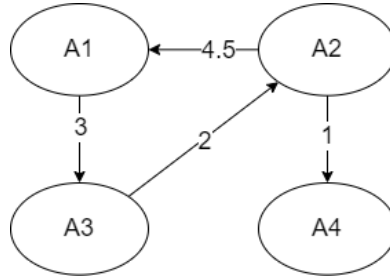


Figure 5.7: Diagram describing the transport between agents for Use case 3. Costs of transitions are in s .

	Parts quantity	A1	A2	A3	A4
Operation 1	1 x Part 1	1.5	-	2.5	-
Operation 2	4 x Part 2, 4 x Part 3	4	2	-	-
Operation 3	2 x Part 4	2	-	-	2.2
Operation 4	1 x Part 5	-	3	4	-
Transport parts	-	1	1	1	1

Table 5.10: Agent capabilities with their cost [s] and number of parts needed for each operation for Use case 3.

	A1	A2	A3	A4
Part 1	2	0	1	0
Part 2	6	6	0	0
Part 3	6	6	0	0
Part 4	4	0	0	2
Part 5	0	1	2	0

Table 5.11: The initial state of the agents' magazines in Use case 3.

5.3.2 Production flow of the first product

The Figure 5.8 shows the first run of the algorithms. Since the cost of agent A_4 was higher, both algorithms successfully found a solution. In this case, backtrack was not needed.

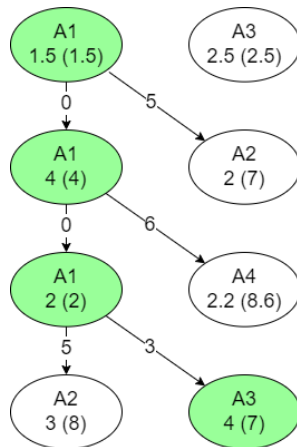


Figure 5.8: Diagram describing first flow through the Use case 3 for both algorithms. Costs are in seconds.

	Plan, commit execute algorithm			Backtracking algorithm			
Successfully completed	Yes			Yes			
Total time	17.668 [s]			17.483 [s]			
	P	A1	A2	A3	A4	T	D
Call for proposal	0	3	2	2	1	4	0
Propose	8	0	2	1	1	0	0
Refuse	0	0	0	0	0	0	0
Not understood	0	0	0	0	0	0	0
Reject proposal	0	0	2	1	1	0	0
Accept proposal	0	3	0	1	0	1	0
Confirm	4	0	0	1	0	0	0
Disconfirm	0	0	0	0	0	0	0
Request	0	4	1	2	1	2	10
Inform	18	0	0	2	0	0	15

Table 5.12: Table showing the collected data from the first algorithm run in Use case 3. The bottom part of the table lists the number of messages received by each agent. P = producer agent, T = transport agent, D = Database

5.3.3 Production flow of the second product

The Figure 5.9 shows the second run of the plan, commit, execute algorithm. In this run, the cost of agent A_4 was lower, and therefore the algorithm ended with the third operation in error. The total time of the algorithm was 14.24 s. Moreover, with the immediate execution of the operations, there is now an unfinished product on the production line.

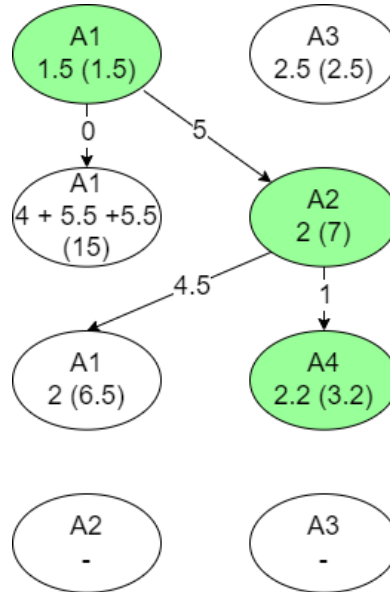


Figure 5.9: Diagram describing second flow through the Use case 3 for plan, commit, execute algorithm. Costs are in seconds.

The Figure 5.10 shows the second run of the backtracking algorithm. The nodes marked in green have executed at the end of the scheduling and the green arrows indicate the passage of the algorithm. The algorithm proceeded identically to the previous algorithm until the third operation. It did not receive any proposal for the fourth operation, so it chose a different agent for the third operation and thus found a solution. Among the messages received in Table 5.13, there are already *Refuse* messages that the *production agent* received in the fourth operation before backtracking occurred.

