Planning of fault-tolerant solutions of operations research problems and multi-level heuristic scheduling

DOI:10.18136/PE.2021.783

A thesis submitted for the degree of Doctor of Philosophy

by: Tibor Dulai

Supervisor: Prof. György Dósa

University of Pannonia
Faculty of Information Technology
Doctoral School of Information Science

2021

Planning of fault-tolerant solutions of operations research problems and multi-level heuristic scheduling

Thesis for obtaining a PhD degree in the Doctoral School of Information Science and Technology of the University of Pannonia

in the branch of Information Sciences

Written by Tibor Dulai

Supervisor: Prof. György Dósa

propose acceptance (yes / no)

.....
supervisor

As reviewer, I propose acceptance of the thesis:

Name of Reviewer: ..... (yes / no)

.....
reviewer

Name of Reviewer: ..... (yes / no)

.....
reviewer

The PhD-candidate has achieved .....% at the public discussion.

Veszprém, ...
Chairman of the Committee

The grade of the PhD Diploma ..... (..... %)
Veszprém,

.....
Chairman of UDHC

Tartalmi kivonat

Termelési folyamatok egy fontos aspektusa az erőforrás-allokáció és az ütemezés, költségminimalizálás szempontjából is fontos részcél a termelési folyamatok lehető legrövidebb átfutása. A rövidebb termelési folyamat általában kisebb környezeti terhelést is jelent. Egy másik lényeges területe a hatékony vállalati működésnek az, hogy miképpen reagál az esetlegesen előforduló meghibásodásokra. Amennyiben a rendszer már eleve hibatűrő módon volt tervezve, egy fellépő hiba elhárítása és a kívánt működés visszaállítása gyorsabban és olcsóbban kivitelezhető, mint ilyen tulajdonságú rendszer hiányában.

A disszertációm három fő részből áll. Az első két nagy egység a termeléstervezés és az ütemezés témaköréhez tartozik. A harmadik rész egy kiszállítási problémák hibatűrő tervezését és végrehajtását lehetővé tevő módszert mutat be.

Az ütemezésről szóló fejezetekhez használt problémamodell hipotetikus. Úgy lett megalkotva, hogy kihívásokat tartalmazó, de mégis egyszerűen áttekinthető feladatot írjon le. A munkafolyamatok feladatai közt jól definiált sorrendiség van, ugyanakkor a folyamatok összeszerelési feladatot is tartalmaznak. A modell tömegtermelést modellez, erőforrásai függetlenek.

Az első nagy egység egy heurisztikus algoritmust mutat be az ütemezési feladat megoldására. Alapötlete az eredeti probléma felosztása kisebb részproblémákra. A megoldás során olyan lépéseket használtam, mint néhány egyszerűsítő feltétel bevezetése, egy preemptív megoldás kiszámolása, kerekítés, majd a talált megoldás javítása. Ugyanaz a kiindulási probléma egy egzakt módszerrel is megoldásra került. A két módszer által kapott eredményeket összehasonlítottam, valamint azt vizsgáltam, hogy a heurisztikus algoritmus milyen mértékben képes segíteni az egzakt módszert.

A második nagyobb témakör egy, az első nagy rész tömegtermelésre felírt ütemezési feladatának szállítási feladatokkal és szállítóeszközökkel kibővített változatának megoldására fejlesztett genetikus algoritmust mutat be. Az algoritmus az erőforrásokhoz rendelt rendezett feladatlistákat használja kódolásként. A feladat hatékony megoldására genetikus operátorokat fejlesztettem és számos algoritmus-specifikus paramétert határoztam meg. Az algoritmust több irodalmi feladaton is teszteltem.

Végül egy tetszőleges, időablak nélküli kiszállítási probléma megoldásának hibatűrő kiegészítése kerül bemutatásra. Ez a módszer a megoldás minden járműve számára egy olyan bejárási irányt javasol azok körútjain, amely egy esetleges járműmeghibásodás esetén a kisegítés extra költségét átlagosan minimalizálja amellett, hogy meghibásodás hiányában az útvonalakat nem változtatja meg. A fejezet a segítséget jelentő tevékenység javasolt végrehajtását is tartalmazza. Az algoritmus hatékonyságát mind egy egyszerű esettanulmányon, mind a népszerű Solomon teszteken megvizsgáltam.

Abstract

In production planning, resource allocation and scheduling are significant problems. When the production process with the shortest makespan is found, it helps to minimize the production costs. The shorter production also has a lower impact on the environment. Another important aspect that influences the efficiency of a company is its reaction to faults. If a system was designed to be fault-tolerant, eliminating the fault and the reconstruction of the intended operation could happen more quickly and cheaply than without this type of design.

The dissertation is composed of three main parts. The first two units belong to the topics of production planning and scheduling. The final part presents a fault-tolerant planning and execution method for vehicle routing problems.

The model of the scheduling problem is hypothetical. It was constructed to be challenging but easily perspicuous. There are well-defined precedence constraints between the operations of each workflow/job; moreover, each job has an assembly operation, too. The considered problem models mass production, where the resources are unrelated machines.

First, a heuristic algorithm is proposed for solving the problem. The basic idea behind this is the separation of the original problem into smaller sub-problems. The algorithm applies different steps/tricks, like some simplifications, creation of a preemptive solution, rounding, and improvement of the solution. The same problem was also solved by an exact method. The results of the two methods are compared, and it is investigated how the heuristic algorithm can support the exact method.

Second, a genetic algorithm is presented to solve an extended variant of the previous part's scheduling problem. Here, the original problem for mass production is extended by transportation operations and transportation devices. The algorithm codes a solution into ordered operation sequences that are assigned to machines. I have developed genetic operators and determined several algorithm-related parameters to solve the problem efficiently. The algorithm also was tested on several benchmarks from the scientific literature.

Finally, a fault-tolerant extension is introduced for a solution to a general vehicle routing problem without time windows. This method proposes an execution direction for all the vehicles on their circular routes that minimizes the extra cost of a helper action for a broken-down vehicle on average but does not modify the routes unless vehicle breakdown happens. Moreover, the suggested execution of the helper action is also described. The algorithm's efficiency is analyzed on both a simple case study and on common benchmarks - the Solomon instances - too.

Auszug

In der Produktionsplanung sind Ressourcenzuweisung und -planung von besonderen Bedeutung. Wird der Produktionsprozess mit dem kürzesten Durchlauf gefunden, hilft dies, die Produktionskosten zu minimieren. Eine kürzere Produktion hat auch geringere Auswirkungen auf die Umwelt. Ein weiterer wichtiger Aspekt, der die Effizienz eines Unternehmens beeinflusst, ist die Reaktion auf Fehler. Wenn ein System fehlertolerant ausgelegt wurde, kann die Beseitigung des Fehlers und die Rekonstruktion des Betriebes schneller und kostengünstiger erfolgen, als ohne diese Art der Konstruktion.

Die Dissertation besteht aus drei Hauptteilen. Die ersten beiden Einheiten gehören zu den Themen Produktionsplanung und -terminierung. Der letzte Teil enthält eine fehlertolerante Planungs- und Ausführungsmethode für Tourenplanungsprobleme.

Das Modell des Planungsproblems ist hypothetisch. Es wurde so konstruiert, dass es herausfordernd, aber leicht verständlich ist. Es gibt genau definierte Prioritätsbeschränkungen zwischen den Operationen jedes Workflows / Jobs. Darüber hinaus verfügt jeder Job auch über eine Montageaufgabe. Das betrachtete Problem modelliert die Massenproduktion, bei der die Ressourcen unabhängig sind.

Das erste große Thema zeigt uns einen heuristischen Algorithmus zur Lösung des Problems. Die Grundidee dahinter ist die Aufteilung des ursprünglichen Problems in kleinere Teilprobleme. Der Algorithmus verwendet verschiedene Schritte / Tricks, wie z. B. einige Vereinfachungen, die Erstellung einer vorbeugenden Lösung, die Rundung und zuletzt die Verbesserung der Lösung. Das gleiche Problem wurde auch durch eine exakte Methode gelöst. Die Ergebnisse der beiden Methoden werden verglichen und untersucht, wie der heuristische Algorithmus die exakte Methode unterstützen kann.

Zweitens wird ein genetischer Algorithmus vorgestellt, um eine erweiterte Variante des Planungsproblems des vorherigen Teils zu lösen. Hier wird das ursprüngliche Problem der Massenproduktion durch Transportvorgänge und Transportvorrichtungen erweitert. Der Algorithmus codiert eine Lösung in geordnete Betriebssequenzen, die den Maschinen zugewiesen sind. Ich habe genetische Operatoren entwickelt und verschiedene algorithmische Parametern bestimmt, um das Problem effizient lösen zu können. Der Algorithmus wurde auch an mehreren Benchmarks aus der wissenschaftlichen Literatur getestet.

Schließlich wird eine fehlertolerante Erweiterung zur Lösung eines allgemeinen Tourenplanungsproblems ohne Zeitfenster eingeführt. Diese Methode schlägt eine Ausführungsrichtung für alle Fahrzeuge auf ihren Rundstrecken vor, die die zusätzlichen Kosten einer Hilfsaktion für ein kaputtes Fahrzeug im Durchschnitt minimiert,

die Routen jedoch nicht ändert, wenn eine Fahrzeugpanne auftritt. Darüber hinaus wird auch die vorgeschlagene Ausführung der Hilfsaktion beschrieben. Die Effizienz des Algorithmus wird sowohl anhand einer einfachen Fallstudie als auch anhand gängiger Benchmarks - der Solomon-Instanzen - analysiert.

Acknowledgements

I would like to thank my consultant Prof. György Dósa and my "not official" consultants Prof. Katalin Hangos and Dr. Ágnes Werner-Stark, for their support both in my studies, my research, and everyday work at the University of Pannonia. They helped my work patiently with their suggestions and supplied me with the best pattern of how to be a good consultant/teacher/expert/college. I am grateful for my supervisors' support in my earlier studies at the University of Veszprém, too: Prof. Katalin Tarnay† and Dr. Anna Medve.

My colleges in the department (Department of Electrical Engineering and Information Systems) and at the Faculty (Faculty of Information Technology) have established a good environment that also motivated my work. I appreciate it. I have to highlight one of them: Dr. Péter Görbe, who took over the doctoral school secretary tasks from me, which ensured me more time to deal with the doctoral studies. I hope I can return his gesture soon.

I would like to thank my parents for establishing the possibility to reach this status both in my life and in my studies. Last but not least: спасибо Татьяна!

Supported by the ÚNKP-20-4 New National Excellence Program of the Ministry for Innovation and Technology from the source of the National Research, Development and Innovation Fund. I acknowledge the financial support of Széchenyi 2020 under the EFOP-3.6.1-16-2016-00015.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Scope

In this thesis, two phases of industrial logistics are investigated: the production scheduling and the distribution of products. The appropriate optimization of both phases has a huge financial impact and influences environmental protection, too. An efficient production schedule means huge aid in the production planning process. The definition[1] of production planning by [5] is as follows:

Definition 1. *Production planning: Decisions upon the use of production resources in order to satisfy forecast demand over a certain planning horizon at a reasonable cost.*

The outcome of production planning is one or more production processes.

Definition 2. *Production process: Mechanical or chemical steps used to create an object, usually repeated to create multiple units of the same item. Generally involves the use of raw materials, machinery, and human resources to create a product.*[2] *Operations research applies the concept of "job" for the process.*

Definition 3. *Operation: An elementary step of a production process. This thesis also uses the synonyms activity and task for this concept.*

The classic review on production planning by Gelders and van Wassenhove [5] classifies production and production processes related to the type of manufacturing. The classifications are shown in Figure 1.1.

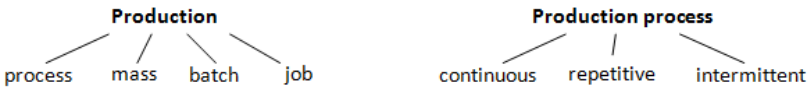


Figure 1.1. Classes of production and production processes.

[1]Some basic concepts have different kinds of definitions, depending on their source. Here only one of them - the one that suits the best to this thesis's aspect - is used.

[2]http://www.businessdictionary.com

Process production means continuous production of the commodity in bulk (e.g., fluid or gas). Both mass production and batch production produce discrete products - batch production in less number - by executing the same sequence of operations. On the contrary, job production deals with made-to-order products. The topic of this thesis can not be classified clearly into any of these groups: the algorithms presented here deal with production that produces more than one types of products like job production (2 in the presented example); however, they are produced in a high number for mass production for each product type.
This thesis's production processes - because of the high cardinality of the same type of products - belong to the repetitive process class. When custom-made products are produced, the processes are intermittent.
As the definition of production planning shows, decisions have to be made on the resources: which resource has to be assigned to an operation and when to start it.

Definition 4. *Resource: An actor that is capable of operating an operation. Usually a resource is a machine (as it can be seen in most of the scientific papers on production planning or scheduling - e.g. [6]); however, the human workforce belongs to this concept, too.*

Definition 5. *Operation time: The amount of time that a resource needs to carry out an operation from its beginning until its end without interruption.*

Since this thesis deals with constant operation times, resources can be handled as machines.

Definition 6. *Resource allocation: The assignment of a resource from the available resource set to an operation that can be operated by this resource.*

Definition 7. *Schedule: For each operation, an allocation of one or more time intervals to one or more resources [7].*

In this thesis, exactly one time-interval is allocated on exactly one resource for each operation of each process. This short-term scheduling supports the medium-term production planning by giving an effective algorithm for determining an optimal or near-optimal schedule [8].
In complex processes, an exact solution often does not exist or cannot be detected. Instead, heuristic methods are applied to obtain an already acceptable near-optimal solution. Results of heuristics are commonly applied to increase the efficiency of exact solvers.
Despite efficient algorithms for the operation of a system, real-life always can produce immediate unwanted cases, e.g., the fault of a resource. Having quick answers and/or appropriate prevention methods, the loss of efficiency can be decreased.

Definition 8. *Fault: The adjudged or hypothesized cause of an error [9].*

The concept of "error" is defined by Avižienis et al. in [9] as follows:

Definition 9. *Error: The deviation of a state of a system from its correct state.*

During the computation process, an error can propagate from component to component in the system. When the error reaches an external interface of the system, it causes service failure. To prevent that, the system has to be designed to be fault-tolerant. For example, Hohl et al. [10] presents a multiprocessor system (MEMSY) with fault tolerance, Arlat et al. [11] models and evaluates two significant fault tolerance software approaches, and Poola et al. [12] presents a survey on fault-tolerant workflow management systems.

Definition 10. *Service failure: An event that occurs when the delivered service deviates from the correct service [9].*

Definition 11. *Fault tolerance: Avoids service failures in the presence of faults [9].*

The third thesis point gives a fault-tolerant solution for the supply chain's distribution phase, related to vehicle routing.

The mission statement of my work is - on the one hand - to create efficient scheduling algorithms for specially structured problems (see in subsections 2.2 and 3.1). On the other hand, I intend to design and implement a fault-tolerant extension for Vehicle Routing Problems (VRP) solutions that can compensate for the immediate breakdown of a vehicle (an operation unit) efficiently.

1.2 Motivation

Both scheduling and routing have a wide application area, and because of that, rich scientific literature.

Scheduling has great importance - besides production process scheduling - in, e.g., scheduling a batch of applications either on multiprocessors [13] or in a client-server environment [14]. An example of an exciting mixture of these applications is the reservation problem of both computing and networking resources in an optical grid system for big data applications [15].

A big challenge of ICT (Information and Communication Technology) networks is the problem of channel allocation. It becomes more complex in wireless networks and sensor networks, where the requirement of energy efficiency is added to the expected QoS (Quality of Service). Advanced dynamic channel allocation methods must adapt to the current needs and the current state of the communications system's environment [16–19]. An example of the intensity of the evolution of this research area can be seen in [20]. In this paper, Dezfouli et al. present a distributed and concurrent link scheduling algorithm applied for data gathering in wireless sensor networks. In their work, the slot reservation of a node is performed one hand by the node itself, while on the other hand, each node participates concurrently in the slot reservation of its neighbours. They achieve 50%-170% higher network throughput with half energy consumption than other methods from the literature.

Routing also arises in several practical application areas - besides vehicle routing - as a significant problem. Next to the common distribution or transportation tasks, maintenance routing is also a challenging task, e.g., in the case of aircrafts [21]. In packet-switched telecommunications networks, data packets have to be routed across

the elements of the subnet. Tanenbaum and Wetherall compose a valuable presentation of network routing protocols in [22]. An interesting ad-hoc network type is created by handheld devices, where the nodes closely follow human movement patterns. Amah et al. give an exhaustive analysis and classification of existing routing protocols in these PSNs (Pocket Switched Network) [23]. Routing has a notable role in the case of public utility systems, too, both in designing their trajectory [24] and in the aggregation and forward of the data from their smart meters [25].
An interesting combination of scheduling and distribution-related location change is crew scheduling in transportation. Heil et al. [26] review more than 100 publications on railway crew scheduling from 2000 to 2019 and classifies the approaches according to model formulation, objectives, constraints, and solution methods.
In the last decades, these two phases of the supply chain - production and distribution - started to be handled jointly.

Integrated Production and Outbound Distribution Scheduling

The separate and sequential optimization of production and distribution phases leads to sub-optimal decisions regarding the supply chain's overall performance. It is one reason why researchers started to coordinate and handle these interdependent functions together [27–30]. In industries that can create production plans for long terms - e.g., vehicle manufacturers with a time horizon of weeks or even a year - both the joint planning of the two phases are easier [31], and the usage of efficient distribution types (rail or water transport) is yielded. The integration of production and distribution planning becomes especially important in the case of perishable products [32, 33], like fruits, vegetables, flowers, nuclear medicines (110 minute is the half-life of a special radioactive substance that is applied in the treatment of many cancer types), blood, concrete, for example. This integrated approach is called IPODS (Integrated Production and Outbound Distribution Scheduling).

Definition 12. *Integrated Production and Outbound Distribution Scheduling: The joint decision of scheduling of production and distribution operations in a supply chain environment.*

Chen presents a state-of-the-art survey on IPODS problems in [34], then classifies the existing models, gives an overview of both solution algorithms and the optimality properties for each class [35].

The complexity of IPODS problems varies on a wide scale - just like its components. There are problems where there is only one manufacturer [36]; however, in other problems, the production phase happens in a multifactorial environment. Behnamian et al. give a review on multi factory supply chains [37]. Based on the interaction of the factories, parallel, series, and network structures can be distinguished [38]. In a series structure, each factory produces semi-finished products for the next plant. The network structure is a combination of parallel and series structures. The shop configuration in each factory is a scheduling problem (see Figure 1.2), including either a single machine [36] or parallel machines [38, 39], and can be handled, e.g., like flow shop, job shop, or open shop (see in subsection 1.3).
Some problems have inventories for temporary storage. Either from the factory or

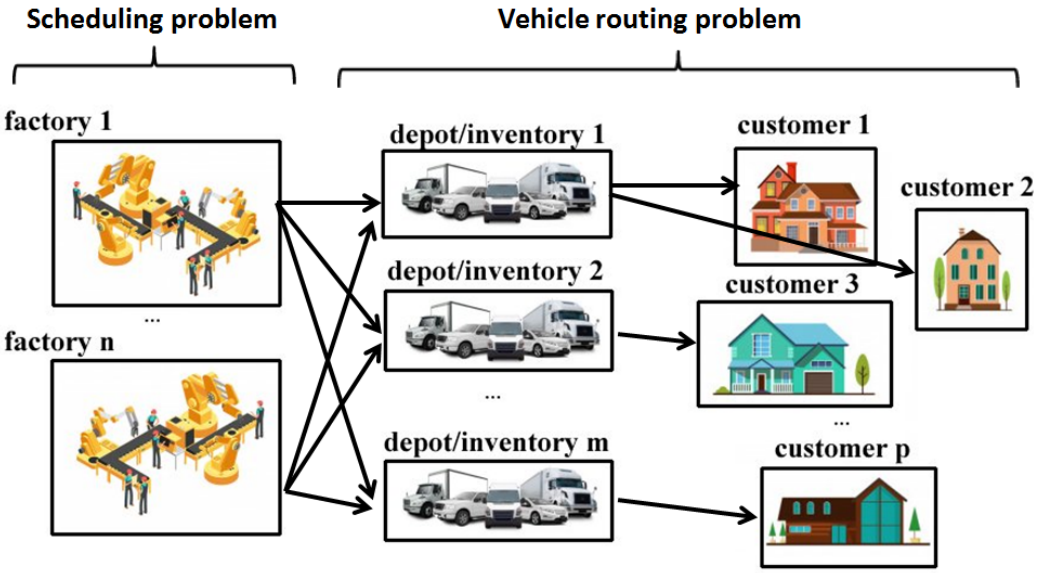


Figure 1.2. The connection between the production and the distribution phases of the supply chain.

an inventory, products have to be delivered. Simple delivery methods include direct shipping or batch delivery to a single customer and batch delivery to multiple customers with direct shipping. Contrary, complex delivery methods determine routes where multiple customers are visited. Because of the integration of production and distribution scheduling, the vehicle routing part of IPODS problems depends on the production phase, too [40]. Kesen and Bektaş give a chronological summary of IPODS literature where routing is considered [39]. The distribution phase's complexity depends on the vehicles (homogeneous or heterogeneous fleet, limited or unlimited number of vehicles, capacity constraints), and the customer needs (e.g., time windows constraints).
The objective of an IPODS problem is usually the minimization of the system cost that includes production, inventory holding, and transportation costs; in some cases, the customers' satisfaction is also taken into account. For example, Liu and Liu consider a single machine, multi-customer IPODS problem with finite, homogeneous vehicles with capacity constraints, and the objective is to minimize the total order weighted delivery time [33]. Even this paper is from 2020, the needs for simplification (a single machine, homogeneous vehicles) are highlighted because of the high complexity of real IPODS problems. Kesen and Bektaş solve a single factory IPODS problem with time windows, where production happens on identical parallel machines, the fleet has a limited number of vehicles, each product has to be shipped to a unique customer, and the objective is the minimization of total earliness and tardiness of all orders [39]. Fu et al. consider a single manufacturer, single machine, single customer IPODS problem where each job has a release date and a delivery deadline [36]. The paper investigates the splittability of both the production and delivery phases, and the objective is to minimize the transportation cost subject to the production release dates and delivery deadline constraints.

The approaches to find acceptable solutions to IPODS problems includes exact methods [32], branch-and-bound algorithms [41], problem-specific heuristics, or metaheuristic methods [42] - like, e.g., multi-objective genetic algorithm [43], ant colony optimization [44] -, and hybrid strategies [45], similarly to other problems of operations research (see detailed in subsection 1.3).

Both production scheduling and distribution, separately, and their integration (an IPODS problem) are complex problems. They have lots of variants, determined by the application area's special constraints. Because of the practical problem's size and complexity, exact solutions are often impossible to find or take too much time. In these cases, approximation algorithms are applied to find an acceptable near-optimal solution. Although a near-optimal solution is a solution with reduced quality, the optimum is often undeterminable for practical reasons: in several cases, some problem parameters can be estimated only, e.g., operation time or setup time (especially considering human resources). It has to be noted that even a few days computation time of problem-solving in the industry is less critical than the quality of the solution. One of its reasons is that the cost of computation on powerful resources is relatively small compared to other costs of the supply chain (and to the value of the available improvement). Another reason is that the computation is usually offline - without a strict deadline - in the planning phase.
Based on the facts presented in this section, both scheduling and distribution algorithms' practical importance and the need for simplification of the real-life-sized problems are justified. My thesis contributes to both the scheduling and distribution phases of the supply chain. On the one hand, it introduces efficient algorithms for production scheduling in mass-production that can be applied in scheduling problems even with special inner structure (see Chapter 2 and Chapter 3). It is also investigated how the developed heuristic algorithm offers efficient aid to an exact solver. On the other hand, an algorithm is developed for vehicle routing solutions without time windows - and composed by circular routes -, that makes them fault-tolerant efficiently (see Chapter 4).

1.3 Scheduling: a wide area in Operations Research

Operations research (OR) is a part of applied mathematics that applies advanced analytical methods for decision support. OR - often by solving an extreme value problem - gives aid in decision preparation, determines the economic optimum or a near-optimal solution [46]. In several cases, the optimal solution can not be found in a reasonable time or at a reasonable cost; in these cases, near-optimal solutions are accepted. The investigated problems of OR include network optimization, assignment, scheduling, routing, transportation, for example [47]. The British Operational Research Society originates the modern OR from 1937, related to improving the UK's early warning radar system [48]. Soon after that - mainly because of World War II. - its importance increased, e.g., in the fields of transportation and logistics. Nowadays, both engineering sciences and economic informatics apply OR.
Scheduling is a significant research area in operations research. Its practical im-

portance was justified in subsection 1.2. The main goal of scheduling is to allocate resources to activities over time [6]. Nagar et al. classified scheduling problems [49]; Figure 1.3 is based on their classification and contains finer partitioning of job-shop problems, following some recent papers [50, 51].

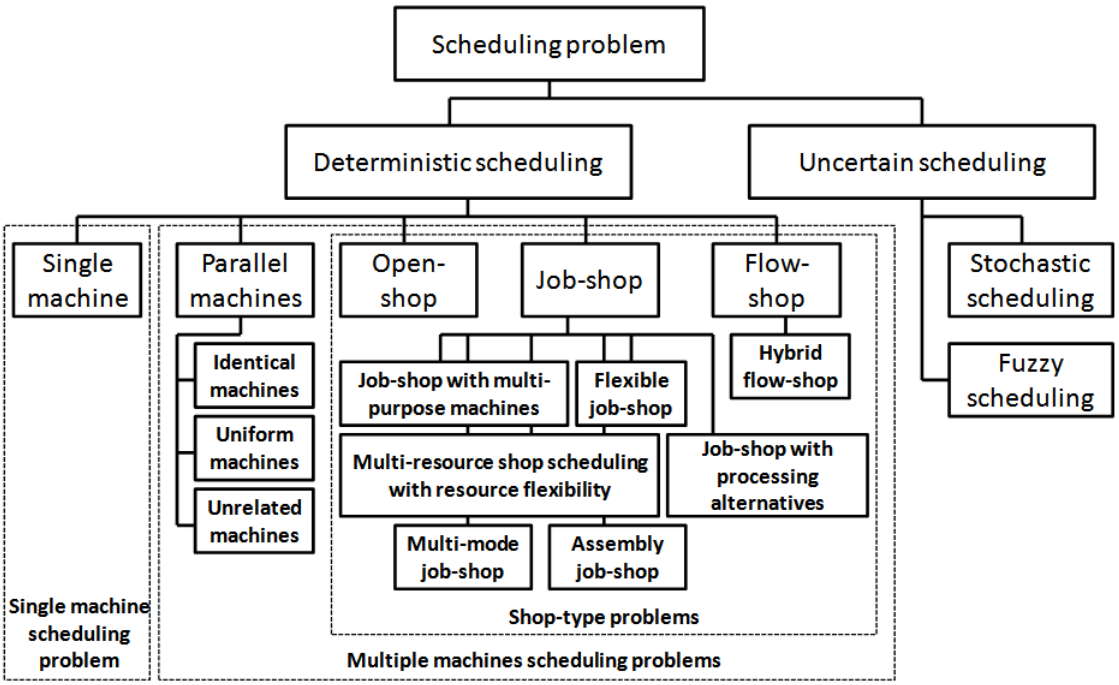


Figure 1.3. The classification of scheduling problems.

The early studies include methods to find the optimal solution for simple problems. The models of these problems contain a single machine. There are related papers that aim to find the solution with the smallest maximum tardiness [52] or obtain the solution with the minimal sum of weighted completion times of the scheduled processes [53].
Contrary to single machine problems, in more complex cases, there are multiple machines. If they have the same function, they are called parallel machines. The machines can be identical, uniform, and unrelated [54].

Definition 13. *Identical machines: Machines of a problem, where the operation time does not depend on the machine.*

Definition 14. *Uniform machines: Machines that may have different operation speeds, however, are identical in all the other senses.*

Definition 15. *Unrelated machines: Machines of a problem, where the operation times depend on the machine assignment.*

The problem model of this thesis contains unrelated machines.
Regarding the inner structure of processes (jobs), single-stage and multi-stage production systems can be differentiated.

Definition 16. *Single-stage systems: One job is composed of only one operation.*

Definition 17. *Multi-stage systems: Jobs are composed of multiple operations.*

A multi-stage system is usually called a "shop problem".

Definition 18. *Shop problem: There are n jobs $J_1, ..., J_n$ and m machines $M_1, ..., M_m$. Each J_i job consists of a set of operations $O_{i,j}$ $(j = 1, ..., n_i)$ that must be processed on machine $\mu_{i,j} \in \{M_1, ..., M_m\}$. There may be precedence relations between the operations of all jobs. Each job can be processed only by one machine at a time. The objective is to find a feasible schedule that minimizes some objective function of the finishing times C_i of the jobs $J_i, i = 1, ..., n$.*

Three general shop problem's three important variations are the open-shop problem, flow-shop problem, and job-shop problem [7, 54, 55].

Definition 19. *Open-shop problem (OSP): Each J_i job consists of m operations $O_{i,j}$ $(j = 1, ..., m)$, where $O_{i,j}$ must be processed on machine M_j. There are no precedence relations between the operations. This problem class is denoted by $Om||C_{max}$, where m is the number of machines and the objective function is the minimization of the makespan (minimization of the maximal finishing time of the jobs).*

Definition 20. *Flow-shop problem (FSP): Each J_i job consists of m operations $O_{i,j}$ $(j = 1, ..., m)$, where $O_{i,j}$ must be processed on machine M_j. There are precedence relations between the operations: $O_{i,j} \to O_{i,j+1}$ $(j = 1, ..., m-1)$ for each $J_i, i = 1, ..., n$. The objective is to find a job order π_j on each machine M_j, that minimizes the makespan. This class of the general shop problem is denoted by $Fm||C_{max}$. If $\pi_1 = \pi_2 = ... = \pi_m$, then the problem is called permutation flow-shop problem.*

Definition 21. *Job-shop problem (JSP): Each J_i job consists of n_i operations $O_{i,1}$, $O_{i,2}, ..., O_{i,n_i}$ which must be processed in this order. There are precedence relations between them: $O_{i,j} \to O_{i,j+1}$ $(j = 1, ..., n_i - 1)$. There is a machine $\mu_{i,j} \in \{M_1, ..., M_m\}$ associated with each operation $O_{i,j}$. It is assumed that $\mu_{i,j} \neq \mu_{i,j+1}$ for $j = 1, ..., n_i - 1$. The general notation of a job shop problem is $Jm||C_{max}$. If there are other constraints, they are noted in the middle part of the notation. For example, if each job has maximum 3 operations, then the notation $Jm|n_i \leq 3|C_{max}$ is applied.*

Figure 1.3 shows that this classification of deterministic scheduling problems can be refined.

Definition 22. *Hybrid flow-shop problem (HFS): Multiple parallel machines are applied at a stage of a flow-shop problem [56]. In practice, an HFS problem often has further assumptions and constraints [57]. Despite its different variants, an HFS problem has most of the following characteristics [56]:*

- *The number of stages is at least 2.*
- *Each stage can have more parallel machines, and there is at least one stage with at least 2 parallel machines.*
- *All the jobs follow the same sequence of stages; however, some stages can be skipped.*

Definition 23. *Assembly job-shop problem (AJSP): It starts with JSP and appends an assembly stage to the completed jobs [58].*

Definition 24. *Job-shop problem with multi-purpose machines (MPM-JSP): An extension of a JSP, where for each operation, a subset of machines is associated from which exactly one machine must be chosen to perform the operation [59].*

Definition 25. *Flexible job-shop problem (FJSP): An extension of an MPM-JSP, where the operations may have different operation times on the different machines [51, 60].*

Definition 26. *Multi-mode job-shop problem (MMJSP): There is a set of machine sets (processing modes) associated with each operation. Each processing mode has its own processing time. Exactly one machine set must be chosen for each operation, and the operation has to be performed on each machine of the selected machine set, in parallel [61, 62].*

Definition 27. *Multi-resource shop scheduling with resource flexibility (MULFLEX-SP): Each operation must be executed on a pre-specified number of distinct machines in parallel, where each machine must be chosen from a specified set of machines. An operation may have more than one predecessor and/or more than one successor in its job [51, 63].*

Definition 28. *Job-shop scheduling with processing alternatives (AJSP): An extension of a JSP, where the directed graphs of job routings contain additional dummy nodes that identify and-subgraphs representing partial order of operations and or-subgraphs whose branches mean alternative subroutes [51].*

This thesis's scheduling problem (described in detail in subsections 2.2 and 3.1) has common characteristics with some of these extensions; however, it does not fit completely into any of these categories. It deals with multi-stage, multi-product, multi-machine system with more than one parallel machines at the stages. In this sense, the problem belongs to HFS. However, not all the jobs follow the same sequence of stages. In the problem, there are 2 types of jobs that are nearly the same: one of the job types has an additional operation compared with the other. Both job types have to be executed several times since many products must be produced of both product types. In this sense, the problem belongs to mass production regarding each job type. For both job types, the precedence constraints between the operations are well defined, including one operation in each job type with exactly two predecessors. This operation realizes an assembly stage (see assembly job-shop problem) and ensures the parallel execution of the connected branches (an and-subgraph of AJSP). However, opposite to the AJSP presented in [51], here, two operations of the same job - located on the parallel branches - may be processed in parallel. The problem belongs to FJSP in the sense that there are operations that can be executed on different machines with different operation times. However, in the presented problem, the assumption "$\mu_{i,j} \neq \mu_{i,j+1}$ for $j = 1, ..., n_i - 1$" does not exist since successive activities of a job can be processed on the same machine. These properties result in a really complex problem. Following the three-field-notation [6],

the scheduling problem of this thesis is a special variant of the problem that can be denoted as $R_m|prec|C_{max}$, i.e., given m unrelated machines, there are precedence constraints between the activities, and the objective is to minimize the makespan. In fact, the considered problem is located at the frontier of $R_m|prec|C_{max}$ and FJSP problems. Classifying the problem into one of these groups, some characteristics of the problem are lost.

Optimization algorithms were developed to solve search problems: the search for the optimum. They can be used, for example, to solve scheduling problems. Based on [64], [65] and [66], Figure 1.4 classifies them and enumerates some significant members for each class.

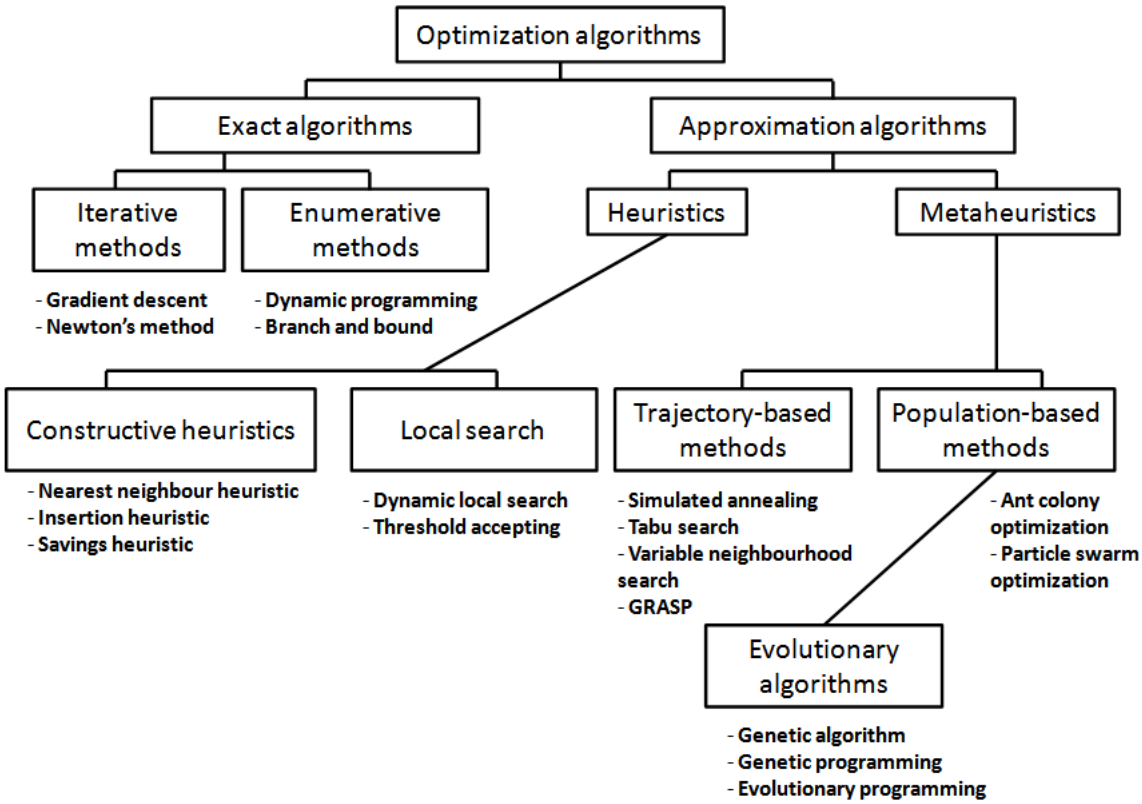


Figure 1.4. The classification of optimization algorithms.

Heuristics vs. exact methods and genetic algorithms are presented in the following subsections in detail because they have the main role in solving this thesis's scheduling problem.

1.3.1 Exact methods and heuristics in scheduling

The more complex the problem is, the more computation is required to find its optimal solution. Complexity theory [67] deals with the hardness of solving problems. Scheduling problems are usually classified as polynomially solvable or NP-hard. As the job-shop scheduling problem (JSP) is NP-hard [68], it is impossible to find its optimal solution in a reasonable time as the search space of the problem increases.

For less complex problems, enumerative algorithms, like branch and bound, or dynamic programming can be used efficiently. It is a trend in OR that after preparing the problem model, a solver is applied to that to find its solution. For example, such solvers/tools are the freely available GLPK (GNU Linear Programming Kit) and the commercial CPLEX, CP Optimizer or Gurobi.

An introduction to production planning with mixed-integer programming techniques can be found in [69]. Other books on the topic of production planning and scheduling are [70], [71] and [72]. In [73] two extreme cases of batching (i.e., a successor item can only be produced if the full batches of the predecessor items are finished) and lot-streaming (i.e., the successor item can be produced simultaneously with its predecessor items as long as it uses a different resource) are considered. The problem of this thesis is a mix of these two variations. In [74], precedence constraints are applied; however, only on a single machine. This case can be regarded as a subproblem of the model of this thesis. A single resource model (for producing different kinds of products) is considered in [75], too. The paper gives an exact algorithm based on a Cut and Branch procedure. An earlier paper [76] treats a similar model where a single product is produced. This thesis's production model applies a multi-machine environment and has two product types, each of them with high cardinality (see subsections 2.2 and 3.1).

Unfortunately, the method that builds the problem model and uses an exact solver to find its solution has limits. The required computational complexity may cause that we have to be satisfied with a near-optimal solution instead of finding an optimal solution. Local search methods, constructive heuristics, and metaheuristics are used commonly instead of exact solution methods.

Definition 29. *Heuristic method: A problem-dependent solving technique that tries to get full advantage of the particularities of the problem. Heuristic methods are applied when results have to be obtained quickly, and a near-optimal solution is enough.*

That is the reason why these greedy algorithms are special and get trapped often in a local optimum.

1.3.2 Metaheuristics and a special method: the Genetic Algorithm

Contrary to heuristic methods, metaheuristics can be applied more generally.

Definition 30. *Metaheuristic: A high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop a heuristic [77].*

As Figure 1.4 shows, metaheuristics include tabu search (TS) [78], simulated annealing (SA) [79], ant-colony optimization (ACO) [80], or genetic algorithm (GA) [81], for example.

Definition 31. *Genetic algorithm: A global search metaheuristic method inspired by evolutionary biology, belonging to a special class of evolutionary algorithms. John Holland originated the basic idea of GA [81].*

GA belongs to population-based metaheuristics, as it maintains a pool of solutions and changes its instances from generation to generation to find the solutions to optimization problems. Like the evolution process in biology, it tends to create new instances that accommodate better to the (environmental) challenges than the instances of the former generations. The most relevant concepts are:

Definition 32. *Generation: The set of solutions with the same age (originated from the same iteration of the algorithm).*

In GAs, the size of the generations of a problem is usually constant.

Definition 33. *Population: The set of solutions of the same generation.*

Definition 34. *Instance: One solution of a population.*

Definition 35. *Chromosome: The set of the variable values of the problem that the genetic algorithm is trying to solve, encoded in an instance.*

Definition 36. *Fitness function: A function that maps chromosomes into real numbers, expressing how the instance is adequate to the desired purposes.*

Definition 37. *Fitness value: Function value of the fitness function.*

Random instances can fill up the initial population of a GA, or some/all of its instances can be constructed in a "clever" way, e.g., by applying a heuristic. Then, the next generation is constructed from the instances of the previous generation, applying genetic operators. During this process, the intention is - on the one hand - to increase the (average) fitness value from generation to generation. On the other hand, genetic diversity should be ensured, avoiding stuck in a local optimum. Elitism, crossover, and mutation are applied to create all the instances of the new generation.

Definition 38. *Elitism: Instances with high fitness value are copied without any modification from a generation to the next generation.*

Definition 39. *Crossover: Pairs of instances of the population are selected for recombination. The results of this genetic operation are placed in the new generation.*

Definition 40. *Mutation: A small modification of a chromosome of an instance. In GA, the objective of applying mutation is to jump out of a local optimum, conducing to the discovery of the whole search space.*

Unless meeting the termination condition, the creation process of a new generation is repeated. Otherwise, the instance with the best fitness value is chosen to be the solution, and the algorithm stops. The workflow of general GA is illustrated in Figure 1.5.

Alba and Chicano [82] prove that GA is an appropriate tool for project scheduling and can be applied for automated task assignment efficiently. Jans and Degraeve [83] review various metaheuristic methods for different lot-sizing models. Several papers consider very special models, like [84]. Chang et al. [85] - extending their former work [86] - present a GA with improved representation, which takes into

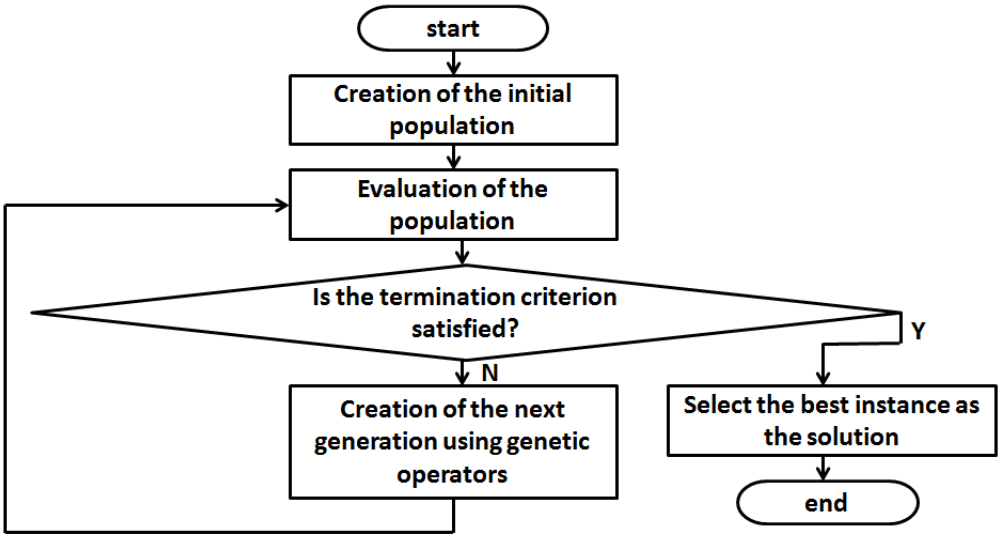


Figure 1.5. The workflow of a general GA.

account human resource factors more. Moreover, as the representation introduces a time-line axis as a third dimension (next to tasks and resources), suspension and resumption of tasks and reassignment of resources are possible.

Sadegheih uses simulated annealing to determine the effect of GA parameters on the schedule [87]. In his work, there are 8-jobs and 7-machines. He found that the mutation rate is important; however, the effect of the crossover rate is not significant.

Zhang et al. [88] review some GA for FJSP and introduce their effective problem representation and genetic operators. They demonstrate that a high-quality initial population is crucial in an efficient GA. Li and Gao [89] hybridize GA - which has a good global searching ability - with tabu search - that has powerful local search ability. This way, they efficiently solve FJSP problems. Kis [51] creates a GA for job-shop scheduling problems with processing alternatives, where the chromosomes are schedules. The size of the problem of Pongcharoen et al. [2] is similar to the size of the problem considered in this thesis (similar number of resources and tasks). However, their problem has a less complex structure. They determine the optimal parameters of their GA and describe how they face and eliminate time-loop (circle in the directed graph of the problem; they call it deadlock). When they get an infeasible schedule, they swap the problematic operation with a random one. This thesis handles this problem in another way.

Here, in one of my thesis points, a GA is also presented that is applied for scheduling in a multi-project environment, where resources may have multiple functions, and they can substitute each other. The feasibility of instances is guaranteed by the manner how genetic operators are constructed. The production planning model that is considered in this thesis is quite special (for the details, see subsections 2.2 and 3.1), as despite the number of products can be huge, there are only a few different product types (i.e., 2), and each product consists of only a few components. None of the cited papers is devoted "just" to this problem. The results related to the constructed GA are published in [90].

1.4 Vehicle Routing - the final phase of the supply chain

When products get ready - e.g., in a production process scheduled to be efficient -, they have to be distributed. The cost of product delivery is decreased as the total length of the routes of delivering vehicles becomes shorter. Motivated by this fact, a significant amount of research in applied optimization is devoted to design strategies of optimal vehicle or vehicle fleet operation, where the problem of vehicle routing is of primary importance. Therefore, a large number of papers and a lot of research aim to optimally determine the routes of a vehicle fleet serving consumers under different conditions. Problems of routing vehicles are classified based on these conditions. Following the taxonomy presented in [91], [92], [93] and [94], Figure 1.6 shows the most significant and most relevant variants of the problem. It is also worth mentioning the paper of Wang and Wasil [95] that reviews all the significant VRP-related papers published by the journal Networks in the last 50 years.

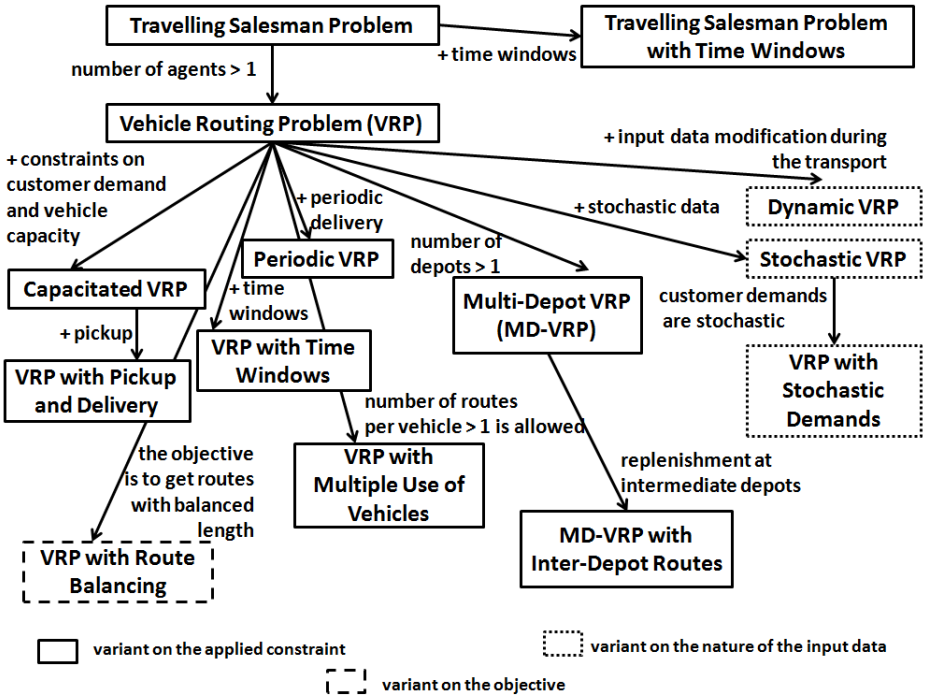


Figure 1.6. Significant variants of Vehicle Routing Problem.