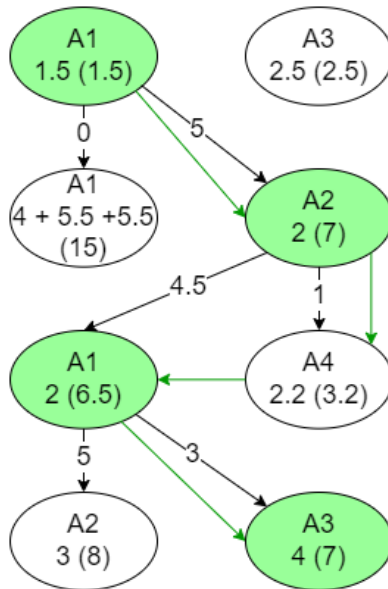


Figure 5.10: Diagram describing second flow through the Use case 3 for backtracking algorithm. Costs are in seconds.

	Backtracking algorithm						
Successfully completed	Yes						
Total time	25.424 [s]						
	P	A1	A2	A3	A4	T	D
Call for proposal	0	3	5	3	1	9	0
Propose	8	3	5	2	1	0	0
Refuse	2	0	0	0	0	0	0
Not understood	0	0	0	0	0	0	0
Reject proposal	0	1	3	1	0	0	0
Accept proposal	0	2	1	1	1	4	0
Confirm	5	1	1	1	1	0	0
Disconfirm	0	0	0	0	0	0	0
Request	0	2	1	1	1	4	14
Inform	22	4	2	2	1	0	14

Table 5.13: Table showing the collected data from the second backtrack algorithm run in Use case 3. The bottom part of the table lists the number of messages received by each agent. P = producer agent, T = transport agent, D = Database

5.3.4 Production flow of the third product

The Figure 5.11 shows the third run of the backtracking algorithm. In this run, the components needed for the third operation were located at agent A_4 . It was not possible to transfer them to another agent, and there was no possibility of coming back with the product. Thus, the algorithm sequentially searched the entire state space and then ended with a parts shortage error. The Table 5.14 shows the number of messages received by each agent. Because there was repeated backtracking, the total number of messages is high compared to previous number of messages.

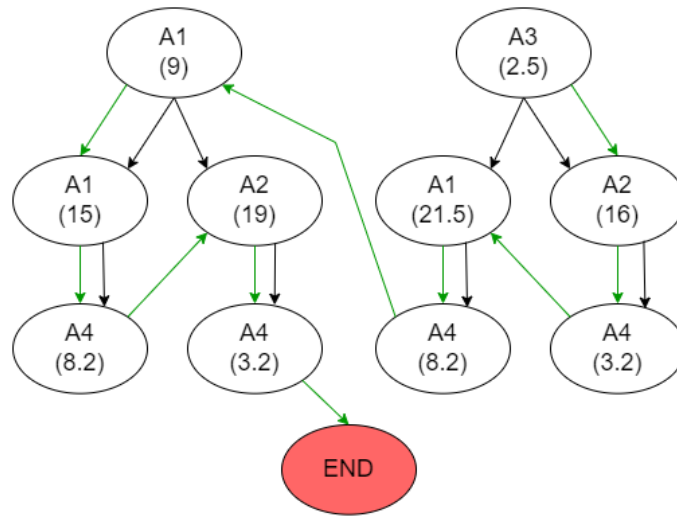


Figure 5.11: Diagram describing third flow through the Use case 3 for backtracking algorithm. Costs are in seconds.

	Backtracking algorithm						
Successfully completed	No						
Total time	7.088 [s]						
	P	A1	A2	A3	A4	T	D
Call for proposal	0	14	12	8	9	34	0
Propose	18	11	16	6	9	0	0
Refuse	12	4	0	0	0	0	0
Not understood	0	0	0	0	0	0	0
Reject proposal	0	0	2	1	1	0	0
Accept proposal	0	10	6	3	4	17	0
Confirm	14	10	10	2	4	0	0
Disconfirm	0	0	0	0	0	0	0
Request	0	11	7	4	5	18	46
Inform	61	9	14	3	4	0	82

Table 5.14: Table showing the collected data from the third backtrack algorithm run in Use case 3. The bottom part of the table lists the number of messages received by each agent. P = producer agent, T = transport agent, D = Database

5.4 Use case 4

This use case tested multi-level and repetitive backtracking.

5.4.1 Description of the use case

The transport between robots does not form a complete graph. The graph is formed by two strongly connected components, see Figure 5.12. The Table 5.15 lists the capabilities of the agents and their costs. The product manufacturing process corresponds to the operations in the table from top to bottom, except for the transport parts operation. The Table 5.16 describes the initial inventory state of each agent.

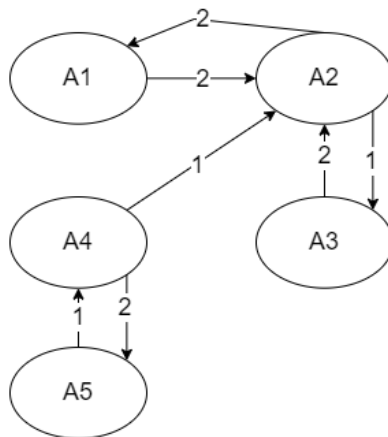


Figure 5.12: Diagram describing the transport between agents for Use case 4. Costs of transitions are in s .