Definition 41. *Travelling Salesman Problem (TSP): An agent starts from the depot, visits all customers exactly once, and returns to the depot. This route has to be done with minimal cost. The problem is equivalent to finding the minimal Hamiltonian cycle in the graph that describes the road-system. TSP is NP-hard [96–98].*

Definition 42. *Vehicle Routing Problem (VRP): A multi-agent generalization of TSP. Here, the procedure of solving TSP is extended by partitioning the customers into disjoint routes [99].*

Definition 43. *Capacitated VRP (CVRP): A VRP that contains constraints related to the amount of customer demands and the capacity of the vehicles.*

Because of the evident nature of capacity constraints, CVRP is often reflected as - simply - a VRP.

Definition 44. *VRP with Time Windows (VRP-TW): A VRP, where every customer has a time window and serving time. If the vehicle arrives at the customer before the lower bound of the customer's time window, it has to wait. When the arrival is later than the upper bound of the customer's time window, the vehicle missed the customer. In some cases, the depot has a time window, too [100–103].*

Definition 45. *Multi-Depot VRP (MD-VRP): A VRP with more than one depot. Each vehicle starts from a depot, and it has to return to the same depot after performing the delivery tasks [104].*

Definition 46. *MD-VRP with Inter-Depot Routes (MD-VRPI): An MD-VRP, where the replenishment of vehicles at intermediate depots is allowed [105].*

Definition 47. *Periodic VRP (PVRP): A VRP version, where the transfer has to happen at certain times in a well defined time interval (e.g., two times a week) instead of a single delivery [106].*

Definition 48. *VRP with Pickup and Delivery (VRPPD): In this VRP version, the customers are allowed to send goods by the vehicles besides receiving products [107, 108].*

Definition 49. *VRP with Multiple Use of Vehicles (VRPM): A VRP, where one vehicle can be used for more than one trip. It often happens when a company has a small cardinality vehicle set, or the average distance between the customers and the depot is small.*

Aside from various constraints, different objectives of the delivery also generate their VRP variants.

Definition 50. *VRP with Route Balancing (VRPRB): A VRP, where the objective is to minimize the difference between the shortest and the longest route [109].*

Extra demands of customers, a technical problem of the vehicle, or a closed road necessitates modifying the predefined transports. Recent advances in information technology made it possible to process additional data and do modifications while vehicles are on the road.

Definition 51. *Dynamic VRP (DVRP): A VRP, where modification of problem-related data during the transport is allowed, and the trips can be adjusted to that [110–112].*

There are problem variants, in which some data are not determined as static values but are random variables. These data can be customer demand, the travel times, or the service/waiting times, for example.

Definition 52. *Stochastic VRP (SVRP): A VRP variant, where some data are stochastic variables [113, 114].*

Definition 53. *VRP with Stochastic Demands (VRPSD): An SVRP where customer demands are stochastic [115–117].*

An important class of time-varying circumstances is when the reason for the change of the problem state is an error during execution, caused by a fault of an element of the system, e.g., the engine of a vehicle or a blocked road. Despite the modified state of the system, the main goal is to avoid service failure, i.e., servicing all the customers. Fault-related topics, like fault detection, diagnosis, or tolerance, are extensively examined, e.g., in the case of control and technical systems of both military [118] and public applications for a long time [119].
In [120], the threat is either an unknown future request or a traffic problem. The authors apply a fuzzy fault-tolerant control approach: all customer requests whose future servicing is affected by a problem will be reassigned as new requests. In VRPSD literature, the notion of "route failure" is used when a vehicle does not have enough capacity to service the customers who are assigned to its route. Usual solution methods for this problem have two stages: the first one specifies a route for each vehicle, then the second stage deals with the recourse action in case of route failure [111]. It means that an appropriate response may be significantly delayed. In [121], the traditional recourse action - when in case of route failure, the vehicle may return to the depot and then perform an extra trip - was completed by a new recourse strategy. In this strategy, a vehicle that has completed its own original route and has remaining capacity can serve customers from the failed route. (However, it needs to be emphasized that contrary to the deterministic demand that is used in our research, Novoa et al. [121] deals with stochastic demands. Moreover, their concept of "route failure" also differs from ours.)
The aid one vehicle offers another is one of the main priorities in my third thesis point - however, the threat to the service considered here is the breakdown of a vehicle before it has served its last customer. Based on the fault classification in [122], this kind of fault belongs to the category of "abrupt faults". Moreover, in this thesis, the routes' execution is already prepared in the route planning phase, which minimizes the fault-related extra costs without changing the set of routes itself.
To achieve fault tolerance, the fault has to be detected. In my thesis, it is assumed that when a fault - e.g., the fault of the engine of a vehicle - causes a vehicle failure - e.g., a vehicle breakdown - and the fleet's vehicles communicate with one another, fault detection is trivial. Specific fault detection methods can be found for other VRP-related fault concepts in [120] and [122].
The execution of a VRP program can be made less sensitive to vehicle breakdown by considering the possibility of cooperation among vehicles. Cooperation has a positive effect on transportation costs, even in cases where there are no unforeseen events. This is demonstrated in [123]. The concept of "cooperation" here means that a vehicle takes over the servicing of a customer or a set of customers originally assigned to another vehicle. In the common case of multiple vehicles servicing customers, two questions need to be considered: selecting which vehicle should take over the task and its costs.

1.5 The structure of the thesis

The part of my thesis that is related to production scheduling covers - on the one hand - the development of a specific heuristic scheduler, the exact solving of a MILP (Mixed-Integer Linear Programming) model of the same problem that the heuristic solver was applied to, and the investigation of their combination. On the other hand, I developed a genetic scheduler for a different problem with a similar structure. The GA was tested on several FJSP problems and compared to other results from the scientific literature. After production scheduling - considering the supply chain's final phase - a fault-tolerant execution proposal related to the Vehicle Routing Problem (VRP) is considered.
The detailed content of the chapters of the thesis are:

Chapter 2 presents a complex scheduling problem model, a heuristic approach for solving the problem, and analyses the applied heuristic method's efficiency. Here, it is also investigated how efficiently the heuristic algorithm can support an exact solver. Since the problem was also solved by an exact solver, the chapter contains the problem's MILP model. The essence of this chapter was published in [124].

Chapter 3 covers the presentation of a genetic scheduler algorithm. The GA was developed for a slightly different problem model - related to the previous chapter. First, the specialities of the problem are introduced. Then the details of my GA are presented: the applied encoding technique, the developed genetic operators, and other specialities of the algorithm. After this presentation, the efficient settings of the genetic parameters are investigated. Finally, the quality of the GA is compared with other results from the scientific literature on some FJSP problems. The results contained in this chapter are published in [90].

In **Chapter 4**, a method for Vehicle Routing Problem (VRP) is given that improves the fault tolerance of an arbitrary VRP solution without Time Windows. Here, the idea behind the algorithm is introduced, then the algorithm is presented, and also a fault-tolerant execution of the algorithm is proposed. Finally, the solution's efficiency is proved both by a simple example and through the Solomon test instances that are commonly applied as VRP benchmarks. The content of this chapter is published in [125].

Finally, **Chapter 5** concludes my work, presents the new scientific contribution (thesis points) and suggests further research.

The main part of this thesis contains my work and the closely related knowledge that helps to understand it. The description of other researchers' methods and the details of the comparisons can be found in the Appendices.

Chapter 2

Production planning with multi-level heuristic scheduling

2.1 Background

Heuristic methods (see Definition 29) usually only approximate the optimal solution, but they do it far quicker than exact solvers find the optimum. Moreover, often only small or moderately-sized problems can be practically solved to provable optimality by exact solvers. This property makes heuristics especially useful when time and cost are critical factors for decision making, like, e.g., in production planning [126]. A heuristic approach creates solutions by applying rules determined during the conscious exploitation of the problem's specialities. The initial solution is often improved progressively by the application of a series of modifications.
The evaluation of the efficiency of a heuristic algorithm considers both the quality of the solution and the way how it was obtained. If the optimal solution of the problem is known, the distance between the optimum and the solution of the heuristic gives a measure for the quality. Heuristic search is based on the problem space hypothesis - defined by Newell and Simon [127] -, which states that every possible state of the problem can occur in the problem space. Thus the efficiency of a heuristic search algorithm is usually measured by the number of the visited states. However, because of the difference in the state expansion overheads, CPU time is also considered during the comparison of heuristic algorithms [128]. In his Ph.D. thesis [128], Dillenburg proposes different techniques that improve the efficiency of a heuristic search algorithm, including an efficient data structure, cycle checking that prevents the generation of duplicate states caused by cycles in the search space, and improved search algorithms.
Both heuristics and exact optimization techniques have specific advantages that complement each other. By combining them, higher performance can be reached. A survey of Puchinger and Raidl classifies the combinations of exact methods and metaheuristics and justifies its usefulness [129]. Figure 2.1 shows the major classes.

In collaborative combinations, the algorithms exchange information but are not part of each other, unlike the algorithms of integrative combinations. In the case of sequential execution of the collaborative algorithms, two different approaches exist. In the first one, the exact solver is used before the heuristic method, e.g., first, a

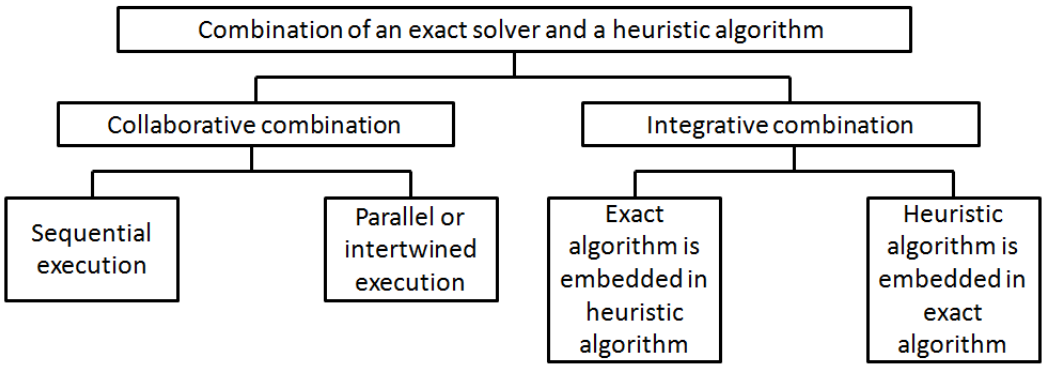


Figure 2.1. Main classes of the combination of an exact solver and a heuristic algorithm.

relaxation of the original problem is solved to optimality, and this optimum helps in the construction of the starting solution of the heuristic that is applied to find solutions for the complex problem. The other approach is applied to obtain the optimal solution in a shorter time, e.g., by using the near-optimal solution obtained by a heuristic to narrow the time horizon of an exact solver in its search for the optimum. I followed the second approach and developed a specific heuristic for a complex scheduling problem - the problem is presented in subsection 2.2 - and investigated how much the heuristic algorithm's result can support the operation of an exact solver. The essence of this chapter is published in [124].

2.2 The model of the problem

The problem that is the subject of the multi-level heuristic scheduler is presented in this section. This problem contains two types of production processes and four types of machines. To define processes, process models are adequate tools. Process models are used in various fields, e.g., to describe business processes, production processes, communication protocols, or software's operation. Contrary to state-based models (like state machines), in the case of process models, state transitions have a temporal dimension - so, they are not instantaneous. A process model emphasizes the activities of its processes and their relation [130]. As one of the five process modeling paradigms presented by Conradi et al. [131], task net is used to present the model of the scheduling problem of this chapter.
Figure 2.2 shows the process models of the two types of production processes of the considered problem, represented by UML activity diagrams. Both process types are executed several times; the number of their execution is $n(p_1)$ and $n(p_2)$, respectively. The figure contains the resource set and illustrates - by different patterns on the operations - the allowed resource assignments. E.g., a t^C-type operation can be performed either by a D_1-type resource or by a D_2-type resource.

D_1-type and D_2-type resources can perform both t^B-type and t^C-type operations; however, the operation time depends on both the resource type and the operation type. Switching between operations with different types on a machine requires setup

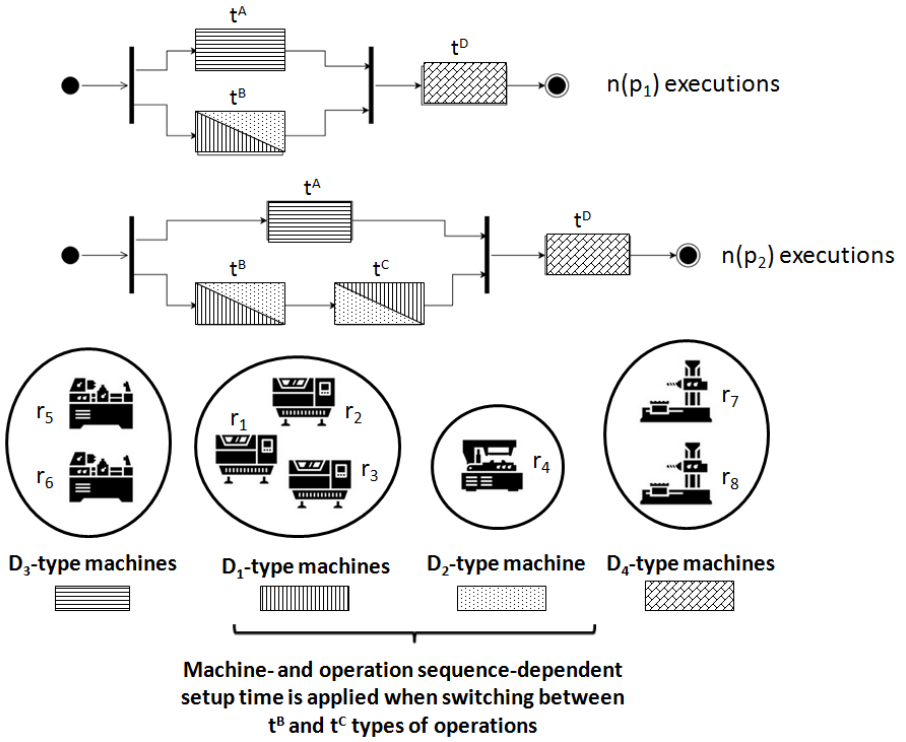


Figure 2.2. The production processes, the resource set, and the resource allocation possibilities of the scheduling problem.

time. This setup time depends on the machine type and the operation sequence. The objective is to find the schedule with minimum makespan.

Definition 54. *Makespan: The distance in time between the start and end time of a schedule. It is the difference between the finish time of the latest operation and the start time of the earliest one.*

The formal description of this problem can be found in Appendix C, using the notations introduced in Appendix B. The notations presented in Appendix B covers the scheduling problem of both Chapter 2 and Chapter 3.
I developed a heuristic algorithm that efficiently solves this scheduling problem. Its efficiency is analyzed on different problem sizes, where the numbers of the products, the operation times, and the setup times are varied. Then the effect of the result of the heuristic algorithm on an exact solver's performance is analyzed.
This production problem includes several special features of a huge class of industrial-type automated manufacturing problems. It realizes mass production; both process models are executed several times. The problem belongs to FJSP (see Definition 25) since an operation's operation time depends on the performing machine. Moreover, the problem includes multi-purpose machines. On the other hand, both process models include an assembly operation. These - t^D-type - operations can be found directly after the join notations in the UML activity diagrams. They result in and-subgraphs (see job-shop scheduling problems with processing alternatives in Definition 28). The two joint branches of any process can be executed parallel. It

means that there is only a partial ordering of the operations of the parallel branches. E.g., a process of the second process type can be executed in 3 different ways, where the sequence of the operations is (regarding their type): t^A,t^B,t^C,t^D or t^B,t^A,t^C,t^D or t^B,t^C,t^A,t^D. Moreover, some of the operations may be performed at the same time or partially covered in time. Then, the two parallel branches are synchronized at the join (before the assembly operation).
Although both [132] and [133] deal with unrelated machines, no of them considers the same model that is treated here. Papers that consider scheduling problems with unrelated machines in the presence of precedence constraints discuss mainly processes that are chains. In the production processes of this thesis, some components are assembled. It means that there are nodes in the precedence graph with more than one predecessor. Kumar et al. [134] consider a similar problem; however, in their paper, approximation algorithms are proposed with approximation guarantee. My thesis proposes a complex heuristic algorithm and also provides the optimal solution by an exact solver. Kis [51] developed a GA and a TS algorithm for job-shop scheduling problems with processing alternatives. His algorithms can handle problems where precedence constraints include and-subgraphs and/or or-subgraphs. Contrary to this thesis, he assumes that no two operations of the same job may be processed in parallel. In [135], an effective priority rule-based heuristic algorithm is provided for a $R_m|prec|C_{max}$ problem (the three-field notation is based on [54], and it means that there are m unrelated parallel machines, there are precedence constraints on the operations, and the objective is to minimize the makespan). In [136] its variant is considered, where each job has to be assigned to a unique machine. However, the problem of this thesis is an extended FJSP problem and contains multi-purpose machines.

2.3 The specific heuristic algorithm

I created a heuristic algorithm that efficiently solves the introduced problem. It produces a "good enough" solution quickly for any input. The algorithm was designed for the specific problem; however, the basic ideas of the algorithm's construction can be applied to schedule other production processes, too.
The main idea was to separate the master problem into subproblems that can be handled individually. The separation was based both on the structure of the processes that have to be scheduled and the resources' characteristics. The initial scheduling problem was cut into three subproblems; the concept "room" was introduced for these subproblems. They are illustrated in Figure 2.3.

When separating the initial problem into subproblems, those operations got into the same room that

- has a common machine that can perform them (e.g., both t^B-type and t^C-type operations can be performed by a D_1-type resource - and it is valid to a D_2-type resource, too),
- and there is no operation from other room that is inserted between them in the operation sequence of their process.

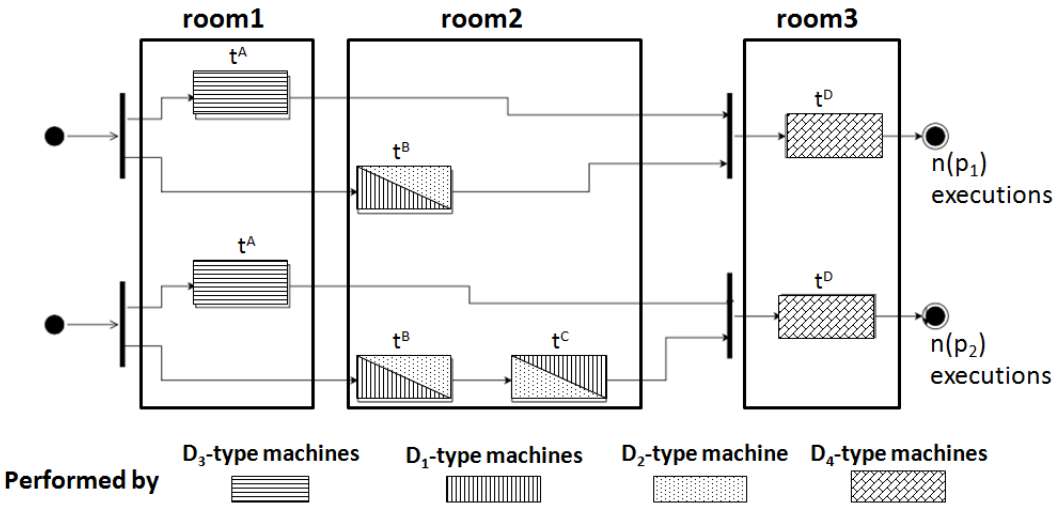


Figure 2.3. The subproblems of the initial problem are handled in separate "rooms".

Based on it, when separating a set of general process models into subproblems, necessary conditions could be that (1) the operations that can be performed by machines that can substitute each other have to belong to the same subproblem and (2) in the process graph there can not be two nodes of the same room that are connected through a node of another room. Here, only two - possible - necessary conditions are noted; the general problem separation can be the topic of future research.
Every room can be handled individually by a scheduling technique that fits the best for that. There can be, of course, some dependencies between the different rooms that have to be cared for. The details are presented in the following.

room1: initial operations

It is easy to confirm that in this room, a greedy method is sufficient. Since both tasks of this room have task type t^A, they are handled by both r_5 and r_6 machines in the same way (they have the same processing time). Moreover, there is no release time and no deadline dedicated to the tasks, so they can be distributed equally (or almost equally if the number of machines does not divide the number of tasks) among the machines, which must be an optimal solution for this subproblem.

room3: the final assembly

For the problem that is represented by this room, another greedy method is sufficient. Here, the solution is a bit more difficult than in room1. As soon as a product piece in room1 is ready and another product piece created in room2 is also available, and there is a free machine in room3 to start a new operation, the two product pieces' assembly can be started. This is an optimal strategy if both process models are identical regarding their assembly times. (In a more general case, dedicated products could have to be created for different customers, where a high-quality product needs

larger processing time compared to a common quality product. However, this is not the case in the considered model.) Hence, this room's subproblem can be solved by a simple list scheduling algorithm, which also considers release times.

room2: the complex part of the problem

This is the hardest subproblem among the three, which can be still easy in some cases, depending on the processing times of the operations (including the processing times of other rooms' operations, too). If the "bottleneck" occurs in the first room (the operation time in room1 is higher than the other operation times in the other rooms), then a "lazy" schedule in room2 is still good enough to reach an overall optimal schedule. Otherwise, if the time in room2 is very constrained (compared to the time required for the work in the other rooms), it is still possible to get a quite good overall schedule with the help of some tricks. These tricks include a list of simplifications, a relaxation, a rounding technique, and finally, a kind of improvement. They are described in the following in more detail.

2.3.1 Simplifications related to job assignment and scheduling; and the main steps of the heuristic algorithm

A sequence of simplifications has been applied for the subproblem of the second room. There are 4 machines that can perform the operations of room2. These machines are of two types: r_1, r_2 and r_3 are D_1-type machines, while the type of machine r_4 is D_2. Following the notations of Appendix B, $n(D_y)$ denotes the number of machines with type D_y. In this concrete case, $n(D_1) = 3$ and $n(D_2) = 1$. Either machine type can carry out both operations of type t^B and operations of type t^C (it means that all the 4 machines of room2 can execute all operations of room2). Suppose that D_1 and D_2 machine types differ in their efficiency as follows: the three machines of type D_1 are better in executing t^B-type operations and less effective in executing operations of t^C-type. Moreover, the D_2-type r_4 machine is better at executing t^C-type operations and less effective in performing operations of t^B-type.

The question is how to distribute the t^B-type operations and the t^C-type operations among the four machines. At this point, several simplifications were made. It has to be emphasized that some of the simplifications surely hold in any optimal solution to the problem. Some others do not hold in general; however, now the goal is only to find a feasible approximate solution, which can be used to find an optimal solution using a MILP solver.

2.3.1.1 Exclusion of setup times by a good job sequence if possible

The simplifying assumptions made for room2 (S1-S5) are based on some aspects of the optimal solution's suspected form. The first two simplifications are:

(S1) On each machine, setup time occurs at most once.

(S2) On each machine, any t^B-type operation precedes any t^C-type operation.

Reasoning: Suppose that there are operations that have already been assigned to some machine. In this case, the moving of a t^B-type operation to the beginning of its machine's queue will not increase the makespan. Recall that t^B-type operation precedes t^C-type operation in the process model (see Figure 2.2). By the running of some computer simulations, it was found that assumptions S1 and S2 hold in many cases. However, in some rare cases, assumption S1 is not satisfied. In these cases, the optimum is better than the solution provided by the heuristic, and some t^C-type operations must be made before t^B-type operation(s) so that the assembly operations of the third room can be started earlier. In these cases, setup time has to be applied more than once on some machines. The satisfaction of S1 and S2 assumptions strongly depends on the different operations' processing time; however, they are often satisfied.

2.3.1.2 Pick an appropriate subcase from the set of all possible scenarios

The third simplification is:

(S3) Either D_1-type machines make no t^C-type operation, or the D_2-type machine makes no t^B-type operation.
(The second part of the assumption more generally - in case of two types of machines in the room - is: "or D_2-type machines make no t^B-type operation". However, in the model under consideration, there is only one D_2-type machine (r_4).)

Reasoning: Suppose that the opposite of the assumption happens. It means that there is a D_1-type machine which carries out also t^C-type operation (in performing a t^C-type operation, this machine type is less efficient than the D_2 machine type), and also, the single D_2-type machine of the concrete model executes t^B-type operation(s), too (in performing t^B-type operation this machine is less efficient than a D_1-type machine). If the two operations in consideration are swapped, the number of operations of the different task types remains. Moreover, the operation time of both machines under consideration decreases. Unfortunately, in general, it does not hold that also, the makespan decreases.

According to S3, there are two cases to consider:

Case 1: The (in general: any) D_2-type machine makes only t^C-type operations.

Case 2: Any D_1-type machine makes only t^B-type operations.

Moreover, within Case 1, the following possible situations can be distinguished: There are $i \in \{0, 1, 2, 3\}$ (in general: $i \in \{0, 1, \ldots, n(D_1)\}$) D_1-type machines that will have only t^B-type operations, and the other $3 - i$ (in general: $n(D_1) - i$) D_1-type machines perform both t^B-type operations and t^C-type operations (and the one /in general: all the $n(D_2)$/ D_2-type machine performs only t^C-type operations). Similarly, within Case 2, there are 2 (in general: $n(D_2) + 1$) possible subcases, according to the number of D_2-type machines that perform both operations (in the concrete case 0 or 1 machine). All these 5 (in more general: $n(D_1)+1+n(D_2)+1-1 =$

$n(D_1) + n(D_2) + 1$) possible subcases are considered one by one, and finally, the subcase with the best solution is chosen. (The reason for the "-1" term in the previous sentence is that both Case 1 and Case 2 have the subcase where the number of machines carrying out both operation types is 0, and these two subcases are identical.)

From now on, let us suppose that a subcase within Case 1 is considered (a subcase within Case 2 can be handled similarly), that can be characterized by the following properties (simplifying the notation):

- there are α D_1-type machines that perform only t^B-type operations,
- there are β D_1-type machines that perform both t^B-type operations and t^C-type operations,
- and there are γ D_2-type machines that carry out only t^C-type operations.

(For the considered model $0 \leq \alpha \leq 3$, $\alpha + \beta = 3$, and $\gamma = 1$. In general: $0 \leq \alpha \leq n(D_1)$, $\alpha + \beta = n(D_1)$, and $\gamma = n(D_2)$.) From now on, this subcase is referred to as "the chosen subcase". For simplicity, it is also assumed that all the values α, β, γ are positive. If any of them equals zero, that subcase should be handled similarly.

The simplifications introduced so far concern only the machine assignment and the sequence of the operations. To calculate the makespan, the start time of each operation of a feasible assignment has to be determined. The result of this timing process is the schedule.

The fourth simplification is related to the timing of the operations.

(S4) The (in general: any) D_2-type machine can make all its work without any idle time (except that the /in general: any/ D_2-type machine can start its processing at time $dur(t^B, D_1)$, where $dur(t^B, D_1)$ is the processing time of a t^B-type operation on a D_1-type machine - see the notations introduced in Appendix B.). Moreover, any D_1-type machine that makes t^C-type operation(s), too, can work on the t^C-type operation(s) continuously after the setup time is inserted after its last t^B-type job.

Reasoning: The start of a t^C-type operation necessitates a finished t^B-type operation as its pair. Following (S2) and based on the number of D_1-type and D_2-type machines, (S4) is generally fulfilled. Both the simplifications regarding the machine allocation/operation sequence (S1-S3) and the simplification regarding the timing (S4) make it easier to create an efficient assignment and schedule since they significantly reduce the search space. The main task is to find an optimal assignment. An optimal schedule from this assignment can be easily obtained as follows:
Any t^B-type operation has to be started without idle time. The consequence of the simplifications is that in the "chosen subcase", when a t^C-type operation is started, we do have an appropriate finished t^B-type operation for whose result the t^C-type operation can be applied.

It means that on any D_1-type machine that carries out both types of operations, the t^C-type operations can be started immediately after inserting a setup time between its last t^B-type operation and its first t^C-type operation. Moreover, the machine (in general: any machine) that makes only t^C-type operations starts its operation just when the first t^B-type operation is finished on a D_1-type machine.

2.3.1.3 Creation of the preemptive solution

This part focuses on finding a good assignment for the chosen subcase. It is done by constructing a preemptive (optimal) assignment of the operations on the machines. (Preemptive schedule means that some machine makes some part of an operation, and then, the remaining part of this operation is continued on the same, or possibly on some other machine, without overlapping in time.)

The question is: how to partition the united set of the t^B-type and the t^C-type operations among the machines? In the preemptive solution, the finish time of the machines is aimed to be balanced. In the chosen subcase, the single D_2-type machine (in general: the D_2-type machines) gets only t^C-type operations. (In the general case - because of the preemptive solution - it can be assumed that all of the D_2-type machines get the same - possibly fractional - amount of t^C-type operations.) Let the number of the t^C-type operations on one D_2-type machine be x. Also, the α D_1-type machines that get only t^B-type operations get all the same number (y) of t^B-type operations. Finally, it is assumed that each D_1-type machine (their number is β) gets v t^B-type operations and w t^C-type operations.

The optimal values of x, y, v, w can be found easily by solving the next equation system. Recall that - following the notations introduced in Appendix B.:

- $setup(D_1, t^B, t^C)$ denotes the setup time between the t^B-type operation execution and the t^C-type operation execution of a D_1-type machine,
- $dur(t^B, D_1)$ is the processing time of a t^B-type operation on a D_1-type machine,
- $dur(t^C, D_2)$ is the processing time of a t^C-type operation on a D_2-type machine,
- $dur(t^C, D_1)$ is the processing time of a t^C-type operation on a D_1-type machine,
- $n(p_1)$ is the number of products of type p_1 (they need only the execution of a t^B-type operation but no t^C-type operation in room2),
- and $n(p_2)$ is the number of products of type p_2 (they require the execution of both a t^B-type operation and a t^C-type operation in room2).

$$\alpha \cdot y + \beta \cdot v = n(p_1) + n(p_2) \tag{2.1}$$

$$\beta \cdot w + \gamma \cdot x = n(p_2) \tag{2.2}$$

$$\begin{gathered} \gamma \cdot dur(t^B, D_1) = \\ v \cdot dur(t^B, D_1) + w \cdot dur(t^C, D_1) + setup(D_1, t^B, t^C) = \\ x \cdot dur(t^C, D_2) + dur(t^B, D_1) \end{gathered} \tag{2.3}$$

There are four variables (x, y, v, and w) and four equations. In (2.1), the number of t^B-type operations is summed up for the whole model. Equation (2.2) does it similarly for the t^C-type operations. Finally, (2.3) expresses that each machine's completion time is the same in the optimal preemptive solution. For those D_1-type machines that make both t^B-type operations and t^C-type operations (their number is β), this equation takes into account setup time. Moreover, the D_2-type machine can

start its work only after a t^B-type operation has been finished (on another machine). Its reason is that - based on the presented model - any t^C-type operation has to be preceded by a t^B-type operation for every p_2-type product.

Previously it was shown that the D_2-type machine (in general: any D_2-type machine) starts its work at time $dur(t^B, D_1)$, at the time point when D_1-type machines finish their first t^B-type operation. The other part of the assumption (S4) (i.e., t^C-type operations are made without idle times) not surely holds for any subcase (within the 5 - in more general $n(D_1) + n(D_2) + 1$ - subcases). Still, it is assumed that there will be such a subcase among all subcases, for which this part of the assumption also holds. In fact, this optimal subcase will be the chosen subcase. It implies the next assumption:

(S5) The solution of the above equation system (2.1)-(2.3) is composed of only nonnegative values, i.e., $x, y, v, w \geq 0$.

Reasoning: If at least one variable is negative, that subcase is not considered (it cannot be the chosen subcase). The negative value of a variable happens, for example, if there are too many D_2-type machines and too few D_1-type machines. In this case, equation (2.3) cannot hold with non-negative values of the variables. However, in the case of the best possible subcase, the assumptions are assumed to be valid.

Appendix D presents an example of the calculation of the preemptive solution of a small example. In that example, all the obtained variables are non-negative; however, their values are fractional. Only integer number of tasks can be assigned to the machines, so this result has to be modified. It is done by rounding.

2.3.1.4 Rounding

In the rounding procedure, a non-preemptive solution is created from the preemptive one. The resulted solution is no longer an approximation but a real solution to the problem. For getting this solution, rounding is applied with the following steps:

Step 1: Truncate all variable values (round them down).

Step 2: Starting from the first D_1-type machine progressing one-by-one forward to the $n(D_1)^{th}$ D_1-type machine, increase the number of the t^B-type operations assigned to the machine by one until the required number of the assigned t^B-type operations $(n(p_1) + n(p_2))$ is reached.

Step 3: Starting from the last $(n(D_2)^{th})$ D_2-type machine progressing one-by-one backward to the first D_2-type machine, increase the number of the t^C-type operations assigned to the machine by one until the required number of the assigned t^C-type operations $(n(p_2))$ or the first D_2-type machine is reached. If the first D_2-type machine is reached and there remains an unassigned t^C-type operation, continue the increase of the number of the assigned t^C-type operations by one on the D_1-type machines that make both t^B-type operations and t^C-type operations (there are β pieces of them), starting with the last machine with this type, towards the first one.

The rounding procedure is demonstrated on the small problem of Appendix D in Appendix E.

2.3.1.5 Improvement with Metaheuristics

After the rough job-assignment for the subproblem of room2 is determined, the result's improvement is attempted by applying a metaheuristic method. For judging whether the improvement was successful or not, the evaluation of the job-assignment is required both before and after applying the metaheuristics. For this evaluation, the schedule has to be created from the assignment.
Based on the introduced simplifications and assumptions - that always results in a feasible job-assignment - a schedule can be obtained in the following simple way: the t^B-type operations are executed as early as possible. The resulted half-ready products are stored in a virtual warehouse. Then any t^C-type operation is started as early as possible, suppose that there is a free machine that can start its execution and there is a half-ready product that waits for a t^C-type operation in the virtual warehouse. By applying this simple (greedy) method, a feasible schedule is obtained from the given assignment. The makespan of the schedule is used for the evaluation of the schedule.
In the Improvement phase, two different approaches were investigated. One of them is a Local Search (LS) that simply moves or swaps operations, and the other one is Tabu Search (TS) that is based on the paper of Hurink et al. [1].

Local Search

The first version of the Improvement phase - whose input is the job-assignment that is resulted by the rounding procedure - uses the following two elementary steps:

- MOVE: A t^B-type operation or a t^C-type operation assigned to a certain machine is picked and moved to another machine.
- SWAP: A t^B-type operation assigned to a certain machine is swapped with a t^C-type operation of another machine.

Two MOVE steps are illustrated in Figure 2.4 by arrows. They are applied - one-by-one - on the small example of Appendix E, after the rounding procedure. The resulted schedule is presented in Figure 2.5. It can be seen that the schedule was improved by Local Search since the makespan was decreased by 7 units.

When MOVE or SWAP is applied, the machine-related constraints are taken into account: e.g., if it was already decided that some machine will not perform t^B-type operations, this machine cannot get t^B-type operation in the Improvement stage neither. Two different approaches are applied for managing the local changes. One of them is the directed way, i.e., the machine with the maximum completion time is chosen, and in the case of the "MOVE" elementary step, an operation from this machine is moved to another machine. The second approach is to choose the operations randomly.
If the schedule's makespan after applying an elementary step of LS is better (lower)

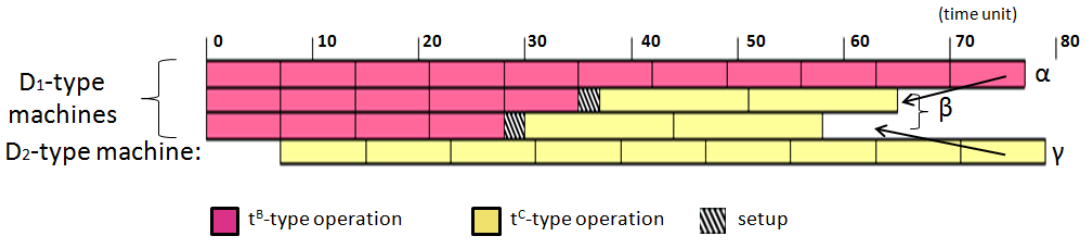


Figure 2.4. The two "MOVE" steps that are applied to the small problem of Appendix E - example for Local Search.

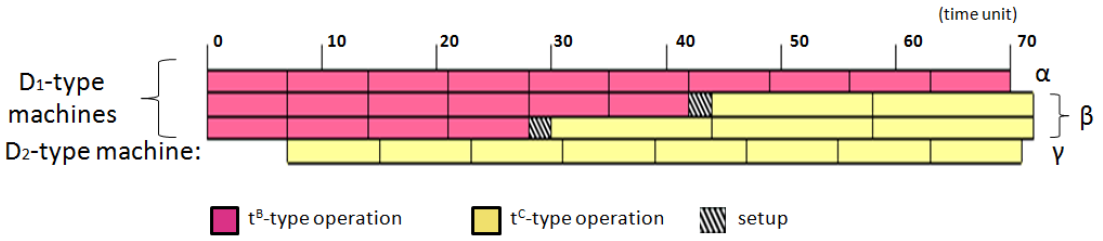


Figure 2.5. The resulted schedule for the small problem of Appendix E, after the two "MOVE" steps of the Local Search. The makespan decreases from 79 to 72.

than that of the initial schedule, the local change is accepted. Otherwise, it is rejected. The local changes are performed several times, where the number of trials is prescribed in advance.

Tabu Search algorithm

The second version of the improvement is the Tabu Search algorithm. In its implementation, the ideas of Hurink et al. [1] were followed: the reposition of some operations of some blocks of the critical path. The related concepts and the basics of TS by Hurink et al. are summarized in Appendix F.
There can be more than one critical path in the graph of the schedule; Figure 2.6 shows an example. This illustration is simplified for better understanding; the total number of processes (workflows) are reduced to 14, the labels on the operations identify the process they belong to, and operation times and setup times were chosen randomly.

The implemented Tabu Search algorithm starts from an initial solution (it is the rough job-assignment obtained after the rounding procedure), just like the Local Search. At most, 20 *valid neighbours* are generated in each iteration from at most 100 attempts (its method is detailed in Appendix F), and the best one - the one with the minimal makespan - is selected at the end of the iteration.

Definition 55. *Valid neighbour: A neighbour of an instance (according to the neighbourhood property of Hurink et al., that is described in Appendix F) that is (1) feasible, (2) is different from the starting instance, and (3) is not an element of the tabu list.*

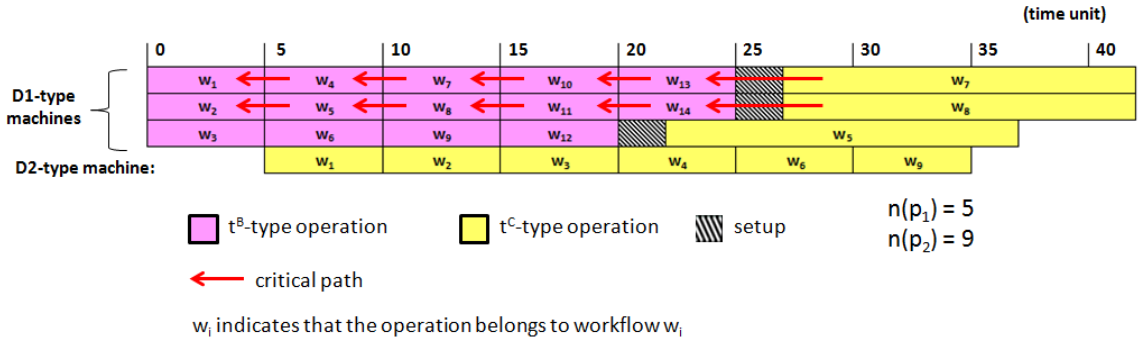


Figure 2.6. Example of two critical paths in a schedule.

The content of the tabu list is stored in the format of sequences of the allocated operations. The output solution of an iteration (the best valid neighbour of the 20 schedules) is added to the tabu list, avoiding to select it again shortly. The Tabu Search algorithm stops when it reaches the 100^{th} iteration, and the best of the found solutions is considered the solution of room2. The structure of the implemented Tabu Search algorithm is illustrated in Appendix G.
The instances (and their neighbours) are in the format of allocated operation sequences; however, they have to be scheduled to calculate their makespan. An allocated operation sequence defines the schedule, following the scheduling method presented at the beginning of this subsection.
It has to be highlighted that contrary to the problem of Hurink et al. [1] - and to other papers in this topic -, here, a t^C-type operation is paired with a t^B-type operation only in the scheduling phase. During the phase of neighbour creation, all the t^B-type operations are equivalent. This is the case for the t^C-type operations, too. It means that the operation-process pairs (which process an operation belongs to) are determined only in the scheduling step. Because of that, the instance often remains the same after moving an operation from a block. An example is shown in Figure 2.7, where a schedule with only one block (constructed by the t^C-type operations labelled by w_1, w_2, w_3, w_5 and w_6) is presented. If any operation of this block is moved to the beginning or to the end of the block, the obtained operation sequence will be the same as the initial one was. In the case of this kind of block, the neighbours differ from the initial instance only if they are created by moving the selected operation to another machine. My algorithm handles it and accepts only neighbours that differ from the instance under consideration.

Comparison of the applied Metaheuristics

Two alternative approaches were applied in the Improvement phase of the heuristic algorithm: the first one was Local Search (with "MOVE" and "SWAP" operations), and the other one was Tabu Search. Originally, following Hurink et al., the Tabu Search algorithm was implemented without keeping (S1) and (S2) simplifications. It means that such an operation sequence of a machine can exist, where a t^C-type operation precedes a t^B-type operation. Moreover, more than one setup time

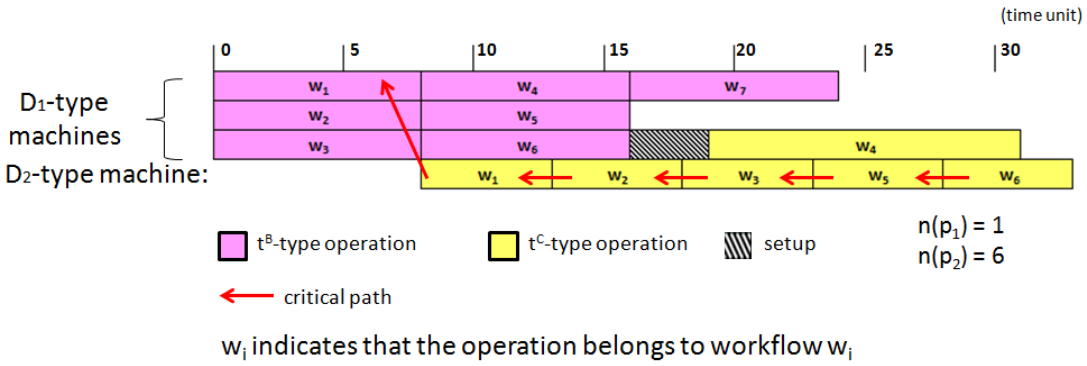


Figure 2.7. Example of a useless move of an operation to the beginning or to the end of its block.

can occur on any machine. The only restriction regarding the operation sequence is that it has to result in a feasible schedule, i.e., a matching preceding t^B-type operation has to be found for every t^C-type operation. Since the Local Search method holds both (S1) and (S2), for the sake of comparability, first, Tabu Search was also implemented with the simplifications (S1) and (S2) [137].
Another modification of the original Tabu Search algorithm was related to the selection of the operation for reposition. In the original version, only a block-element can be a candidate for moving. However, in several cases, the reposition of a standalone operation also can improve the schedule, see Figure 2.8, for example. In the figure - for the sake of better illustration - the total number of processes is reduced to 6, the labels on the operations identify the process (workflow) they belong to, operation times were chosen randomly, and setup times were chosen 0. If the initially assigned operation sequence was the one in the right upper part of the figure, it is worth moving the critical path's last operation; even it is not an element of any block.

That is why a new version of the Tabu Search algorithm was also implemented, where any operation of the critical paths can be selected to move. This variant was created both with and without (S1) and (S2) restrictions.
The results obtained by all 4 different versions of the Tabu Search algorithm were compared with the results of the Local Search algorithm. In this investigation, the size of the tabu list was varied between 10 and 100. The executions were made for 1000 random instances; these instances are presented in detail in subsection 2.4.1. The results are illustrated in Figure 2.9.

Figure 2.9 (a)-(d) shows the improvement and the number of occurrences of the different improvements when Tabu Search is applied instead of Local Search. Here, improvement means the shortening of the whole schedule's makespan (not only after scheduling room2 but when finishing room3, too). For example, when Tabu Search allows the movement of operations only from the blocks of the critical path(s) and (S1) and (S2) simplifications hold, then in the case of 70-80 instances, the shortening of the makespan is 1 unit, there are also cases with 6 units of improvement, however, in one case a worsening (increase) of the makespan with 8 units is experienced (see Figure 2.9 (c)).
It is useful to change Local Search to Tabu Search since there are much more in-

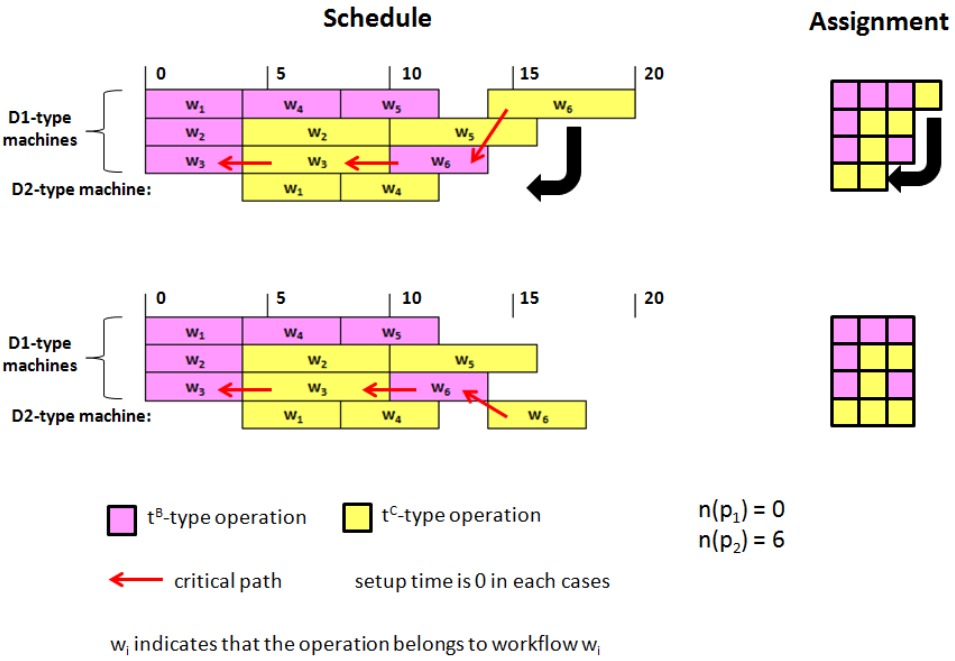


Figure 2.8. An example of why it can be useful to move not only the block elements but standalone operations of a critical path, too.

stances on the positive side of all the 4 figure-parts than on the negative side. However, in most cases, the same makespan is obtained (it means that the improvement is 0 unit in the highest number of the 1000 cases). Since the sum of the number of the instances presented in any figure part is about 100-280, in the cases of the remaining 720-900 instances, the application of both Local Search and Tabu Search in the Improvement phase of the heuristic algorithm results in the same makespan in the end. The improvement happens in higher number when (S1) and (S2) hold, e.g., a t^C-type operation can not precede a t^B-type operation (so, there is at most one setup per machine); however, this case results in a small increase of the number of worse results, too. This result confirms the assumption that in most cases, the higher freedom in operation sequence may abduct from the optimum, it is worth to execute t^C-type operations only after the t^B-type operations and to minimize the need for setup time. Another conclusion is that if the move of any operation of the critical path(s) is allowed (so, it is not restricted only to the block-elements) in the Tabu Search, then a small increase in the improvement can be reached, but the measure of worsening also can increase a lot (even 12 units of worsening of the makespan is experienced). Finally, the figure-parts show that tabu list size 70 gives the best result; however, there are not high differences from the surrounding values. In the case of a short tabu list, the performance decreases (most likely because it lets frequent repetition of the same schedules), just like when the tabu list is long (the reason for this latter case can be that the algorithm sticks in a local optimum). The conclusion is that the best result is obtained when Tabu Search is used with tabu list size 70 in the Improvement phase of the heuristic scheduling algorithm instead of Local Search, the constraints regarding the operation sequence (S1) and