	Parts quantity	A1	A2	A3	A4	A5
Operation 1	1 x Part 1	2	-	-	4	-
Operation 2	1 x Part 2	-	2	8	-	5
Operation 3	1 x Part 3	2	-	2	6	-
Operation 4	1 x Part 4	-	-	-	2	1
Transport parts	-	1	1	1	1	1

Table 5.15: Agent capabilities with their cost [s] and number of parts needed for each operation for Use case 4.

	A1	A2	A3	A4	A5
Part 1	1	0	0	1	0
Part 2	0	0	1	0	1
Part 3	1	0	0	1	0
Part 4	0	0	0	0	1

Table 5.16: The initial state of the agents' magazines in Use case 4.

5.4.2 Production flow of the first product

Figure 5.13 shows the first run of the backtracking algorithm. The nodes marked in green have executed at the end of the scheduling and the green arrows indicate the passage of the algorithm. The plan, commit, execute algorithm performed the first three operations just like the backtracking algorithm and ended with an error. Due to the wrong initial choice of the first operation, the algorithm searched almost the entire state space and only found a solution towards the end. This corresponds to significant communication between agents, see Table 5.17. If the algorithm had been extended with pruning (Graph pruning), the solution would have been found sooner because the state space would have been greatly simplified.

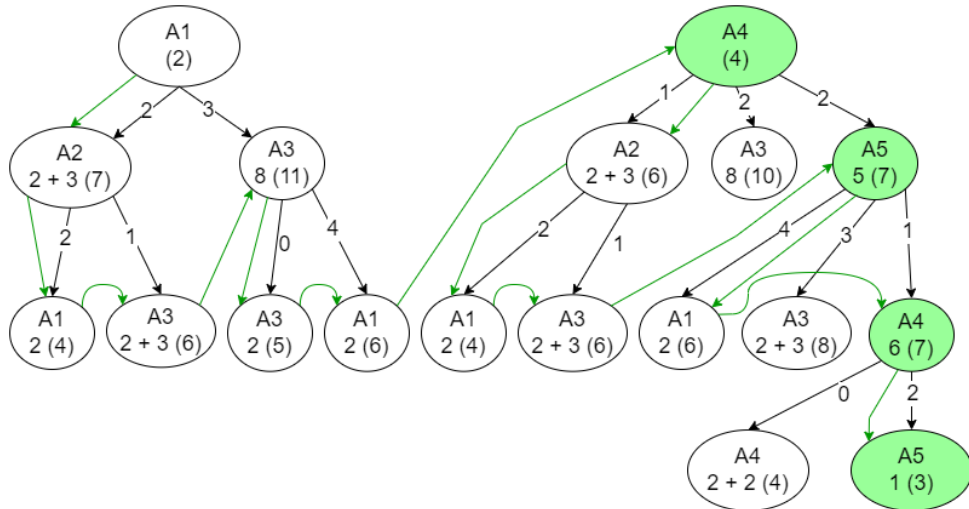


Figure 5.13: Diagram describing first flow through the Use case 4 for backtracking algorithm. Costs are in seconds.

	Plan, commit execute algorithm				Backtracking algorithm			
Successfully completed	No				Yes			
Total time	15.805 [s]				31.345 [s]			
	P	A1	A2	A3	A4	A5	T	D
Call for proposal	0	9	2	8	17	13	45	0
Propose	18	8	6	15	16	13	0	0
Refuse	18	0	0	0	0	0	0	0
Not understood	0	0	0	0	0	0	0	0
Reject proposal	0	4	0	2	2	3	0	0
Accept proposal	0	5	2	6	5	2	17	0
Confirm	15	4	4	8	4	2	0	0
Disconfirm	0	0	0	0	0	0	0	0
Request	0	6	3	7	6	3	18	39
Inform	62	4	6	12	6	4	0	69

Table 5.17: Table showing the collected data from the first backtrack algorithm run in Use case 4. The bottom part of the table lists the number of messages received by each agent. P = producer agent, T = transport agent, D = Database

5.5 Use case 5

This use case tested the problem of component placement between agents. The system has enough components to produce a product, but they are distributed among multiple agents.

5.5.1 Description of the use case

The transport between robots does form a complete graph, and for simplicity, there are three agents in the system, see Figure 5.14. The Table 5.18 lists the capabilities of the agents and their costs. The product manufacturing process corresponds to the operations in the table from top to bottom, except for the transport parts operation. The Table 5.19 describes the initial inventory state of each agent.