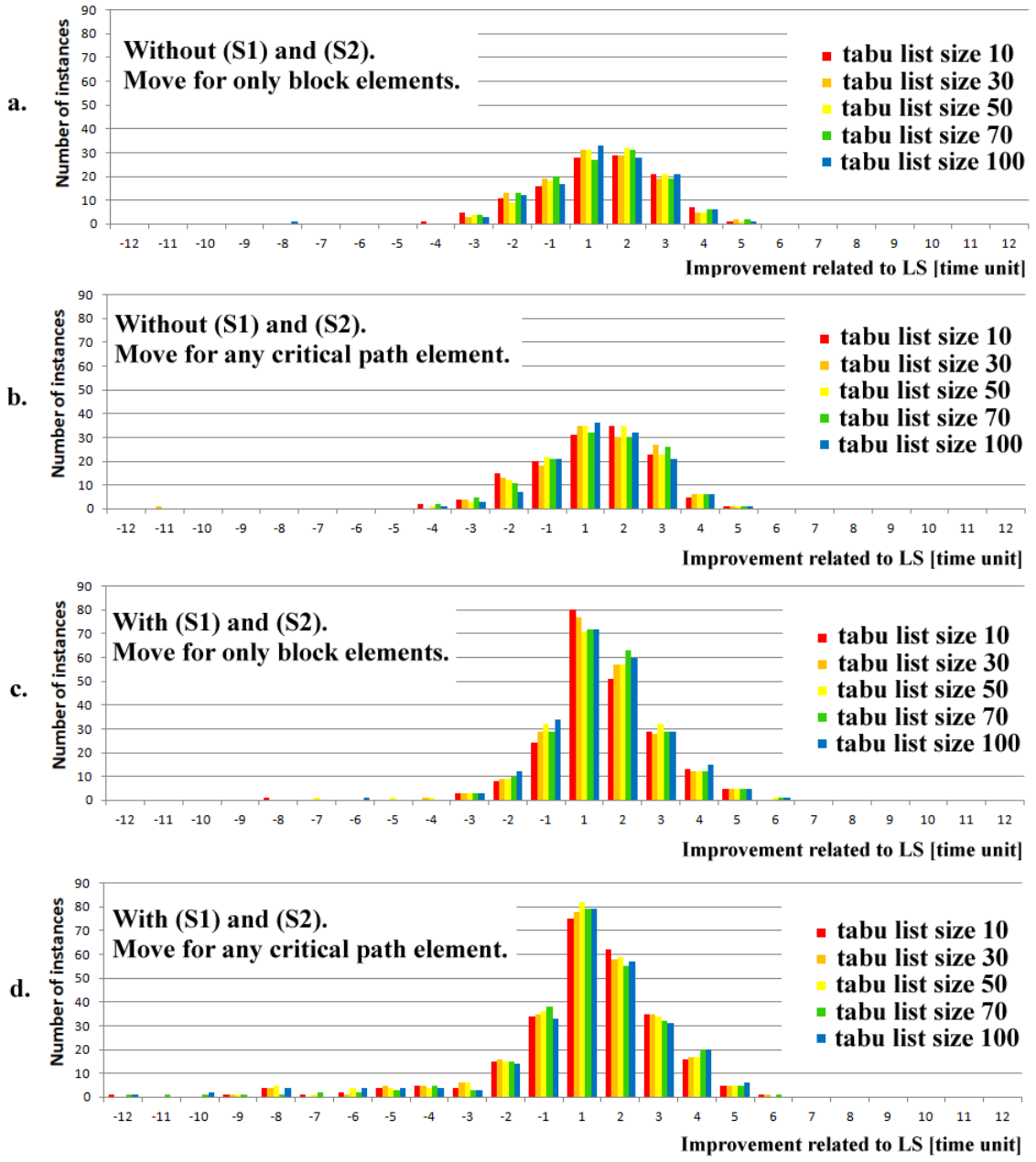


Figure 2.9. Comparison of the four implemented variants of Tabu Search with Local Search.

(S2) are kept, and only operations that are elements of a block of a critical path are allowed to move.

2.3.1.6 Summary of the heuristic algorithm

As the heuristic algorithm's detailed presentation showed, the main idea was to divide the initial problem into three subproblems (room1 - room2 - room3). The subproblem of the first room was solved optimally by a simple greedy method. The ready products from room1 are then transported into a virtual warehouse (the cost of

the transportation was not dealt with in the problem). Any product was transported as soon as it was ready.
Then the subproblem of room2 was considered. Simplifications were made, and the subproblem was divided into several $(n(D_1) + n(D_2) + 1$, in case of the concrete problem 5) subcases. A preemptive optimal solution was calculated for any subcase (if there was at least one negative variable in a solution, that subcase was excluded from the subsequent considerations, all the other subcases remained). Then, by applying a rounding procedure, a rough non-preemptive solution was obtained for the subcases. Finally, the solutions were attempted to improve by performing a metaheuristic method. During this method, schedules had to be calculated from the job-assignments. Afterward, the solutions of all the subcases that remained in the competition were compared, and the best one was chosen (it was the moment when the "chosen subcase" was determined). The ready items of room2 were also transported from room2 into the virtual warehouse.
In room3, the assembly of the products of room1 and the products of room2 takes place by another greedy method. As soon as a product of room1 and also a product of room2 is ready, and a machine of room3 is free, the two parts are assembled, resulting in a final product.

2.4 The efficiency of the heuristic-aided scheduling

To analyze the efficiency of the developed heuristic algorithm, its result has to be compared with the real optimum of the problem on a wide variety of input values and parameters. Exact solvers provide the optimal solution to a problem if they can solve the problem in a reasonable time. For this purpose, the Mixed-Integer Linear Model of the scheduling problem of this chapter was created. It is presented in Appendix H. Instead of creating the MILP model, other approaches could also have been applied. For example, for a CP model of the problem, the IBM ILOG CP Optimizer is supposed to efficiently provide a solution; applying the Optimizer's efficient interval variables and the related tools and functions, instead of the "traditional" arithmetic constraints [47]. However, here, the goal is first to give and analyze a heuristic algorithm, the determination of its quality, and the analysis of its interplay with an arbitrary exact solver. Here, CPLEX was used to solve the MILP model of the problem.
The computation time an exact solver requires increases highly with the complexity of the problem. The search space can be narrowed with the help of the result of the heuristic algorithm. This section shows the efficiency of the support offered for the exact solver by the developed heuristic algorithm.

2.4.1 The method of the evaluation

For getting the exact solution to the problem that is introduced in Appendix H, IBM ILOG CPLEX 12.7.0.0 was run as the exact MILP solver. The solver requires the time horizon $\mathcal{T}$ as an input value. When the time horizon is far from the optimum of the problem, the search space is huge, and the exact solver requires unreasonable time to find the optimal solution. That is why a common way to solve a complex

problem is the support of the exact solver by a near-optimal solution (that defines the time horizon), that is obtained by a much quicker algorithm, e.g., by a heuristic method (for details, see subsection 2.1). It narrows the search space, thus decreases the computation time significantly.
Usually, when an exact solver is applied, a feasible solution is found soon.

Definition 56. *Gap: The difference between the objective value of the feasible solution and the lower bound.*

The running can be stopped if the feasible solution is satisfying (the gap is small enough), or the solver can be enforced to close the gap completely, i.e., the execution carries on until a proven optimal solution is found. I intended to know the optimal solution to the problem.
When the optimum is obtained, the heuristic algorithm's quality can be judged by comparing its output with the optimal solution. Moreover, by providing the solution obtained by the heuristic algorithm to the exact solver as its time horizon, the question "How the performance of the exact solver can be improved by the solution that is provided by the heuristic solver?" also can be answered for the presented scheduling problem.
For the sake of the comparison, five different methods are applied. In the first two of them, the standalone execution of the exact solver is performed (without the help of the heuristic solver), and the other three variants use the result of the heuristic algorithm to help the exact solver.

Method1: starting at a coarse upper bound and decreasing it by one, iteratively. The upper bound approach utilizes an estimation of $\mathcal{T}$ from above. Using this upper bound, it is decreased one by one until there is not a feasible solution. In this way, the variable space is large enough to host an optimal solution. However, a coarse estimation leads to large instances and results in long solution times for the MILP solver. The calculation of the coarse upper bound is presented in Appendix I. The estimation is based on the idea that all the resources in one room operate in parallel, but the next room's operations start only after all jobs in the previous room are finished. This bound can be computed directly from the given input data in $O(1)$, but it turned out to be rather sloppy.

Method2: starting at a coarse lower bound and increasing it by one, iteratively. The lower bound approach estimates $\mathcal{T}$ from below. The calculation of the coarse lower bound is described in Appendix I. In this estimation, an idealized situation is considered where all machines can work in parallel in all rooms, and there is no setup time between the processes. In general, setting up the model with such value for $\mathcal{T}$ will lead to an infeasible MILP. It turned out that the solver is quite fast in detecting this infeasibility. Then $\mathcal{T}$ is incremented by 1, and this whole procedure is iterated until a feasible (thus optimal) solution is found.

Method3: starting at an upper bound that is provided by the heuristic and decreasing it by one, iteratively. The mixed-integer model is set up with a time horizon of $\mathcal{T}^{\text{heu}} = makespan(W)$, where $makespan(W)$ is the makespan of the

solution that is provided by the heuristic algorithm for the problem, whose process set is W (see the notations that are introduced in Appendix B). So, the exact solver gets the feasible solution provided by the heuristic solver as a starter. Then, this upper bound is decreased by one, iteratively, and the solver stops when no feasible solution is found. The last feasible solution is optimum.

Method4: starting at an upper bound that is the result of the heuristic minus 1 and decreasing it by one, iteratively. In this upper bound approach, the time horizon is set to be $\mathcal{T}^{\text{heu-1}} = makespan(W) - 1$, where $makespan(W)$ is the makespan of the solution that is provided by the heuristic algorithm for the problem, whose process set is W (see the notations that are introduced in Appendix B). In this approach, it can happen that no feasible solution is found in the first step; it means that the solution provided by the heuristic solver is optimal (this method further reduces the burden for the MILP solver). Otherwise - like in the other upper bound approaches - the upper bound is decreased by one, iteratively, until optimality.

Method5: starting at a coarse lower bound and applying logarithmic search in the interval bounded by the result of the heuristic from the upper side. This is a variant of the lower bound approach: instead of increasing the lower bound of the exact solver by one, iteratively, starting at the initial coarse lower bound and stop when a feasible solution is found (like it was done in $Method2$), the increase of the lower bound is done by a logarithmic step size (getting $\mathcal{T}^{lb_{log}}$), i.e., taking into account the makespan of the result of the heuristic solver as an upper bound.

The role of both the upper bound and the lower bound is the estimation of the exact solver's time horizon. The approach of the optimal solution's $\mathcal{T}$ value is started from their value. Of course, these estimations do not influence the value of the final solution (the optimum), but they significantly influence the required computations, and thus, on the speed of the exact solver.

To analyze of the quality and the supporting capabilities of the result of the heuristic algorithm, a test set of 1000 random instances was generated. Their parameters were chosen according to a uniform distribution over the integers in the intervals shown in Table 2.1.

2.4.2 The results of the evaluation

First, both the heuristic algorithm and the exact solver were separately executed for the 1000 random instances. Figure 2.10 illustrates the comparison of their results, where the makespan obtained by the heuristic solver is drawn by blue color, and the optimal makespan resulted by the exact solver is shown by red color. The makespan is illustrated on the figure's vertical axis, and the index of the instance is shown on the vertical axis after the 1000 random instances were ordered on their optimal makespan into an increasing sequence.

Table 2.1. Intervals for random data.

name	interval	name in the MILP model	constraints
$dur(t^A, D_3)$	[2,6]	$p_{r_5}^{t^A}, p_{r_6}^{t^A}$	$p_{r_5}^{t^A} = p_{r_6}^{t^A}$
$dur(t^B, D_1)$	[4,8]	$p_{r_1}^{t^B}, p_{r_2}^{t^B}, p_{r_3}^{t^B}$	$p_{r_1}^{t^B} = p_{r_2}^{t^B} = p_{r_3}^{t^B}$
$dur(t^B, D_2)$	[9,13]	$p_{r_4}^{t^B}$	
$dur(t^C, D_1)$	[10,15]	$p_{r_1}^{t^C}, p_{r_2}^{t^C}, p_{r_3}^{t^C}$	$p_{r_1}^{t^C} = p_{r_2}^{t^C} = p_{r_3}^{t^C}$
$dur(t^C, D_2)$	[5,7]	$p_{r_4}^{t^C}$	
$setup(D_1, t^B, t^C)$	[0,5]	$u_{r_1}, u_{r_2}, u_{r_3}$	$u_{r_1} = u_{r_2} = u_{r_3}$
$setup(D_2, t^B, t^C)$	[0,5]	u_{r_4}	
$dur(t^D, D_4)$	[2,4]	$p_{r_7}^{t^D}, p_{r_8}^{t^D}$	$p_{r_7}^{t^D} = p_{r_8}^{t^D}$
$n(p_1)$	[30,70]	$n(p_1)$	
$n(p_2)$	[30,70]	$n(p_2)$	$n(p_2) = 100 - n(p_1)$

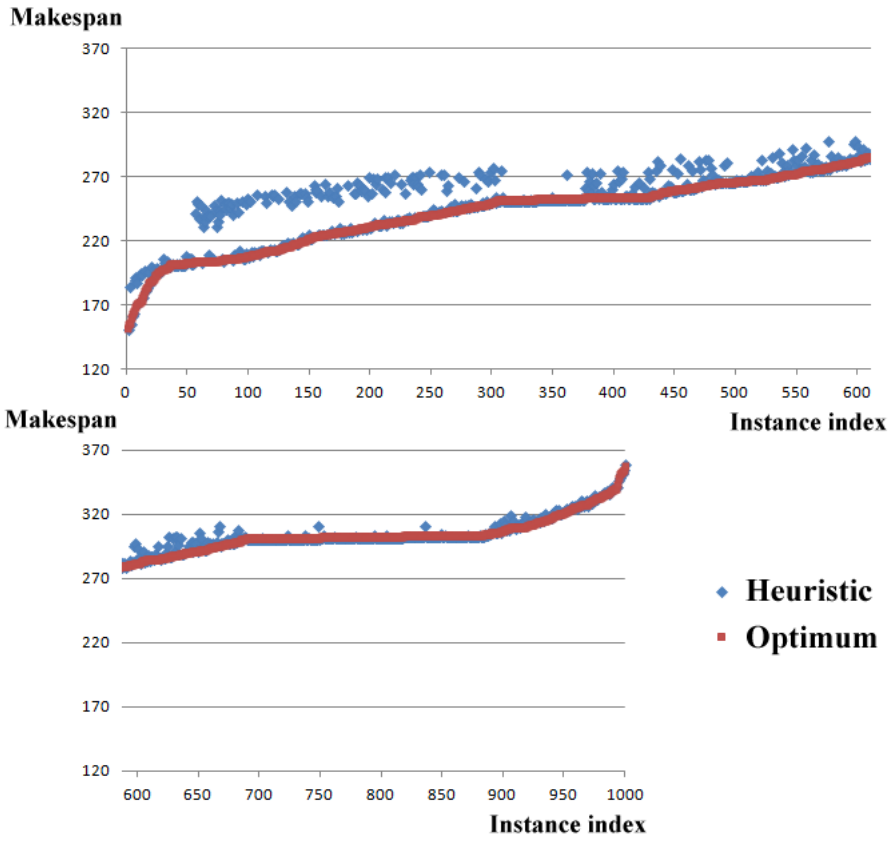


Figure 2.10. The comparison of the makespans obtained by the heuristic (blue) and the exact solver (red) for the 1000 random test instances.

Based on the figure, an interesting fact can be noticed. Consider the intervals where the change of the optimal makespan of the successive instances is close to zero (instances 305-365 and instances 685-889). Here, the heuristic solution was almost always optimal. This is not the case in other instances. Let the remaining

instances be separated into three parts: the first part is where the instance's index is at most 304, part 2 is in the middle section, where the indexes are between 366 and 684, while the third part is where the indexes are higher than 889. For these parts, the slope of the graph of the optimal makespan significantly differs from 0. The heuristic scheduler performed the worst in the first part; it is much better in the middle part and almost optimal in the third part. Its reason can be that the heuristic algorithm cannot correct a prior wrong decision if the optimal makespan is relatively small. When the makespan is relatively high (its reason can be the higher number of desired products), the heuristic scheduler has some opportunity to correct the wrong decisions later.

Figure 2.11 shows the number of the instances as the function of their relative error (the difference between the makespan obtained by the heuristic solver and the makespan of the optimal solution is divided by the makespan of the optimal solution). It can be seen that in about half of the cases, the heuristic solver found the optimum, and in 80% of the cases, the relative error is at most 2%. Moreover, the relative error is at most 5% in 95% of the cases. Based on these results, it can be stated that the heuristic algorithm is efficient.

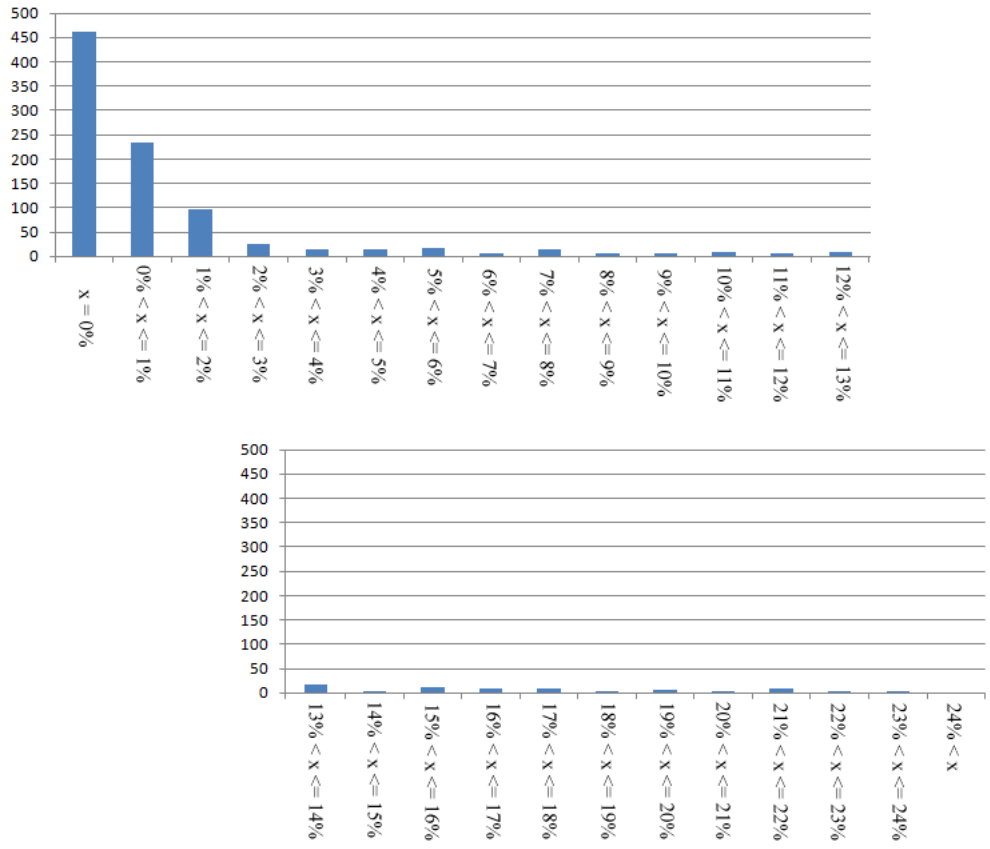


Figure 2.11. Relative error distribution of the heuristic scheduler. Horizontal axis: relative error (x) of the heuristic solver, vertical axis: number of the cases.

Besides the quality of the heuristic solver's solution, the speed of the algorithm related to the exact solver was analyzed, too. The heuristic solver's computation

time was short: 2 minutes for the 1,000 instances (including the generation of the output text file that contained the solution - later, it was used by the exact solver). That means 0.12 seconds for one instance on average. The exact solver's calculation usually took more time: there were cases when 10,000 seconds were not enough for scheduling one instance.

During the analysis of the efficiency of the support that the heuristic solver can give to the exact solver, the exact solver's computation times - applying the methods introduced in subsection 2.4.1 - are compared. It has to be noted that a 10,000 seconds time limit was applied: in all the illustrated cases, the 10,000 seconds computation time means that in those cases, the exact solver did not find the optimum in 10,000 seconds. The calculations were run on a 2014 MacBookPro with a 2.8 GHz Intel Core i7 CPU and 16 GB 1600 MHz DDR3 RAM. In Figure 2.12, the horizontal axis shows the computation time (in seconds) required by the exact solver with the upper bound that was set to be the makespan obtained by the heuristic solver ($Method3$). The vertical axis represents the exact solver's runtime with the coarse lower bound approach ($Method2$). Points above the red line mean that the CPLEX with the aid of the heuristic solver's result beats the coarse lower bound method, and points below the red line are those where the coarse lower bound based method is better. As the figure shows, the heuristic solver's result can be closer to the optimum than the coarse lower bound - especially in the cases with lower runtime. (The correlation of the runtime of the two methods fits the best onto the function $y = 104.9 * x^{0.2805}$, where x is the runtime of $Method3$ and y is the runtime of $Method2$. It means that for $x < 643$, $Method3$ works faster on average.)

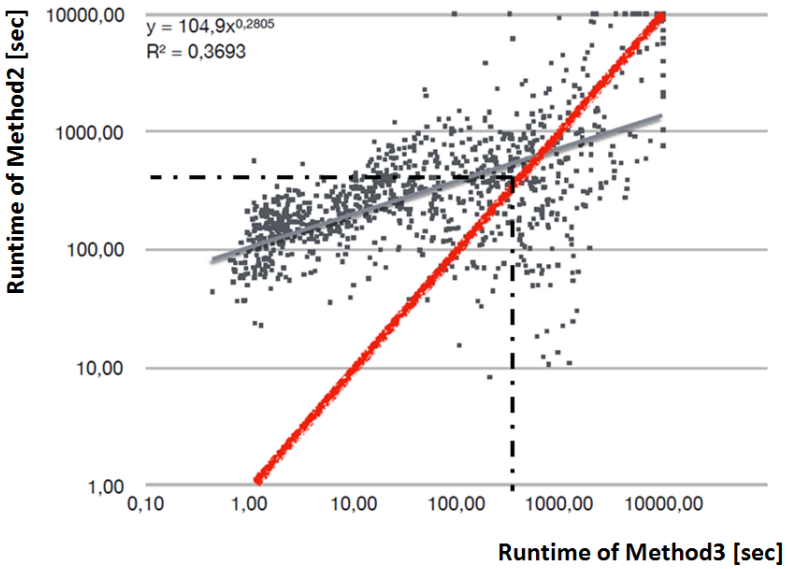


Figure 2.12. Comparison of computation time required by Method3 (heuristic-aided) and Method2 (coarse lower bound) approaches.

In judging whether $Method3$ (the makespan obtained by the heuristic solver was set as the upper bound of the exact solver, then going down step-by-step) or $Method4$ (the makespan obtained by the heuristic solver minus 1 was set as the upper bound

of the exact solver, then going down step-by-step) works quicker, Figure 2.13 helps. This figure shows the correlation between the run of $Method4$ (on the horizontal axis) and $Method3$ (on the vertical axis) on a double-logarithmic scale. A good fit is the function $y = 1.1424 * x^{0.9769}$. This function returns the average expected time y for the "makespan-aided run" (that is, CPLEX using the result of the heuristic solver as the starter), given the time for the "'makespan minus 1'-aided run" as x. For $x < 300$, $Method4$ works faster; for $x > 300$, $Method3$ is quicker. From the experiments, it is concluded that there is no significant difference between them. Its reason can be that the heuristic scheduler found the optimum often.

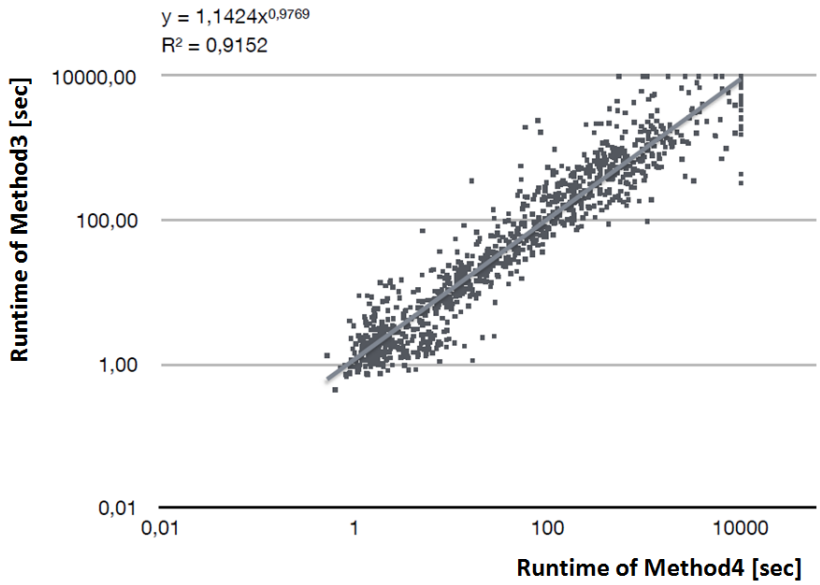


Figure 2.13. Comparison of computation time required by Method4 ("'makespan minus 1'-aided") and Method3 ("makespan-aided") approaches.

The applied lower bound approaches also were compared. Figure 2.14 shows the computation time required by $Method5$ (the strategy where the lower bound was incremented by a logarithmic step size) compared to the computation time required by the simple one-by-one increasing method ($Method2$). The runtime of $Method5$ is illustrated on the vertical axis, while the horizontal axis represents the computation time required by $Method2$. The Figure expresses that - as it was expected - it makes sense to apply the logarithmic step size instead of step size 1 in the increase of the lower bound. It means that the makespan of the schedule provided by the heuristic algorithm is closer to the optimum than the coarse lower bound since in the method that uses logarithmic step size, the application of the result obtained by the heuristic solver is included.

The vertical axis of Figure 2.15 shows the computation time for $Method3$ (where the makespan obtained by the heuristic solver was set as the upper bound of the exact solver) and the runtime of $Method5$ (the strategy where the lower bound was incremented by a logarithmic step size) is illustrated on the horizontal axis. As one could expect, the logarithmic search beats Method3, too.

When comparing the results obtained by $Method3$ (the makespan obtained by

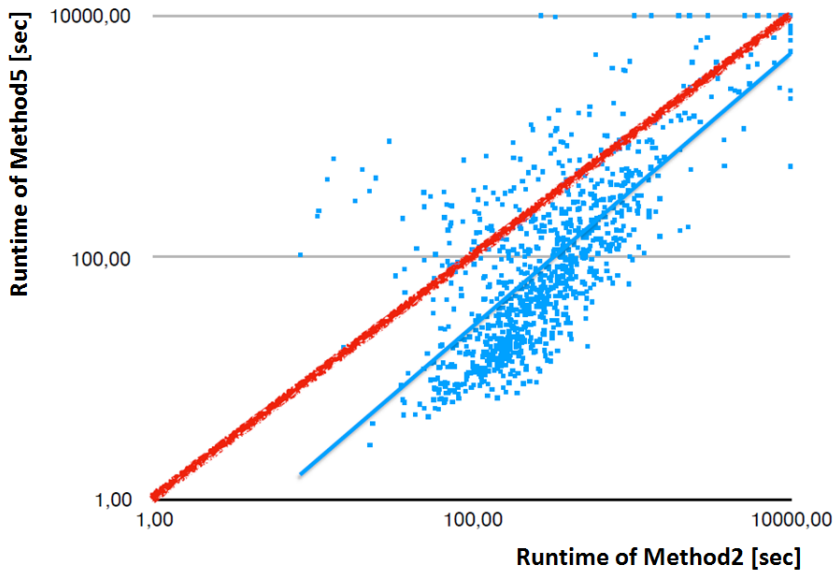


Figure 2.14. Comparison of computation time required by Method2 (lower bound with "one-by-one increment") and Method5 (lower bound with "logarithmic increment") approaches.

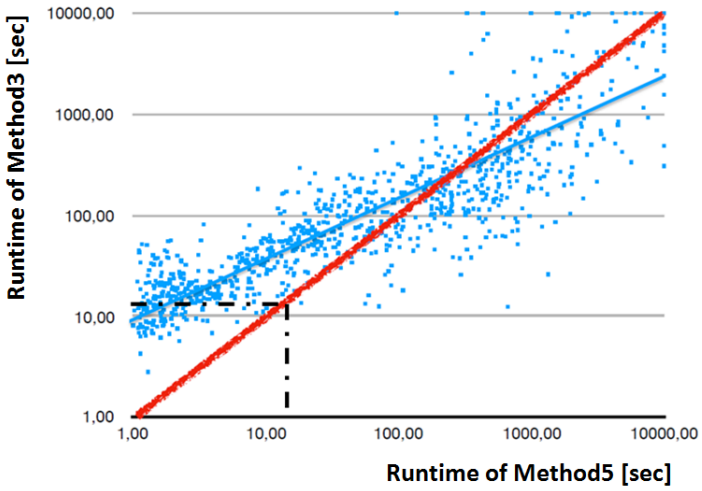


Figure 2.15. Comparison of computation time required by Method5 (lower bound with "logarithmic increment") and Method3 (upper bound by the makespan resulted by the heuristic solver) approaches.

the heuristic solver was set as the upper bound of the exact solver, then going down step-by-step) with the results obtained by *Method*1 (when the exact solver was started at a coarse upper bound and the time horizon was decreased one-by-one), of course, *Method*1 requires at least as much computation time as *Method*3, for every instance.
Finally, it also can be observed in Figure 2.12 and Figure 2.15 - bordered by the broken lines - that in case of short computation times, the heuristic-aided exact solver (where the heuristic algorithm provides the upper bound) performs better

than the exact solver started from a coarse lower bound, however, the logarithmic search case (also started from the coarse lower bound) beats the exact solver that starts from the upper bound provided by the heuristic algorithm. Of course - as it was mentioned earlier - the heuristic algorithm also has a role in the logarithmic search case.

2.5 The contribution of this chapter

The contribution of this chapter has three main parts: (1) it suggests an approach for solving a scheduling problem by a heuristic, (2) it gives the quality of the developed heuristic in case of a concrete problem, and (3) it determines the efficiency of the heuristic in aiding an exact solver in case of the concrete problem.
It was experienced that the separation of the initial problem into smaller subproblems and the heuristic algorithm that was applied on the not trivial subproblem combining the following steps:

- exclusion of setup times if possible,
- pick an appropriate subcase,
- creation of a preemptive solution,
- rounding,
- improvement,

were able to support the exact solver significantly. It can be seen that the smart combination of the heuristic method and the exact solver is very efficient and fast.

2.5.1 The concerned problem class and the efficiency of the applied method

The presented approach can be applied for any scheduling problem that has the following properties:

- there is a predefined arbitrary set of resources; each resource can perform at most two different operation types (this constraint is because of the applied simplifications),
- there is a predefined arbitrary set of operations,
- there is a predefined arbitrary set of processes, each of them can be described by an acyclic directed graph (allowing forks and joins) where each node is an operation,
- any process can be performed several times (mass production is assumed),
- the operation time depends on both the operation and the applied resource,
- operations are uninterruptable,

- for any operation, there is a non-empty subset of the resources that contains machines that can perform the operation,
- for each operation, exactly one resource is assigned,
- setup time can be applied,
- operation times, setup times, and the number of the required products (the number of the execution for any process) can be arbitrary,
- the goal is the minimization of the makespan.

After dividing the initial problem into subproblems, the presented steps of the heuristic scheduler can be applied. However, the investigation of the separability of complex workflow systems and the separation method into smaller sub-problems is not trivial. It requires further research. Briefly, the presented heuristic can be applied to any scheduling problem that can be described by the problem model of Appendix B and keeps the following constraints: (1) any resource can perform at most two different operation types, (2) there is no transportation operation in the process, where the resource has to go back to its initial place after performing the operation (the $dur()$ and $durRe()$ functions of the model of Appendix B are identical) and (3) the problem is separable.
The efficiency of the heuristic was investigated for one model structure (see in Figure 2.2) through 1000 different parameter settings. The relative error was 0.37% in general (averaging the relative error for the 1000 random instances, based on Figure 2.11), and the computation time was significantly short, related to the exact solver. Finally, the heuristic algorithm's result was able to help the exact solver efficiently by reducing the computation time required to find the optimum (see subsection 2.4.2).

Chapter 3

Presentation of the Genetic Scheduler

In this chapter, a genetic algorithm is presented and analyzed that was created for solving an extended variant of the scheduling problem of the previous chapter. Thus, the problem of this chapter is more complex than the problem of the previous chapter.

3.1 The extended problem model

The extension means longer processes that include additional operations. These operations are transportation tasks that are performed by special means of transportation. These resources change their places/positions, and after a transportation resource finishes an operation, it has to get back to its start place/position before it can begin a new operation. It requires setup time; however, this setup time is applied after each operation of a transportation resource, contrary to another kind of setup time applied in the model. This latter kind of setup time - which was also part of the previous chapter's model, is an "operation sequence"-dependent setup time. In the followings, the phrase "setup time" is applied only to the "operation sequence"-dependent setup time, and the transportation-related setup time is referred to in another way (e.g., "the duration after a resource is ready again") for the sake of being able to distinguish the two phenomenons.
The problem of this chapter is illustrated in Figure 3.1. A transportation operation is inserted between each successive operation of the previous chapter's problem and besides both to the beginning and the end of each process. Each transportation operation has its own machine type, and one (half-ready) product belongs to each transport (like, e.g., in the case of robotic arms). Moreover, there is only one resource of each transportation resource type. The other features of the problem remain, and the objective is still to find the schedule with minimum makespan.

The problem is still an extended FJSP problem with multi-purpose machines that realizes mass production. The model of the problem contains assembly operations and allows the operations of the same job to be processed in parallel. All the comparisons with the model of other scientific papers presented at the end of subsection 2.2 are valid here, too.

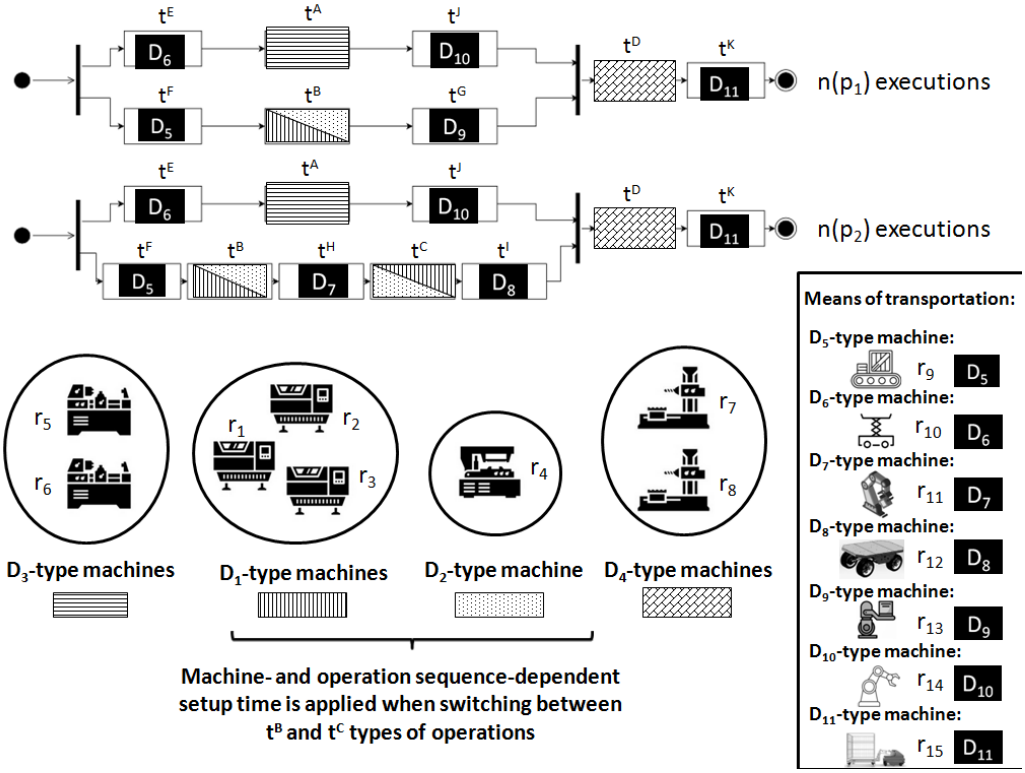


Figure 3.1. The production processes, the resource set, and the resource allocation possibilities of the extended scheduling problem.

Appendix J contains the formal description of this problem, using the notations of Appendix B.
I developed a genetic algorithm (GA) to solve this scheduling problem. The setting of the parameters of the algorithm was performed on a problem variant with predetermined parameter values. After that, the efficiency of the algorithm was investigated on random parameter valued variants of the problem. The obtained results were compared with other approaches' results (coarse lower bound and simulated annealing). The algorithm and the results were published in [90]. The efficiency of the GA was investigated on some benchmark FJSP problems. Because of the specificity of this thesis's problem, the benchmark problems from the scientific literature lack some properties that the GA can handle. Contrary to the benchmark problems, the GA was developed to solve mass production problems and can handle processes that contain parallel branches, deals with transportation operation and setup time, too.

3.2 Genetic algorithms, genetic operators

In this chapter, a genetic algorithm is presented that was developed to solve the complex scheduling problem of subsection 3.1. Here, the simplifications of the heuristic algorithm of Chapter 2 are released, and a popular metaheuristic method - a genetic algorithm - was chosen to solve the problem. Both the elements and the parameters of the GA were developed for this special problem.

The structure and the operation of a general GA were presented in subsection 1.3.2. A GA maintains a pool of solutions and - starting from an initial population - the next population's instances are created from the selected elements of the current population. It is done from generation to generation, applying genetic operators until the termination condition is not met. The selection and the genetic operators have to ensure improvement and genetic diversity. In the literature, several different realizations can be found, even for scheduling problems.

Solution representation: encoding/decoding

In some common encoding methods of genetic algorithms, a sequence of numbers is matched with each solution [138].

Definition 57. *Binary encoding: A solution is encoded by a bit string, where each position (a gene of the chromosome) represents a particular characteristic of the problem.*

In binary encoding, one bit usually defines the presence or absence of the property encoded by this position (e.g., in the case of the knapsack problem, a bit expresses if the corresponding object is in the knapsack). As a significant advantage, it is easy to develop genetic operators for binary encoding. In problems where binary numbers can not characterize the problem well, value encoding is a popular choice.

Definition 58. *Value encoding: A modification of binary encoding, where each solution is encoded by a string of some values instead of a binary number. Here, values can be, e.g., integers, real numbers, characters, objects, etc.*

In value encoding, the set of values can be chosen that suits the problem the best. However, this encoding requires the development of specific genetic operators. Ordering problems prefer permutation encoding to binary or value encoding.

Definition 59. *Permutation encoding: A string of numbers encodes each solution, and the order of these numbers determines the sequencing property of the problem.*

For example, in the Travelling Salesman Problem, a number is assigned to each city to be visited - in a bijective way -, and their order in the chromosome determines the sequence of cities on the path of the salesman. Permutation encoding also has its common genetic operators that maintain the feasibility of the instances.

In project scheduling, especially in FJSP, the presented methods of encoding are often mixed. In [139], four different chromosome representations for FJSP are compared. Since the problem can be divided into two subproblems (resource allocation and sequencing), some papers encode the two subproblems separately. In [88, 89, 140], a solution is encoded by two strings: OS string and MS string.

Definition 60. *OS (Operation Sequence) string: A string that is built up by numbers that are indexes of the processes of the problem. The OS string's length is equal to the sum of the number of operations of all processes of the problem. Each process's index appears in the OS string as many times as many operations the process has.*

The k^{th} appearance of a process index refers to the k^{th} operation of this process. By reading left to right, the OS string determines the sequence of the operations to schedule.

Any permutations of an OS string (together with an MS string) represents a feasible schedule. OS string is an example of permutation encoding.

Definition 61. *MS (Machine Selection) string: Determines the assigned machine for each operation of the problem. The length of the MS string is equal to the sum of the number of operations of all processes of the problem. Each position of the string represents an operation. Their sequence follows an increasing order on the process index and the sequence of the operations within the process (e.g., if the first process has k operations, then the first k positions of the MS string represent the first process's operations). The value of each element of an MS string is equal to an index of the vector of the resources that can perform the operation that is represented by the position.*

MS string is an example of value encoding. To decode a pair of an OS string and an MS string, a greedy algorithm is applied. Based on the sequence defined by the OS string, each operation is scheduled to the earliest time on the resource that is defined for it by the MS string in such a way that (1) the start time of the operation is not earlier than the end time of its preceding operation in the process and (2) the time interval occupied by the operation on its machine does not collide with another operation's time interval that was previously reserved on this resource. An example for encoding and decoding based on MS string and OS string is shown in Figure 3.2.

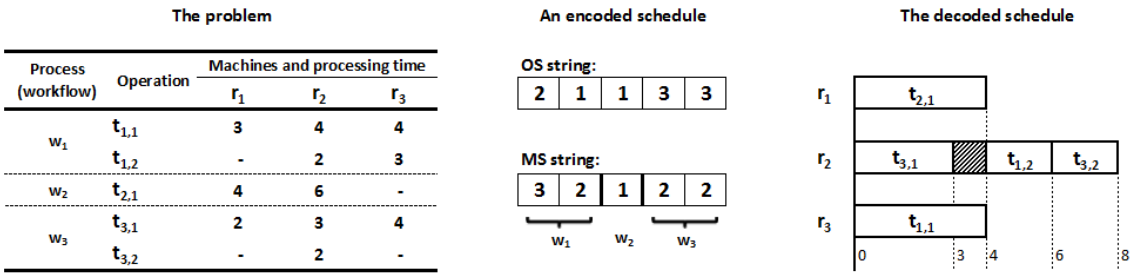


Figure 3.2. Encoding and decoding by OS string and MS string.

It is important to note that both the OS string and the MS string presumes a defined and unambiguous order of the operations for each process.
Kacem et al. [141] propose task sequencing list representation.

Definition 62. *Task sequencing list: A string of triples that encodes a schedule. The length of the string is equal to the sum of the number of operations of all processes of the problem. Each triple represents a scheduled operation, ordered by their start time (operations with the same start time are ordered on their process index). In an (i, j, k) triple, i is the index of the process of the operation, j is the index of the operation within its process, and k is the index of the assigned resource.*

Based on the definition, the task sequencing list applies value encoding mixed with permutation encoding. For example, the task sequencing list of the schedule of Figure 3.2 is illustrated in Figure 3.3.

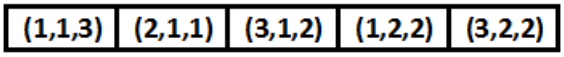


Figure 3.3. Task sequencing list of the schedule of Figure 3.2.

The decoding of a task sequencing list follows the same way as in the case of OS string and MS string (because a task sequencing list expresses both operation sequence and machine allocation). Since the second element of triples is the index of the operation within its process, task sequencing list representation also presumes a predefined ordered sequence of a process's operations.
The most straightforward representation of a schedule is a two-dimensional data structure, where the resources are listed in the first dimension and the second dimension expresses the time. The scheduled operations are placed into this data structure. In the GA of [51], chromosomes are schedules. This is the most direct mapping of a schedule; however, it is difficult to design genetic operators for this representation because of its complexity.

Selection and fitness value

There are several different methods for selecting instances from a generation to recombine them and create new instances of the next generation. Usually, the probability of choosing an instance for crossover depends on its fitness value. Fitness value characterizes the quality of the instance. In the case of FJSP, the fitness value is often determined by the makespan: a better instance has a lower makespan. However, in the case of multi-purpose scheduling problems, additional goals can exist, too, like balancing the load of the resources, the minimization of earliness or tardiness, etc. When the probability of choosing an instance is desired to be proportional with its fitness value, roulette wheel selection is used often.

Definition 63. *Roulette wheel selection: The probability of choosing an instance - that has fitness value f_i - from an n-sized pool is $\frac{f_i}{\sum_{j=1}^{n} f_j}$.*

One variant of roulette wheel selection is linear ranking.

Definition 64. *Linear ranking: The instances of the n-sized population are ordered on their fitness value in an increasing sequence. The natural numbers from 1 to n are assigned to them, respectively, so the i^{th} instance's rank is $r_i = i$. The probability of choosing the instance that has a rank with the value r_i is $\frac{2*r_i}{n*(n+1)}$.*

If the scaling of the instances' fitness values is problematic, tournament selection is also a popular choice for selection.

Definition 65. *Tournament selection: In each tournament, the instance with the best fitness value is chosen from a subset of the instances.*

Zhang et al. [88] applies tournament selection and compares three randomly chosen instances in each tournament, while [89] mixes tournament selection with elitism. In the paper of Pezzella et al. [142], the binary tournament gave better results than the n-size tournament and linear ranking. After obtaining the selected instances, they are recombined.

Crossover

Genetic operators - crossover and mutation - depend on solution encoding. When binary or value encoding is applied, the most common crossover methods are single-point crossover, two- (or k-)point crossover, and uniform crossover [143]. Figure 3.4 gives an example for them.

Definition 66. *Single-point crossover: Within the length of a chromosome, a point is picked randomly. The two children are designed by swapping those genes of the two parents that are to the right of that point.*

Definition 67. *k-point crossover: Within the length of a chromosome, k points are picked randomly. These points cut both parents into $k+1$ segments. The two children are obtained by swapping every second segment of the parents.*

Definition 68. *Uniform crossover: Each gene of the child is chosen from either parent with equal probability.*

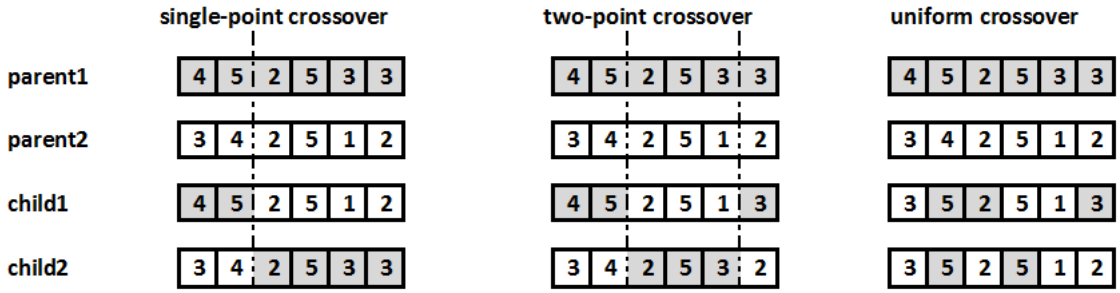


Figure 3.4. Single-point crossover, two-point crossover, and uniform crossover.

Because MS string uses value encoding, the crossover stage of GA in [88] applies half-part two-point crossover and half-part uniform crossover for the recombination of MS strings. In the case of permutation encoding, these methods often would not result in a feasible solution. Several precedence-preserving crossover methods were developed. Two of them are precedence preserving order-based crossover (POX) and job-based crossover (JBX), see Figure 3.5.

Definition 69. *Precedence-preserving order-based crossover (POX): The set of processes are divided into two groups (W_1 and W_2). The genes of the first parent that belong to W_1 are copied into the same position of the first child as it was in the parent. The second parent's genes that belong to W_1 are copied into the same position of the second child as it was in the second parent. Then, the genes of the first parent*

that belong to W_2 are copied into the remaining positions of the second child, and the second parent's genes that belong to W_2 are copied into the remaining positions of the first child, keeping their sequence.