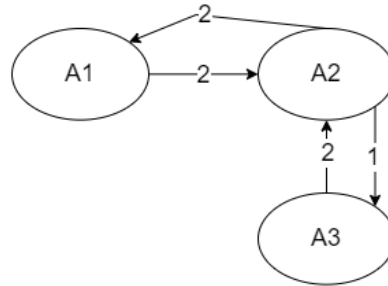


Figure 5.14: Diagram describing the transport between agents for Use case 5. Costs of transitions are in s .

	Parts quantity	A1	A2	A3
Operation 1	2 x Part 1	2	-	-
Operation 2	2 x Part 2, 1 x Part 3	-	2	4
Operation 3	4 x Part 4	2	3	2
Transport parts	-	1	1	1

Table 5.18: Agent capabilities with their cost $[s]$ and number of parts needed for each operation for Use case 5.

	A1	A2	A3
Part 1	3	0	0
Part 2	0	1	1
Part 3	0	1	0
Part 4	2	1	1

Table 5.19: The initial state of the agents' magazines in Use case 5.

5.5.2 Production flow of the first product

The Figure 5.15 shows the flow of the plan, commit, execute algorithm. The first operation is executed, the second operation is executed, and then agent *A1* must import two components, and the remaining agents must import three parts. There is no one in the system, and the run ends with an error of "Not enough parts for operation in production line". As mentioned in Agents implementation with product and parts transport, this problem could be solved. The total time of production flow is 11.086 [s], and the product is unfinished on the production line.

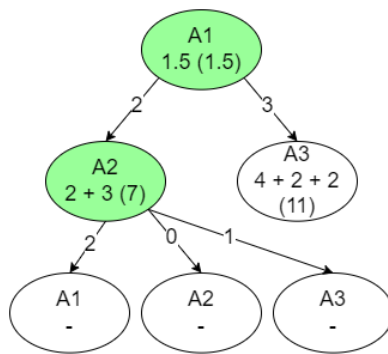


Figure 5.15: Diagram describing first flow through the Use case 5 for Plan, commit, execute algorithm. Costs are in seconds.

The figure shows the flow of the backtracking algorithm. The green arrows describe its behavior. No solution is found, and the algorithm terminates with the same error. The total time of the production flow is 2.773 [s] because there was no start of the product production.

This problem is related to the implementation of each agent.

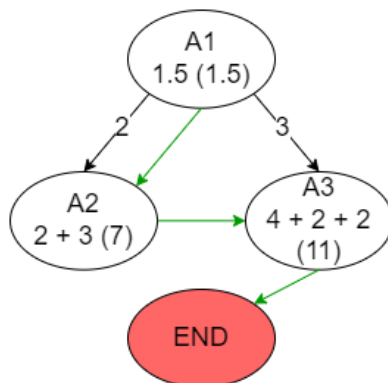


Figure 5.16: Diagram describing first flow through the Use case 5 for Backtracking algorithm. Costs are in seconds.

5.6 Use case 6

This use case tested the production of a more complex product on a virtual twin of the montrac production line Description of the Montrac test line. The production process is similar to the planned production of an RC car.

5.6.1 Description of the use case

The transport between robots corresponds to the assembly line layout, see Figure 5.17. The Table 5.20 lists the capabilities of the agents and their costs. The product manufacturing process corresponds to the operations in the table from top to bottom, except for the transport parts operation. The Table 5.21 describes the initial inventory state of each agent.

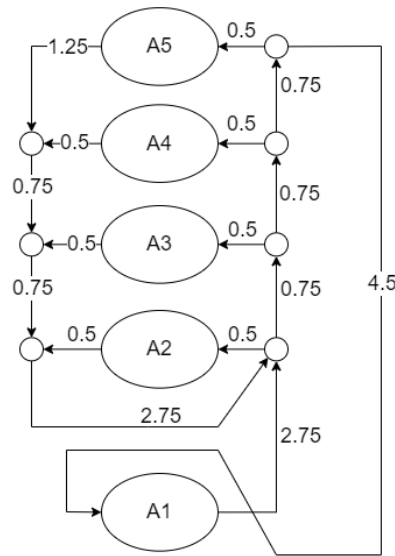


Figure 5.17: Diagram describing the transport between agents for Use case 6. Costs of transitions are in s .

	Parts quantity	A1	A2	A3	A4	A5
Operation chassis	1 x A1	-	-	6	5	4
Operation E1	1 x E1	-	-	6	5	4
Operation M1	1 x M1	-	-	6	5	4
Operation E2	1 x E2	-	-	6	5	4
Operation M2	1 x M2	-	-	6	5	4
Operation screw 1	8 x C1	-	4	8	-	-
Operation tire L	2 x Tire	7	-	-	-	15
Operation tire R	2 x Tire	7	-	-	-	15
Operation screw 2	4 x C1	-	2	4	-	-
Operation body	1 x Body	4	-	-	7	8
Operation screw 3	8 x C1	-	4	8	-	-
Transport parts	-	1	1	1	1	1

Table 5.20: Agent capabilities with their cost [s] and number of parts needed for each operation for Use case 6.