Definition 70. *Job-based crossover (JBX): The set of processes are divided into two groups (W_1 and W_2). The genes of the first parent that belong to W_1 are copied into the same position of the first child as it was in the parent. The second parent's genes that belong to W_2 are copied into the same position of the second child as it was in the second parent. Then, the genes of the first parent that belong to W_1 are copied into the remaining positions of the second child, and the second parent's genes that belong to W_2 are copied into the remaining positions of the first child, keeping their sequence.*

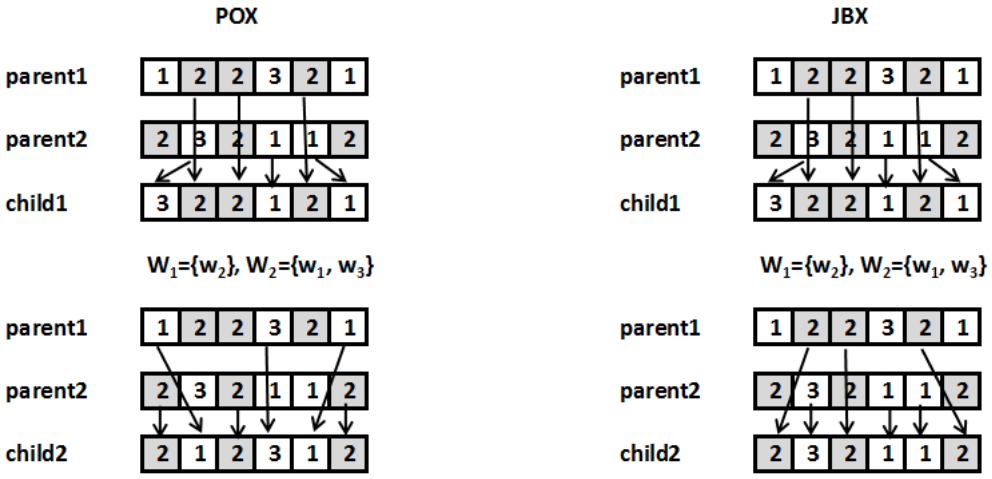


Figure 3.5. POX crossover and JBX crossover.

Both of them are applied to recombine OS strings in [89]. In [142] - where the representation is task sequencing list - also POX is applied, expanded by a simple assignment crossover that is a uniform crossover on parts of the task sequencing list's objects.
When more complex encoding is applied, a special crossover operator has to be defined. For example, Kis - who applied schedules as chromosomes - designs a master/apprentice-based crossover in [51] that focuses on or-subgraphs' critical operations.

Mutation

The mutation genetic operator also depends on the chosen encoding. When binary encoding or value encoding is used, mutation is usually realized by the change of the chromosome's selected genes. In the case of binary encoding, it means the inversion of the selected bit(s), while in value encoding, the new value is chosen from the set of the allowed values. In the latter case, the selection can follow a certain probability distribution, or one bound of the interval of the allowed values can be selected. For

MS string, this mutation is well applicable: [89] changes the value in half of each MS string's positions to the index of a random alternative machine for the corresponding operation. In [88], the new value selection is made in a driven way: instead of a random method, the machine with the shortest processing time is selected. This method is improved in [142], where the gene that represents the machine with the maximum workload is chosen as the gene to change.
For getting a feasible outcome, the mutation of a permutation encoded solution is different. In those cases, the swap of the values of two positions or the move of a value is the most common method. To mutate the OS string, [142] applies the move operation, and [88] uses swap. In the GA of Li and Gao [89], the swap is mixed with neighbourhood mutation for OS strings. Their neighbourhood mutation selects three genes with different values, creates all neighbours of the chromosome by permutating the three values, and chooses one random neighbour as the outcome of the mutation.

Other important features of a GA

The creation of the initial generation

The instances of the initial generation can be generated randomly (e.g., [89] fills the initial population with instances that have both random MS strings and random OS strings); however, Shahryar et al. [144] showed that the initial population influences both the convergence speed of the algorithm and the quality of the final solution significantly. That is why several researchers fill the first generation of their GA in a driven way, also with good quality instances that can be obtained quickly. Zhang et al. [88] start with a generation that is populated by random instances in 10%, by instances obtained by their global selection method in 60%, and the remaining 30% of the population is filled with instances that are the result of their local selection method. In global selection, they allocate the machine with minimum processing time to an operation, while in the case of local selection, they choose the resource with minimum processing time per job. In [141, 142], the processing times and the machines' workload are taken into account in the creation of the initial population. To get different initial assignments in different runs, they also permute the jobs and the machines randomly. The ideas for getting an effective initial population vary on a large scale and highly depend on the problem.

Parameters of the algorithm

Besides the chosen encoding and the design of the genetic operators, the values of different parameters also significantly impact the behaviour of the algorithm. Such parameters are, e.g., the size of the population, the crossover rate, and the mutation rate.

Termination condition

The run of a GA usually stops when (1) the predefined number of generation is reached, or (2) the optimal or a good enough solution is found (it necessitates

the known optimum), or (3) when the improvement decreases under a predefined threshold for generations. These conditions can also be combined.

3.3 My Genetic Algorithm

This section presents the design of the elements of the genetic algorithm that was developed to solve the problem of section 3.1.

The applied encoding

For chromosome representation, a two-dimensional data structure was chosen: the resources are listed in the first dimension, and the second dimension is the ordered sequence of the operations assigned to the certain resource. The reason for this choice is that more compact encodings (like, e.g., OS string/MS string or task sequencing list, see subsection 3.2) require a predefined sequence of all the operations for each process. Here, because of the model processes' and-subgraphs (see Figure 3.1), the sequence of the operations of a process is not predefined, e.g., the operation sequence of the second process model can be either $t^A - t^B - t^C$ or $t^B - t^A - t^C$ or $t^B - t^C - t^A$, regarding the types of the operations.
Figure 3.6 shows an example for the applied chromosome representation, for the sake of simplicity with only two processes ($n(p_1) = n(p_2) = 1$, the process of each operation is marked after the operation type in parentheses, and the two processes are differentiated with the used colors, too).

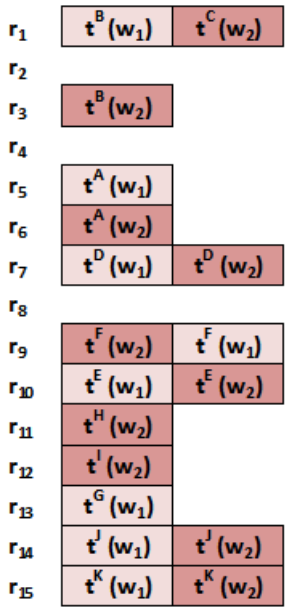


Figure 3.6. An example of the applied encoding.

This representation includes only sequences; it deals with neither exact timing data nor task duration.

Instance creation and the initial population

Each instance of the initial population is created randomly, applying the following steps:

Step 1. If there is at least one unscheduled workflow (w_i) in the set of workflows (W), select one of them randomly. Otherwise, STOP.
Step 2. For each operation of the selected workflow, do:
Step 2.1. Collect all the resources that are capable of carrying out the operation. It results set R^+. (Following the notations of Appendix B and considering operation $t_{i,j}$, $R^+ = \{r_i \in R | capable(RT(r_i), TT(t_{i,j})) = 1\}$).
Step 2.2. Select one random element (resource) from resource set R^+.
Step 2.3. Select a random feasible position in the operation sequence of the selected resource.
Step 2.4. Insert the operation into the selected position.
Step 3. GOTO Step 1.

The concept of "feasible position" used in Step 2.3 has to be explained in more detail. If the feasibility is not checked during the insertion of an operation, there may be cases when the timing of the operations is impossible. It happens when the union of the directed graphs of the workflows' successive operations and the directed graphs of the successive operations of the resources contains a loop. Here, this problem is called "time-loop". The source of the problem can be either the infeasible sequence of the operations of a workflow on one machine (Figure 3.7) or the infeasible sequence of multiple processes' operations on multiple resources (Figure 3.8).

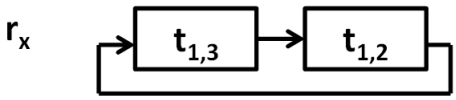


Figure 3.7. Time-loop related to one workflow and one resource.

An operation $t_{i,j}$ of a process must not be inserted after an operation $t_{i,k}$ in the ordered operation sequence of a resource if $t_{i,k}$ is later operation than $t_{i,j}$ in the partially ordered operation sequence of the same process (using the notations of Appendix B, $t_{i,j} \in allPre(t_{i,k})$). Otherwise, from the point of view of the resource, operation $t_{i,k}$ should be performed before operation $t_{i,j}$, and - from the aspect of the operation order of the workflow - operation $t_{i,j}$ should be performed before operation $t_{i,k}$. This infeasible sequence is illustrated in Figure 3.7.
In case of careless operation insertion, a more complex infeasible sequence can be obtained that spreads over multiple resources. Two examples for them are illustrated in Figure 3.8. The timing of the operations of infeasible solutions is impossible. That is why the concept of "feasible position" in Step 2.3 is critical. The selected position's feasibility is checked recursively, taking into account the operation sequence both in processes and on the resources.

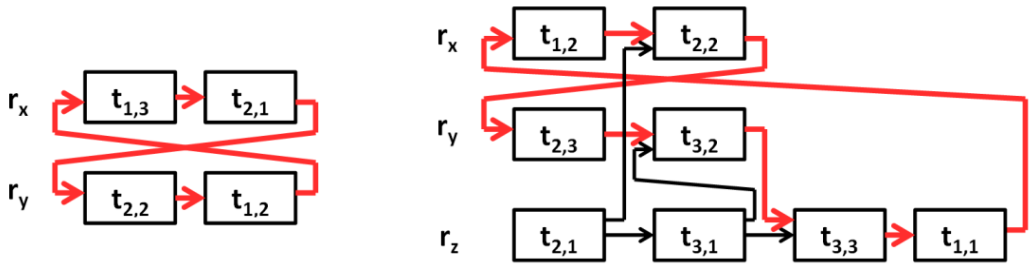


Figure 3.8. Time-loops related to multiple workflows and multiple resources.

The initial population is filled up with random, feasible instances. One parameter of the algorithm predetermines the size of the GA's population. It has to be noted that every genetic operator is constructed so that it produces feasible instance(s). It means that all the instances are feasible, not only in the initial population but also in every further generation.

The fitness value and the timing of the operations

The objective of the developed GA is to find the schedule with the minimum makespan. Selection also necessitates the qualification of the instances. The lower makespan an instance has, the better it is. Based on these requirements, the inverse of the makespan is chosen to be the fitness value.
The determination of the makespan (and thus, the fitness value) of an instance requires the timing of each of its operations. The timing is based on:

- the instance that is an ordered sequence of operations for each resource,
- the workflows' model that orders the operations of the workflows partially,
- the operation time for each allocated operation,
- the duration of the return that a transportation resource needs after it finished an operation to start a new one,
- and the "operation sequence"-dependent setup times of the other type of resources.

Timing applies the following principles:

- Each operation has to be started as early as possible.
- An operation can not be started until its direct prior operation in the operation-queue of its resource is not finished, and the resource has not been transferred to the state in which it is ready to start a new operation.
  (Applying Appendix B's notations and considering the subsequent $t_{i,j}$ and $t_{k,l}$ operations on resource $X(t_{i,j})$, it means that $start(t_{k,l}) \geq start(t_{i,j}) + durRe(TT(t_{i,j}), RT(X(t_{i,j}))) + setup(X(t_{i,j}), TT(t_{i,j}), TT(t_{k,l}))$; of course $X(t_{i,j}) \equiv X(t_{k,l})$.)

- An operation can be started only after all of its prior operations in its process have finished.
  (Formally, based on the notations of Appendix B,
  $start(t_{i,j}) \geq end(t_{i,k}), \quad \forall t_{i,j}, t_{i,k} \in T_i$, where $t_{i,k} \in allPre(t_{i,j})$.)

These principles result in a greedy algorithm for scheduling. After the creation of the schedule, its makespan and fitness value can be determined.
(Following the notations of Appendix B, $makespan(W) = \max_{t_{i,j} \in T} end(t_{i,j})$, and the instance's fitness value is $\frac{1}{makespan(W)}$.)

The derivation of the further generations

Starting from the initial population, the next generations' instances are derived by three different methods/genetic operators: elitism, crossover, and mutation. As usual (when a GA is developed), two objectives were kept in mind when the applied genetic operators were designed: (1) to find the schedule with the maximum fitness value (so, the good instances should be kept and the worse instances should be rejected) and (2) to explore the search space to prevent getting stuck in a local optimum.

Elitism

My GA contains two elitism-related parameters:

- the percentage of the instances of a generation that are obtained by elitism (this value is constant, regarding the generations of the GA),
- and a logical value that shows whether an elite instance can be reselected when the remaining part of the new generation is created (actually, this parameter determines whether the elites are copied or moved from a generation to the next generation before all other genetic operations are applied on the instances of the earlier generation to create the new instances of the new generation).

Crossover and selection

The solution encoding used in the developed GA does not make the use of the common, simple crossover methods possible. A special crossover operator was developed that can be controlled through several variables. After applying elitism, the new generation's missing part is populated by random instances of the previous generation. One parameter - described in the latest subsection - controls whether elites can be reselected in this phase. Then, another parameter is used that shows the probability of applying crossover between the instances of a (half-ready) generation. Based on this parameter's value, a binary decision is made for each generation, separately, one-by-one. If the crossover is applied in a generation, another parameter shows the percentage of the new generation's instances created by crossover.
The basic variant of the GA randomly selects the instances that are intended to be recombined. This random selection is changed later; see subsection 3.6. Each instance that is created by crossover is the result of the following steps:

Step 1. Select two different instances randomly from the new (half-ready) generation (*parent*1 and *parent*2).

Step 2. A parameter of the algorithm determines what percentage of the workflows are inherited from *parent*1 (the remaining part origins from *parent*2). Based on the value of this parameter:

Step 2.1. Select randomly as many processes from *parent*1 as the value of the parameter indicates.

Step 2.2. Create a new instance where the resource assignment for each operation of the selected workflows is the same as in *parent*1. The sequence of the operations is random, on the condition that it results in a feasible instance (using the same feasibility check during the operation insertion as that of the initial population creation step).

Step 2.3. The resource assignment of the operations of the remaining workflows is the same as in *parent*2. The sequence of the operations is also random, taking care of obtaining a feasible instance.

Finally, randomly chosen instances of the (half-ready) new generation are replaced with the resulted children. Because crossover is applied to the instances of the new generation, a parameter was introduced that determines whether the instances selected by elitism can be replaced with the results of crossover or not (originally, GAs operate like this GA with false value for this parameter).

Mutation

The developed GA also has a global parameter for mutation, which gives the probability of applying mutation for a generation. Based on its value, a binary decision is made for each generation, separately, one-by-one. It means that there may exist generations whose instances are not influenced by mutation genetic operation at all. Another parameter shows the probability of mutation per instance. It has to be taken into account if the mutation is applied to a certain generation. In this GA, mutation means the execution of the following steps on the instance that is intended to be mutated:

Step 1. Select one process from the model of the problem randomly.

Step 2. Delete every operation of the selected process from the instance.

Step 3. Do the following steps for each operation of the selected process:

Step 3.1. Collect the resources that can carry out the operation.

Step 3.2. Select randomly one from these resources.

Step 3.3. Insert the operation into a random, feasible position of the operation sequence of the selected resource.

The GA has a parameter that shows whether an elite can be mutated or not (usually, GAs operate like this algorithm with true value for this parameter).

The next generation is finalized after applying these steps (elitism, crossover, and mutation). The size of the population remains unchained during the whole execution of the algorithm.

Termination condition

My GA stops when it reaches a predefined number of generations.

3.4 Parameter values of the Genetic Scheduler and computational results

This section presents the computational results for the previously introduced problem obtained by the developed genetic scheduler. For finding a good solution efficiently, the appropriate setting of the parameters of the algorithm is necessary. The search for the effective values of the genetic operator related parameters was started from the values suggested by Pongcharoen et al. [2]. Because the problem of this thesis differs from their problem, their proposed values have to be adjusted to find the appropriate ones. After finding the parameter values for a fixed input problem, the genetic scheduler's behaviour is investigated on a random input set, where both the number of products and the operation times are chosen randomly from a predetermined interval.
In both cases (deterministic and random input), the following algorithm-specific parameter settings are applied:

- The number of the generations: 400
- The percentage of elites in a population: 20%
- The probability of applying crossover on a generation: 100%
- The percentage of the processes that originate from the first parent in the crossover: 50%
- The probability of applying mutation on a generation: 100%
- Crossover and mutation cannot rewrite the instances that were resulted from elitism.

3.4.1 Genetic operator related parameter values for the deterministic problem

This subsection presents the way to find the appropriate values of the population size, the crossover rate, and the mutation rate of the genetic algorithm when the problem's parameters are fixed. The product numbers are: $n(p_1) = 15$, $n(p_2) = 20$.

Table 3.1. Fixed and random operation times of the problem.

Resource(s)	Operation type	Operation time		Duration of the comeback of the transportation resource	
		Fixed	Random	Fixed	Random
r_1, r_2, r_3	t^B	6	[4,8]	-	-
r_1, r_2, r_3	t^C	10	[8,12]	-	-
r_4	t^B	10	[8,12]	-	-
r_4	t^C	6	[4,8]	-	-
r_5, r_6	t^A	6	[4,8]	-	-
r_7, r_8	t^D	4	[3,5]	-	-
r_9	t^F	3	[2,4]	3	2*(the selected op.time)
r_{10}	t^E	5	[3,7]	5	2*(the selected op.time)
r_{11}	t^H	3	[2,4]	3	2*(the selected op.time)
r_{12}	t^I	5	[3,7]	5	2*(the selected op.time)
r_{13}	t^G	4	[3,5]	4	2*(the selected op.time)
r_{14}	t^J	3	[2,4]	3	2*(the selected op.time)
r_{15}	t^K	4	[3,5]	4	2*(the selected op.time)

The setup time on r_1, r_2 and r_3 machines is 5 time units, and on machine r_4, it is 6 time units (both switching from a t^B-type operation to a t^C-type operation and vice versa). The fixed operation time of all machine-operation pairs and the fixed duration that a transportation resource needs to start a new operation after finishing a previous task are indicated in Table 3.1. Appendix K presents these input parameter settings regarding the notations of Appendix B.

First, the algorithm's parameters were set to the values suggested by [2] for their problem. These are:

- The percentage of the instances of a generation that origin from crossover: 70%
- The probability of applying mutation on an instance: 18%
- The size of the population: 60.

Figure 3.9 shows the average of the minimum, the average, and the maximum makespans of 50-sized generation groups for one run of the GA with these settings (10 different runs of the genetic algorithm are illustrated - with one generation resolution - in Appendix L.1). It can be seen that the big generation number is reasonable since there are improvements even in the latest generations. The computation takes a quite long time: 422-436 minutes for the 400 generations on a Fujitsu Lifebook with a 1.7 GHz Intel Core i5 CPU and 8 GB RAM.

While analyzing the impact of the values of population size, crossover rate, and mutation rate on the GA's efficiency, the number of the generations was fixed to 400, and the 3 examined parameters were changed independently from each other. First, the size of the population (ps) was varied between 10 and 150, and the algorithm was executed 5 times for each value. The results are shown in Table 3.2. The

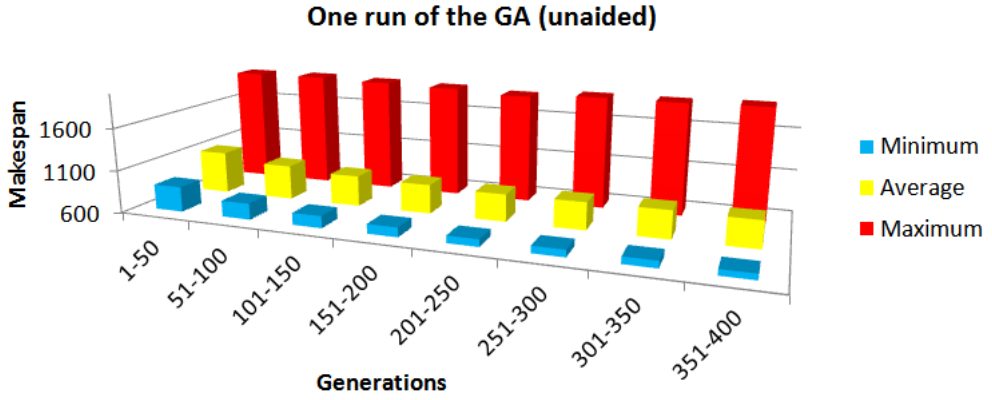


Figure 3.9. Computational results of the unaided genetic algorithm - with the genetic operator related parameter values suggested by [2] - for the deterministic problem.

Table 3.2. The best makespans and the average/standard deviation of the makespans of 5 runs of the GA - with different population sizes - on the deterministic problem /values are in time units/.

generation: 400, crossover rate: 0.7, mutation rate: 0.18, population size: ps									
	ps=10	ps=20	ps=30	ps=40	ps=50	ps=60	ps=80	ps=100	ps=150
The best	829	796	779	792	767	740	743	732	732
Avg.	850	810.2	811.6	807.6	781.2	778.4	771.4	763.8	744.4
Std.dev.	16.96	9.97	20.26	18.70	12.09	24.69	21.30	19.96	10.61

best makespan was obtained when the population size is 100 or larger; moreover, the average makespan decreases with the increase of the population size (however, the standard deviation is quite large). This result was predictable since a larger population size means more opportunity to generate/produce an instance with high fitness. For the further experiments, $ps = 100$ was applied as a balance between the good enough result and the necessary computation time.

The second examined parameter is the percentage of the instances of a generation that is resulted by crossover (*cross*). Originally, its value was set to 0.7, but the genetic scheduler's behaviour was analyzed for other values, too. Table 3.3 shows the results. The best makespan and the smallest average makespan were obtained when $cross = 0.3$ (however, the standard deviation was the highest in this case). The results show that the percentage of the instances of a generation that is resulted by crossover - in contrast to the experiences of Pongcharoen et al. [2] - influences the quality of the result. Its reason can be the difference in the design of the genetic operators. The results show (taking into account the small value of mutation rate and elitism usage, too) that a small modification of the chromosomes is advantageous. However, too much alteration is harmful (because the algorithm progresses toward the population of higher quality chromosomes on average).

Finally, the impact of the change of the mutation rate (*mut*) was investigated. Till now, this parameter was set to 0.18, following the results of [2]. The results for various mutation rate values (with unchanged values of the other parameters) are

Table 3.3. The best makespans and the average/standard deviation of the makespans of 5 runs of the GA - with different percentages of the chromosomes of a generation that are resulted by crossover - on the deterministic problem /values are in time units/.

generation: 400, population size: 100, mutation rate: 0.18, crossover rate: cross				
	cross=0.7	cross=0.5	cross=0.3	cross=0.1
The best	732	689	660	684
Avg.	763.8	706.8	687.6	703.6
Std.dev.	19.96	9.62	20.24	11.89

Table 3.4. The best makespans and the average/standard deviation of the makespans of 5 runs of the GA - with different probabilities of applying mutation on a chromosome - on the deterministic problem /values are in time units/.

generation: 400, population size: 100, crossover rate: 0.3, mutation rate: mut					
	mut=0.08	mut=0.18	mut=0.28	mut=0.38	mut=0.48
The best	698	660	674	648	648
Avg.	750.2	687.6	694.8	667	674.2
Std.dev.	70.83	20.24	21.75	11.17	14.59

shown in Table 3.4. Some increase in mutation rate results in better performance, but above a certain value, the average makespan increases. Mutation rate value 0.38 performed the best (out of the examined ones): it gave the smallest makespan, the smallest average makespan, and the smallest standard deviation. Presumably, this quite high mutation rate makes the algorithm often jump out of the local minimum, but the GA's construction does not let to leave a better part of the search space for a worse part.
After determining the most promising values for the three parameters, the search for the best values could have been started again for the population size, the crossover rate, and the mutation rate from this point of the 3D search space. However, the experiences revealed that the new iterations do not signify.

3.4.2 Investigation of the performance of the genetic algorithm on random inputs

After determining efficient values of the genetic operator related parameters for the deterministic problem, the behaviour of the GA with this parameter setting (population size: 100, crossover rate: 0.3, mutation rate: 0.38) was investigated on random inputs. Both the number of the workflows (desired products) and operation times were chosen from predefined intervals randomly, following uniform distribution on integers. The desired product numbers are: $n(p_1) \in [10, 20]$ and $n(p_2) \in [15, 25]$. It means that there are 25-45 workflows per problem that have to be scheduled. The intervals of that the operation times are chosen randomly, and the duration that a transportation resource needs to start a new operation after finishing a previous

Table 3.5. The random parameter values of 6 inputs and the obtained makespans by the GA.