	A1	A2	A3	A4	A5
A1	0	0	0	4	3
E1	0	0	3	2	2
M1	0	0	3	1	3
E2	0	0	4	0	3
M2	0	0	2	3	2
C1	0	90	50	0	0
Tire	18	0	0	0	10
Body	4	0	0	2	1

Table 5.21: The initial state of the agents' magazines in Use case 6.

5.6.2 Production flow for five product

The Table 5.22 describes the production runs for five products for both algorithms. Both algorithms successfully completed each production and the total times are almost identical. The behavior of both algorithms is also identical.

The specific messages received by the agents are described in the tables (B.2,B.3,B.4,B.5,B.6) in the appendix Additional tables. The tables are very similar to the previous tables showing received messages and take up a lot of space, so they are placed in the appendix.

Plan, commit, execute algorithm	Successfully completed	First product	Second product	Third product	Fourth product	Fifth product
	Total time [s]	88.318	87.675	105.982	99.347	123.769
Backtracking algorithm	Progress of the product production with the cost of the each operation	R5-(4), R5-(4), R5-(4), R5-(4), R5-(4), R2-(10), R1-(17), R1-(7), R2-(5.25), R4-(12.25), R2-(9.25)	R5-(4), R5-(4), R5-(4), R5-(4), R5-(4), R2-(10), R1-(17), R1-(7), R2-(5.25), R4-(12.25), R2-(9.25)	R5-(4), R5-(11.75), R5-(4), R5-(4), R5-(11.75), R2-(10), R1-(17), R1-(7), R2-(5.25), R5-(14), R2-(10)	R4-(5), R4-(5), R4-(5), R3-(12), R3-(6), R3-(8), R1-(17.75), R1-(7), R2-(5.25), R1-(14), R2-(7.25)	R4-(5), R4-(5), R3-(12), R3-(6), R4-(11), R2-(9.25), R1-(17), R1-(20.25), R2-(5.25), R1-(14), R3-(12)
	Total number of messages sent between agents	300	288	344	340	380
Backtracking algorithm	Successfully completed	Yes	Yes	Yes	Yes	Yes
	Total time [s]	88.247	87.819	105.937	99.181	123.818
Backtracking algorithm	Progress of the product production with the cost of the each operation	R5-(4), R5-(4), R5-(4), R5-(4), R5-(4), R2-(10), R1-(17), R1-(7), R2-(5.25), R4-(12.25), R2-(9.25)	R5-(4), R5-(4), R5-(4), R5-(4), R5-(4), R2-(10), R1-(17), R1-(7), R2-(5.25), R4-(12.25), R2-(9.25)	R5-(4), R5-(11.75), R5-(4), R5-(4), R5-(11.75), R2-(10), R1-(17), R1-(7), R2-(5.25), R5-(14), R2-(10)	R4-(5), R4-(5), R4-(5), R3-(12), R3-(6), R3-(8), R1-(17.75), R1-(7), R2-(5.25), R1-(14), R2-(7.25)	R4-(5), R4-(5), R3-(12), R3-(6), R4-(11), R2-(9.25), R1-(17), R1-(20.25), R2-(5.25), R1-(14), R3-(12)
	Total number of messages sent between agents	300	288	344	340	380

Table 5.22: The table describes the individual product production flows for use case 6.

■ 5.7 Discussion

The previous examples (Use case 1 - all production flows, Use case 2 - all production flows, Use case 3 - first production flow, and Use case 6 - all production flows) show that both algorithms behave the same when backtracking is not needed. The number of messages sent between agents is exactly the same, and the total time differs by a few tenths of a second, which is negligible deviation. If the transport graph is incomplete, backtracking may be needed (Use case 3, Use case 4). In this case, the original algorithm ends with a work in progress on the production line and does not complete production. The backtracking algorithm either finds a solution if one exists and produces the product or does not find a solution and does not start producing the product. This case can be seen in the Use case 5 (where no solution to the problem), where the total time for the backtracking algorithm is shorter because it has not started production. The backtracking algorithm removes the original algorithm's main weakness, i.e., finding a solution if one exists in incomplete transport graphs.

Testing of the algorithms was performed in the test environment, which in Use case 6 is a one-to-one copy of the Motrac production line. Unfortunately, the development of the multiagent platform was delayed, and therefore it was impossible to test the backtracking algorithm on a real Montrac line. For this reason, a comparison between the multiagent approach and the central approach, which runs on the physical production line, cannot be made. The approaches are sufficiently different that a comparison cannot be made based only on the results from the test environment.



Chapter 6

Conclusion

In chapter three, chronological backtracking was proposed as a suitable algorithm to overcome the weaknesses of the original solution that uses multiagent systems. Chapter four describes an implementation based on the multiagent platform under development, which allows for easy sustainability and scalability of the solution. In chapter five, the functionality of the proposed algorithm has been tested and compared with the original algorithm.

The proposed algorithm has identical behavior when backtracking is not required. Otherwise, it searches the state space and finds the solution if one exists. Chronological backtracking behaves as well or better than the original algorithm and is, therefore, a suitable replacement.

Future work on the new algorithm will involve the implementation of pruning to speed up the solution search. Furthermore, the proposed algorithm does not address the solution's optimality; therefore, we will look for ways to achieve some optimality of the solution.

Now only the production of one product running simultaneously has been tested. There is a need to test the algorithm's behavior during parallel scheduling and production of multiple products and modify the algorithm accordingly to work in this mode. The necessary modifications will include, for example: changing the code to reschedule the production of a product if an order confirmation is rejected and solving the dining philosophers problem.



Appendix A

Bibliography

- [1] Gerhard Weiss. *Multiagent systems*. MIT press, 2 edition, 2013.
- [2] Ondřej Šebek. Testbed multiagent platform manual. https://github.com/testbedCIIRC/braine-multiagent-platform/blob/master/doc/TestbedMultiAgentPlatform_Manual.docx, 2021.
- [3] Ali Dorri, Salil S Kanhere, and Raja Jurdak. Multi-agent systems: A survey. *Ieee Access*, 6:28573–28593, 2018.
- [4] Fei Chen, Wei Ren, et al. On the control of multi-agent systems: A survey. *Foundations and Trends® in Systems and Control*, 6(4):339–499, 2019.
- [5] Javier Bajo Pérez, Fernando de la Prieta Pintado, Juan Manuel Corchado Rodríguez, Sara Rodríguez González, et al. A low-level resource allocation in an agent-based cloud computing platform. 2016.
- [6] Jelena Fiosina and Maksims Fiosins. Density-based clustering in cloud-oriented collaborative multi-agent systems. In *International Conference on Hybrid Artificial Intelligence Systems*, pages 639–648. Springer, 2013.
- [7] Maíra Gatti, Paulo Cavalin, Samuel Barbosa Neto, Claudio Pinhanez, Cícero dos Santos, Daniel Gribel, and Ana Paula Appel. Large-scale multi-agent-based modeling and simulation of microblogging-based online social network. In *International Workshop on Multi-Agent Systems and Agent-Based Simulation*, pages 17–33. Springer, 2013.
- [8] Leila Mechtri, Fatiha Djemili Tolba, and Salim Ghanemi. Masid: multi-agent system for intrusion detection in manet. In *2012 Ninth International Conference on Information Technology-New Generations*, pages 65–70. IEEE, 2012.

- [9] Gianni Di Caro, Marco Dorigo, et al. An adaptive multi-agent routing algorithm inspired by ants behavior. In *Proceedings of PART98-5th Annual Australasian Conference on Parallel and Real-Time Systems*, pages 261–272, 1998.
- [10] Angel Soriano, Enrique J Bernabeu, Angel Valera, and Marina Vallés. Multi-agent systems platform for mobile robots collision avoidance. In *International Conference on Practical Applications of Agents and Multi-Agent Systems*, pages 320–323. Springer, 2013.
- [11] Dirk Helbing. Agent-based modeling. In *Social self-organization*, pages 25–70. Springer, 2012.
- [12] Dominik Böhnlein, Katharina Schweiger, and Axel Tuma. Multi-agent-based transport planning in the newspaper industry. *International Journal of Production Economics*, 131(1):146–157, 2011. Innsbruck 2008.
- [13] Chafik Abid, Sophie D’amours, and Benoit Montreuil. Collaborative order management in distributed manufacturing. *International journal of production research*, 42(2):283–302, 2004.
- [14] Maria Caridi, Roberto Cigolini*, and D De Marco. Improving supply-chain collaboration by linking intelligent agents to cpfr. *International journal of production research*, 43(20):4191–4218, 2005.
- [15] Roberto Dominguez and Salvatore Cannella. Insights on multi-agent systems applications for supply chain management. *Sustainability*, 12(5):1935, 2020.
- [16] Michael Wooldridge. *An introduction to multiagent systems*. John wiley & sons, 2 edition, 2009.
- [17] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Upper Saddle River, NJ, USA:, 2 edition, 2003.
- [18] Michael Wooldridge and Nicholas R Jennings. Intelligent agents: Theory and practice. *The knowledge engineering review*, 10(2):115–152, 1995.
- [19] Stuart J Russell. Rationality and intelligence. *Artificial intelligence*, 94(1-2):57–77, 1997.
- [20] John Langshaw Austin. *How to do things with words*. Oxford university press, 1975.
- [21] Speech acts category. <https://carla.umn.edu/speechacts/descriptions.html>. Accessed: 2021-12-19.
- [22] Daniel Vanderveken and Susumu Kubo. *Essays in speech act theory*, volume 77. John Benjamins Publishing, 2002.

- [23] Rodger Kibble. Speech acts, commitment and multi-agent communication. *Computational & mathematical organization theory*, 12(2):127–145, 2006.
- [24] Tim Finin, Richard Fritzson, Don McKay, and Robin McEntire. Kqml as an agent communication language. In *Proceedings of the third international conference on Information and knowledge management*, pages 456–463, 1994.
- [25] Philip R Cohen and Hector J Levesque. Communicative actions for artificial agents. In *ICMAS*, volume 95, pages 65–72. Citeseer, 1995.
- [26] Fipa communicative act library specification. http://www.fipa.org/specs/fipa00037/SC00037J.html#_Toc26729686. Accessed: 2021-12-21, Specification dated 2002-12-03.
- [27] Yoav Shoham and Kevin Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2008.
- [28] Computational game theory, coalition games and the core, lecture from tomáš kroupa. https://cw.fel.cvut.cz/wiki/_media/courses/cgt/cg01_lectures.pdf. Accessed: 2022-02-03, Specification dated 2002-12-03.
- [29] Kuka kr manual. http://www.wtech.com.tw/public/download/manual/kuka/KUKA%20KR%206%2010_AGILUS.pdf. Accessed: 2023-01-03, Specification dated 2015-03-25.
- [30] Kuka lbr iiwa manual. https://www.oir.caltech.edu/twiki_oir/pub/Palomar/ZTF/KUKARoboticArmMaterial/Specz_LBR_iiwa_en.pdf. Accessed: 2023-01-03, Specification dated 2015-01-28.
- [31] Kuka cybertech kr 8 r1620 manual. http://www.wtech.com.tw/public/download/manual/kuka/KUKA%20CYBERTECH_nano_en.pdf. Accessed: 2023-01-03, Specification dated 2016-07-25.
- [32] Petr Novák, Jiří Vyskočil, and Petr Kadera. Plan executor mes: manufacturing execution system combined with a planner for industry 4.0 production systems. In *International Conference on Industrial Applications of Holonic and Multi-Agent Systems*, pages 67–80. Springer, 2019.
- [33] Petr Kadera and Pavel Tichý. Plan, commit, execute protocol in multi-agent systems. In *International Conference on Industrial Applications of Holonic and Multi-Agent Systems*, pages 155–164. Springer, 2009.
- [34] Marc-André Dittrich and Silas Fohlmeister. Cooperative multi-agent system for production control using reinforcement learning. *CIRP Annals*, 69(1):389–392, 2020.

- [35] Georg Egger, Dmitry Chaltsev, Andrea Giusti, and Dominik T Matt. A deployment-friendly decentralized scheduling approach for cooperative multi-agent systems in production systems. *Procedia Manufacturing*, 52:127–132, 2020.
- [36] Dijkstra shortest path algorithm. <https://www.baeldung.com/java-dijkstra>. Accessed: 2022-10-31, Specification dated 2022-03-29.
- [37] Branislav Bošanský. Normal form games. https://cw.fel.cvut.cz/wiki/_media/courses/cgt/cgt_102_nfgs_2022.pdf, 2022.
- [38] Branislav Bošanský. Solving normal-form games. https://cw.fel.cvut.cz/wiki/_media/courses/cgt/cgt_103_solvingnfgs_2022.pdf, 2022.
- [39] Branislav Bošanský. Extensive-form games. https://cw.fel.cvut.cz/wiki/_media/courses/cgt/cgt_104_efgs_2022.pdf, 2022.
- [40] Branislav Bošanský. Solving extensive-form games. https://cw.fel.cvut.cz/wiki/_media/courses/cgt/cgt_105_solvingefgs_2022.pdf, 2022.
- [41] Michal Jakob. Auctions. https://cw.fel.cvut.cz/wiki/_media/courses/cgt/cgt2022-auctions-1.pdf, 2022.
- [42] Iso ts 15066. <https://www.iso.org/standard/62996.html>. Accessed: 2022-12-28.
- [43] En iso 12100. <https://www.iso.org/standard/51528.html>. Accessed: 2022-12-28.
- [44] En iso 13849-1. <https://www.iso.org/standard/69883.html>. Accessed: 2022-12-28.
- [45] En iso 13850. <https://www.iso.org/standard/59970.html>. Accessed: 2022-12-28.
- [46] En iso 13855. <https://www.iso.org/standard/42845.html>. Accessed: 2022-12-28.
- [47] En iso 13857. <https://www.iso.org/standard/69569.html>. Accessed: 2022-12-28.
- [48] En iso 10218-2. <https://www.iso.org/standard/41571.html>. Accessed: 2022-12-28.

Appendix B

Additional tables

Name	date of release	Description	Accessible at
ISO TS 15066	2016	Robots and robotic devices -Collaborative robots	[42]
EN ISO 12100	2010	Safety of machinery - General principles for design, risk assessment and risk reduction	[43]
EN ISO 13849-1	2017	Safety of machinery - Safety components of control systems - Part 1: General principles for design	[44]
EN ISO 13850	2017	Safety of machinery -Emergency stop function -Design principles	[45]
EN ISO 13855	2010	Safety of machinery - Positioning of safeguards with respect to the approach speeds of parts of the human body	[46]
EN ISO 13857	2020	Safety of machinery - Safety distances to prevent hazard zones being reached by upper and lower limbs	[47]
EN ISO 10218-2	2011	Robots and robotic devices - Safety requirements for industrial robots - Part 2: Robot systems and integration	[48]

Table B.1: The table shows the list of ISO standards that were used to create the safety on the Montrac line

	P	A1	A2	A3	A4	A5	T	D
Call for proposal	0	3	3	9	7	10	24	0
Propose	28	2	3	10	8	5	0	0
Refuse	0	0	0	0	0	0	0	0
Not understood	0	0	0	0	0	0	0	0
Reject proposal	0	1	0	9	6	5	0	0
Accept proposal	0	2	3	0	1	5	5	0
Confirm	11	1	3	0	1	0	0	0
Disconfirm	0	0	0	0	0	0	0	0
Request	0	3	4	1	2	6	6	26
Inform	46	2	6	1	3	0	0	29

Table B.2: Table showing the messages received by each agent of the first backtracking algorithm run in use case 6. The number of messages is the same for both algorithms. P = producer agent, T = transport agent, D = Database

	P	A1	A2	A3	A4	A5	T	D
Call for proposal	0	3	3	9	7	10	24	0
Propose	28	2	3	10	8	5	0	0
Refuse	0	0	0	0	0	0	0	0
Not understood	0	0	0	0	0	0	0	0
Reject proposal	0	1	0	9	6	5	0	0
Accept proposal	0	2	3	0	1	5	5	0
Confirm	11	1	3	0	1	0	0	0
Disconfirm	0	0	0	0	0	0	0	0
Request	0	2	3	0	1	5	5	26
Inform	46	2	6	1	3	0	0	23

Table B.3: Table showing the messages received by each agent of the second backtracking algorithm run in use case 6. The number of messages is the same for both algorithms. P = producer agent, T = transport agent, D = Database

	P	A1	A2	A3	A4	A5	T	D
Call for proposal	0	4	3	11	9	11	30	0
Propose	28	3	3	12	12	10	0	0
Refuse	0	0	0	0	0	0	0	0
Not understood	0	0	0	0	0	0	0	0
Reject proposal	0	2	0	9	9	5	0	0
Accept proposal	0	2	3	2	0	6	7	0
Confirm	11	1	3	2	0	3	0	0
Disconfirm	0	0	0	0	0	0	0	0
Request	0	2	3	2	0	6	7	29
Inform	46	2	6	5	2	8	0	25

Table B.4: Table showing the messages received by each agent of the third backtracking algorithm run in use case 6. The number of messages is the same for both algorithms. P = producer agent, T = transport agent, D = Database

	P	A1	A2	A3	A4	A5	T	D
Call for proposal	0	5	3	13	11	8	31	0
Propose	28	4	3	11	9	16	0	0
Refuse	0	0	0	0	0	0	0	0
Not understood	0	0	0	0	0	0	0	0
Reject proposal	0	2	1	10	8	8	0	0
Accept proposal	0	3	2	3	3	0	5	0
Confirm	11	2	2	1	0	0	0	0
Disconfirm	0	0	0	0	0	0	0	0
Request	0	3	2	3	3	0	5	33
Inform	46	4	4	3	2	6	0	23

Table B.5: Table showing the messages received by each agent of the fourth backtracking algorithm run in use case 6. The number of messages is the same for both algorithms. P = producer agent, T = transport agent, D = Database

	P	A1	A2	A3	A4	A5	T	D
Call for proposal	0	5	3	14	11	9	34	0
Propose	28	5	4	13	11	15	0	0
Refuse	0	0	0	0	0	0	0	0
Not understood	0	0	0	0	0	0	0	0
Reject proposal	0	2	1	11	8	8	0	0
Accept proposal	0	3	2	3	3	1	8	0
Confirm	11	3	2	2	1	1	0	0
Disconfirm	0	0	0	0	0	0	0	0
Request	0	3	2	3	3	1	8	37
Inform	46	7	5	6	5	8	0	24

Table B.6: Table showing the messages received by each agent of the fifth backtracking algorithm run in use case 6. The number of messages is the same for both algorithms. P = producer agent, T = transport agent, D = Database