		*Run*1	*Run*2	*Run*3	*Run*4	*Run*5	*Run*6
$n(p_1)$ (pieces)		17	16	13	14	13	18
$n(p_1)$ (pieces)		17	23	21	22	15	21
Resource(s)	Operation type	Operation time					
		*Run*1	*Run*2	*Run*3	*Run*4	*Run*5	*Run*6
r_1, r_2, r_3	t^B	8	6	6	8	5	6
r_1, r_2, r_3	t^C	11	10	10	12	9	11
r_4	t^C	5	4	4	5	4	4
r_4	t^B	10	11	12	10	11	9
r_5, r_6	t^A	6	7	6	7	5	6
r_7, r_8	t^D	5	5	5	4	5	3
r_9	t^F	2	4	4	2	2	2
r_{10}	t^E	4	5	4	3	4	3
r_{11}	t^H	2	2	4	4	4	2
r_{12}	t^I	4	7	4	7	6	6
r_{13}	t^G	3	3	5	3	4	4
r_{14}	t^J	2	2	2	4	4	3
r_{15}	t^K	5	5	3	5	4	4
obtained makespan (time units)		567	700	724	658	529	630

task are shown in Table 3.1. All the other properties of the problem remain the same as in the deterministic case. Several random inputs were generated and solved by the GA. 6 runs of them are shown in Table 3.5, where both the chosen random parameter values and the resulted makespans can be seen for each run. An example of the convergence of the algorithm is given in Figure 3.10. This figure shows the best makespans for each generation of *Run*2.

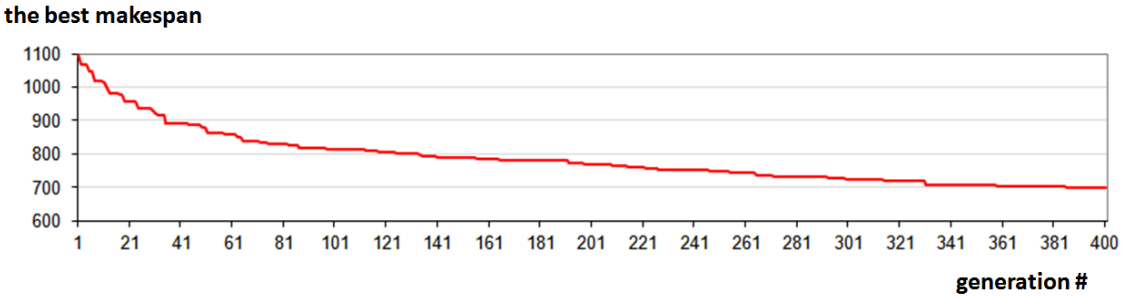


Figure 3.10. The convergence of the developed GA on the random input of *Run*2.

The following statements can be concluded from the results:

- the obtained makespan of the random cases that have a similar size to the deterministic problem was similar to that of the deterministic case (this result was enforcement for the consistency of the algorithm),

- the chosen number of generations (400) is reasonable, especially in cases where the best makespan of the initial population is quite high, and
- the input parameter values significantly influence the resulted makespan.

Further works may discover the impact of the input parameters - e.g., the operation times and the cardinalities of the different product types - on the GA's performance through sensitivity analysis.

3.5 Comparison of the GA with other solution approaches regarding the results

Genetic Algorithm is not the best candidate to solve such a problem, where the calculation of the fitness of one instance requires complex computations, since GA necessitates to do this computation several times (for every instance of the population of all generations). There are also other methods to find a good enough solution in the search space. For example, the simulated annealing-based algorithms modify only one instance in each step of the algorithm. There are also ant colony-based methods, tabu search, hill-climbing, etc., to discover the search space (see Figure 1.4).
To determine the quality of the results obtained by the developed GA, three other approaches were applied to solve the scheduling problem: calculation of a coarse lower bound, simulated annealing (SA), and a fusion where the result of simulated annealing was placed into the first generation of the genetic algorithm. The results of the different methods are compared on their quality.

3.5.1 Coarse lower bound

During the analysis of the scheduling problem, it can be realized that the transportation resources cause a significant bottleneck: they can transport only one raw material/product at a time, they have to get back to their initial location/position before they start a new operation, and the operation time they require is comparable with the operation time of the operations of type t^A, t^B, t^C or t^D (that can be performed by more than one resources).
Based on this observation, a coarse lower bound of the makespan can be calculated. In the case of the deterministic problem (see Table 3.1), the coarse lower bound is calculated as follows:
Considering the longest branch of the second workflow model (see Figure 3.1), all of its transportation operations have to be performed $n(p_2) = 20$ times, and from these 20 executions, the transportation devices have to return to their initial position 19 times.
These actions require $(3 + 3 + 5 + 4) * 20 + (3 + 3 + 5 + 4) * 19 = 585$ time units.
This result has to be increased with the minimal operation time of at least one t^B-type, one t^C-type, and one t^D-type operation. This way, the coarse lower bound is:
$585 + 6 + 6 + 4 =$ **601** time units.

This calculation assumes that there is no delay between the operations of the workflow. In real it is surely not the case, so the obtained lower bound is coarse. However, the calculated lower bound fits well with the deterministic problem's obtained results (see the 10 figures in Appendix L.1).

3.5.2 Simulated annealing

For comparison purposes, a simulated annealing algorithm was also implemented to solve this chapter's deterministic scheduling problem. This algorithm modifies its instance by the mutation genetic operator of the previously presented genetic algorithm (see subsection 3.3). It means that all the operations of one randomly chosen process are reallocated.
The algorithm was executed in two different ways: first, the start temperature was set to 1000, and the cooling rate was 0.03; second, the start temperature was set to 1000, and the cooling rate was 0.000172679. The reason for choosing 0.000172679 as the cooling rate is to get an SA algorithm comparable with the GA, regarding their size: the genetic algorithm had 400 generations and 100 instances in each generation. It means 40000 instances altogether. The same size - 40000 iterations - can be reached by SA if the cooling rate is set to 0.000172679 (assuming that the start temperature is 1000).
The first case (the start temperature is 1000 and the cooling rate is 0.03) resulted in 228 iterations, and the runtime of the algorithm was only about two minutes. The results of one run with these settings are illustrated in Figure 3.11; 10 different runs can be found in Appendix L.2.

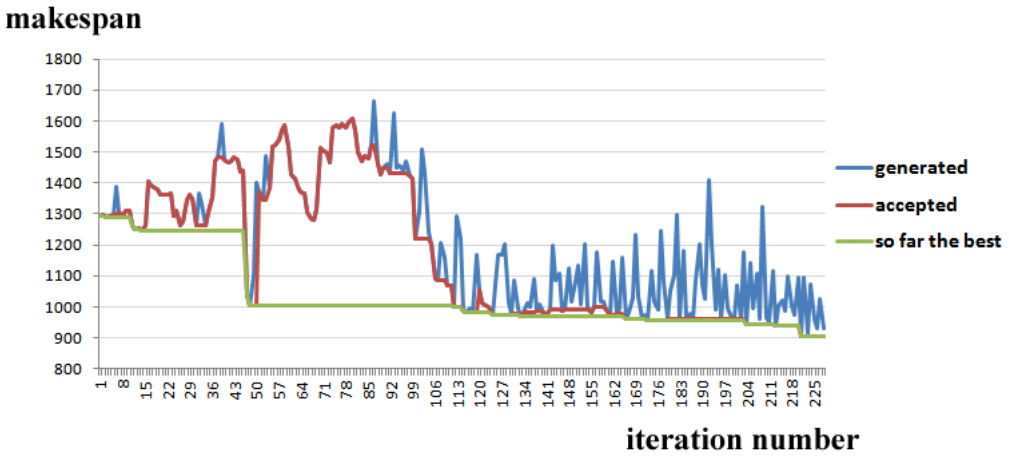


Figure 3.11. The results, obtained by SA through 228 iterations for the deterministic scheduling problem.

The blue curve shows the makespan of the instance of the current iteration. The red curve represents the accepted makespan in the iteration (it is either the makespan of the current instance or the makespan of the previous iteration's instance). Finally, the green color shows the best makespan that is obtained until the current iteration. The first way of execution resulted in a rapid run (2 minutes instead of 7 hours);

however, the solution's quality was much worse than the result of the GA (800-900 instead of 600-700). Its main reason is the small number of iterations (228).
The results of one execution of the SA through 40000 iterations are illustrated in Figure 3.13. As expected, 40000 iterations result in a similar quality solution - and within similar runtime - as GA does with 400 generations and 100 instances in each generation.

3.5.3 Simulated annealing-aided genetic algorithm

It was also examined how the genetic algorithm performs if the solution resulted from the simulated annealing is inserted into the first generation of the GA. As a time/quality trade-off, the first version (228 iterations, about two minutes total runtime) of SA was applied. The results - as the function of the generation number - are illustrated in Appendix L.3 for 10 runs. Figure 3.12 shows one run of them; the average of the minimum, the average, and the maximum makespans of 50-sized generation groups are presented.

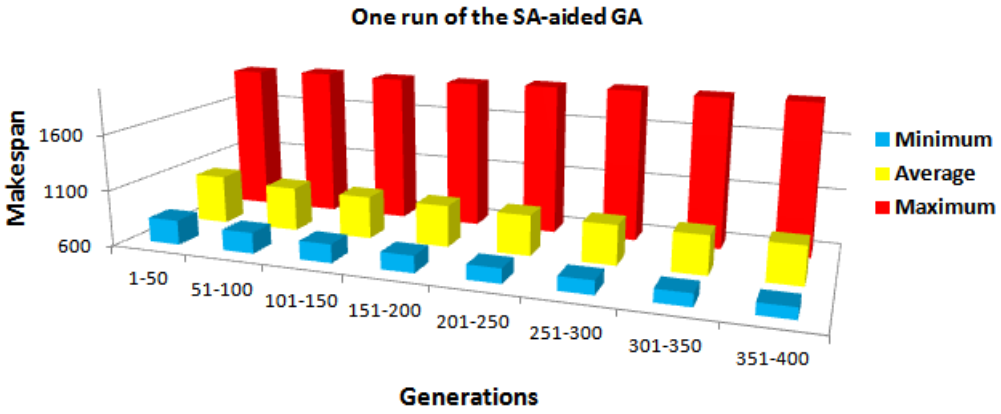


Figure 3.12. Computational results of the SA-aided genetic algorithm for the deterministic problem.

It can be realized that the genetic algorithm initialized by the result of simulated annealing approaches the optimum quicker in the first generations; however, in the later generations, no significant difference was experienced related to the unaided GA.

3.6 The analysis of the GA's efficiency on FJSP benchmark problems and the improvement of the algorithm

Although the genetic algorithm was developed to solve the presented scheduling problem (see subsection 3.1) that is even more than a "simple" FJSP problem, the design and the implementation of the GA makes it possible to be applied on a wider class of problems. This problem class origins from the presented problem; however,

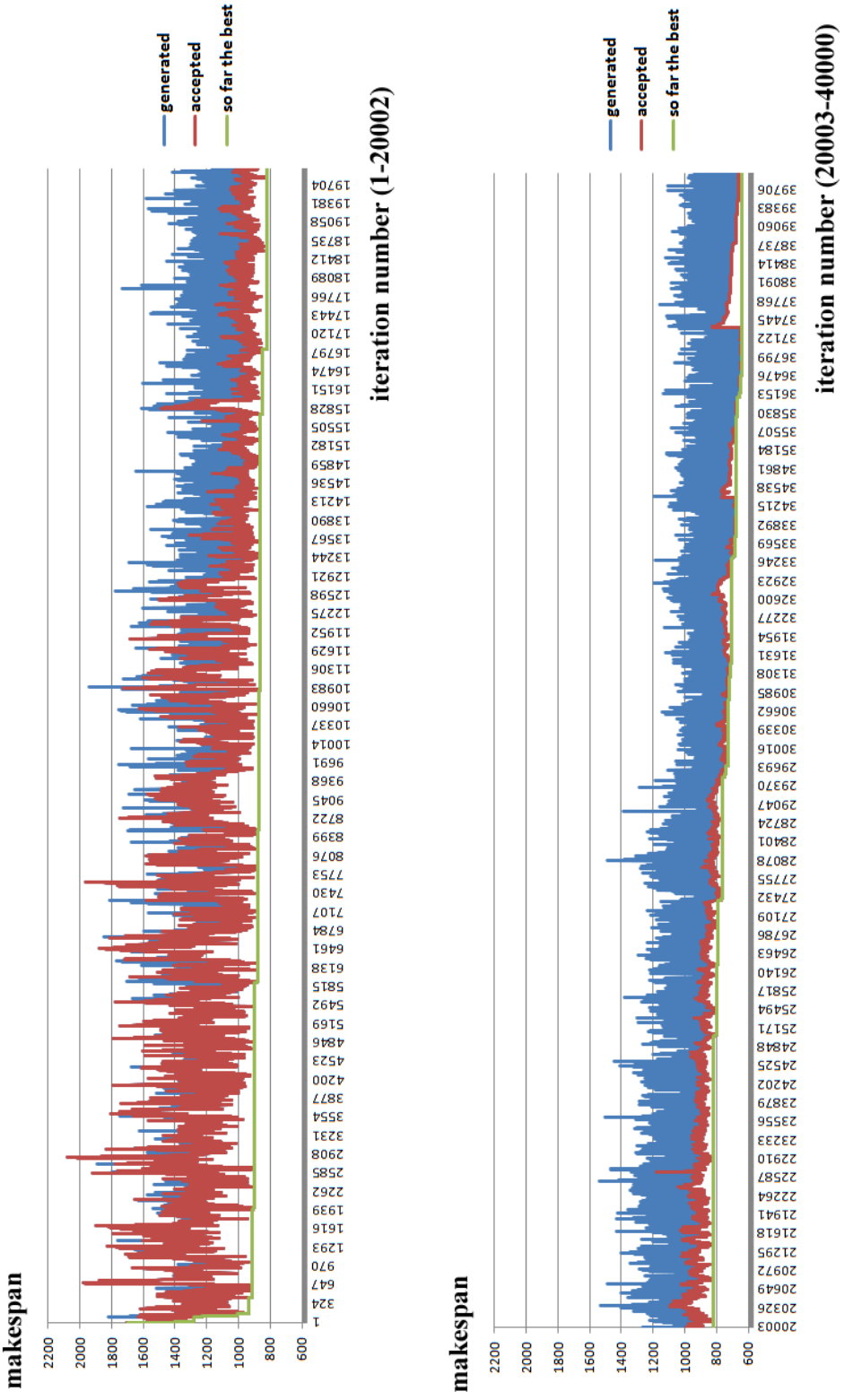


Figure 3.13. The results, obtained by SA through 40000 iterations for the deterministic scheduling problem.

(1) the number of the resource types and the resources can be arbitrary, (2) the number of the production process types and the number of the processes can be arbitrary, and (3) the structure of a process can be an arbitrary directed acyclic graph (thus, with an arbitrary number of and-subgraphs). This design made the GA possible to solve the benchmark FJSP problems that can be found in the scientific literature. The efficiency of the algorithm was analyzed on some of them.
The benchmark FJSP problems for the analysis were chosen from Brandimarte [3], Hurink et al. [1], and Barnes and Chambers [4]. In these problems, the number of jobs is between 7 and 20 (mostly 10, 15, or 20), the number of operations per job is between 3 and 15, and the number of machines varies from 4 to 15. The average number of the machines that can perform an operation is 1 in case of the benchmark problems of Barnes and Chambers, 1.15 in case of the edata, and 2 in case of the rdata test problems of Hurink et al., and varies between 2 and 3.5 in the test problems of Brandimarte. All of the benchmark FJSP problems' jobs have a simple sequential structure: they contain neither and-subgraphs nor or-subgraphs.
It has to be highlighted that the GA was developed to solve more general problems. The differences were summarized in subsection 2.2, where the basic scheduling problem was introduced, which is extended in this chapter. The considered problem contains setup time, applies multi-purpose machines, deals with transportation operations, requires the execution of the same process model several times (because of mass production), but there are also different process models; moreover, it includes and-subgraphs where the operations of the parallel branches can be executed in parallel, in contrary to the benchmark FJSP problems. That is why it can not be expected that the developed GA outperforms the algorithms that were developed especially to solve FJSP problems.

Table 3.6. The makespans obtained on the $mk01$ problem by the GA.

Results of the GA*			
optimum	minimum	maximum	average
40	48	60	55.7

*from 10 runs

First, the GA's efficiency was tested on the benchmark problem $mk01$ of Brandimarte [3]; the obtained results are presented in Table 3.6. The experiences showed that the decrease of the population size to 50 does not influence the solution's quality significantly; however, the increase of the generation number is useful for this problem. 1200 was chosen as the number of generations. Minimum 136 generations were needed to obtain the best solution of one run, and the maximum number of generations - when the best result of execution was reached - was 1145 during the 10 runs. The average number of the generations when the best solution of the run is reached was 604.1. These numbers give a reason for choosing 1200 as the number of generations. The GA was also tested on the $mk02$ test input of Brandimarte [3]. Its results are presented in Table 3.7.

Since the results obtained by the GA were far from the optimum, the GA was improved. The steps of the improvement attempt are detailed in Appendix M and

Table 3.7. The makespans obtained on the $mk02$ problem by the GA.

	Results of the GA*		
optimum	minimum	maximum	average
26	47	73	55.75

*from 8 runs

Table 3.8. Summary of the attempts for the improvement of the GA.

Step	Description	Accepted
1.	Creation of the initial population by a heuristic that works from job to job	yes
2.	Modification of the heuristic that creates the initial population to schedule from operation to operation in a greedy way	yes
3.	Modification of the crossover to apply scheduling from operation to operation instead of from job to job	yes
4.	Modification of the fitness function and introduction of controlled selection	yes
5.	Modification of the mutation by inserting each rescheduled operation into the earliest feasible position	no
6.	Application of heuristic also for crossover	no
7.	Complete review of the mutation genetic operator	yes

are summarized in Table 3.8. Most of them improved the solution; however, some of them proved to be inefficient. The useful improvements are built into the final GA (this decision is highlighted in the last column of Table 3.8).

3.6.1 The final GA and its test on some FJSP benchmarks

The GA found the best results when (1) the initial population was filled by good-quality instances that were generated by the developed heuristic that schedules from operation to operation, (2) the scheduling in the crossover also happens from operation to operation, (3) the fitness calculation involves the loads and the completion times of the machines (next to the makespan of the schedule), (4) selects both parents to recombine from the elites, and (5) applies the new mutation - that reschedules all the operations that are subsequent to a randomly chosen operation of a resource whose completion time is equal to the makespan in the directed graph of the schedule (the randomly chosen operation is also rescheduled, changing the allocated machine only of this operation) - for the best 20 instances of the population (mutating each of them 20 times and accepting the best result as the mutated instance). In each execution, the population's size was 50, and the initial population was filled by the best 50 of the 1000 instances generated by the heuristic.
This improved GA was executed on several test instances of the scientific literature, the results are summarized in Appendix O. As it was stated, it was not expected that the developed GA outperforms the algorithms (see, e.g., [51, 88, 89] or [142]) that were developed especially to solve FJSP problems and found the optimum in

Table 3.9. Comparison of the results obtained by the initial and the final variants of the GA on the $mk01$ and the $mk02$ problems.

problem	optimum	result obtained by the initial GA	result obtained by the final GA
mk01	40	48	41
mk02	26	47	30

most of the cases. Although, the developed GA approximated the optimum well in the case of several instances. The GA's improvement is demonstrated on the $mk01$ and the $mk02$ test inputs of Brandimarte [3] in Table 3.9.

3.7 The contribution of this chapter

The contribution of this chapter can be summarized in three points: (1) it proposes a genetic algorithm for solving a scheduling problem, (2) it sets the parameters of the GA for the case of a concrete problem, and (3) it determines the efficiency of the algorithm on some well-known FJSP test problems.
It was experienced that the developed GA only approximates the optimum of the benchmark problems (contrary to the algorithms that were created to solve FJSP problems); however, it can handle a much wider problem class than the classic FJSP problems.

3.7.1 The problem class that can be solved by the developed GA

The problem class that can be handled by the presented GA is much wider than that of the heuristic algorithm of Chapter 2 since this algorithm works without the simplifications of the previous chapter. This GA can approximate the solution of any scheduling problem that has the following properties:

- there is a predefined arbitrary set of operations,
- there is a predefined arbitrary set of resources; each resource is able to perform operations of an arbitrary subset of operation types,
- it is allowed that some resources are transportation devices with a capacity of 1, that require time to return to their initial position after finishing an operation, to start a new operation (like, e.g., robotic arms),
- there is a predefined arbitrary set of processes, each of them can be described by an acyclic directed graph (allowing forks and joins) where each node is an operation,
- any process should be performed a certain number of times (mass production is assumed),
- the operation time depends on both the operation and the applied resource,

- operations are uninterruptable,
- for any operation, there is a non-empty subset of the resources that contains machines that can perform the operation,
- for each operation, exactly one resource is assigned,
- setup time can be applied,
- operation times, setup times, transportation, and the number of the required products (the number of the execution for any process) can be arbitrary,
- the goal is the minimization of the makespan.

This problem class covers any scheduling problem that can be described by the formal notations of Appendix B.

Chapter 4

Fault-tolerant extension of Vehicle Routing Problem solutions

The final destinations of the products of the production processes are the users. The distribution of the products also has to be cost-efficient. It means - besides the short planned routes - efficient handling of vehicular faults. This chapter proposes a fault-tolerant extension for route executions.

4.1 Background of the research

Depending on the way of describing faults, a special stochastic or deterministic extension of a given VRP solution method can be developed. This way, the costs related to fault occurrences can be decreased or minimized. When the implementation of an optimal plan determined earlier leaves some room for optimal execution, this freedom can be used to adjust the plan if the need arises. The aim of such a post-planning change can be to optimally adapt the execution of the trips in question to the changing circumstances. When the cause of the changes the breakdown of some of the vehicles is, a goal can be to still deliver all of the goods to the customers with a minimum extra cost. Preparing a plan for the implementation of this minimal goal can be regarded as a *fault-tolerant extension of the VRP solution*. In order to minimize the extra cost that results from providing aid to another vehicle in case of a breakdown, an algorithm is intended to be created that determines the starting direction for each vehicle on its route without changing the routes. So, the goal is to select the best possible route orientation considering the case of an unexpected event. Another requirement is that the efficiency of the solution should not decrease without a fault. This makes the execution of the original routes fault-tolerant since, in case of a fault, all the customers are serviced, and the cost of the correction action is minimized on average. The probability of a parallel breakdown of more than one vehicle is extremely low. Moreover, the computation is based on presumed concrete vehicle failures, which results in case-specific recourse action. The improvement obtained in comparison to action taken without using the presented method is also case-specific. It justifies the use of this deterministic procedure. Unlike papers that present recourse action for unforeseen events described by a probability model, here, the average extra cost of vehicle failure is calculated using well-determined break-

down locations. Probability models are presented for power outages in power grids by Al-Kanj et al. in [145].

If a breakdown of a member of a fleet occurs during distribution, the company's goal is to (1) solve the problem with minimal additional cost and (2) to cause an as small delay in customer service as possible. The presented approach intends to reach this goal by inserting the assistance action into an ideal place on the helper vehicle's route. The vehicles are allowed to give aid even before finishing their own routes. If the capacity constraint is not satisfied in the case of giving aid, the vehicle in question is allowed to visit the broken-down vehicle. My algorithm finds the directed route set in which aid can be offered with minimal additional cost by only one vehicle. In my thesis, I propose a fault-tolerant execution for the solution of a Capacitated Single Depot VRP without Time Windows with symmetrical distances between points and making such execution optimal by changing only the direction of vehicle routes. The method determines the direction of the circular routes, which start and end at the depot so that the shortest extra route will be part of the solution in cases when a vehicle needs to take over another one's tasks. The content of this chapter was published in [125].

4.2 The formal description of the Vehicle Routing Problem

The version of the Vehicle Routing Problems considered in this thesis is the Capacitated Single Depot VRP without Time Windows. In this problem, vehicles start from a common depot, visit and service customers and return to the depot. During its trip, no vehicle may exceed its own capacity by the sum of visited customers' demands. Also, the overall cost of the routes should be minimal.

In order to state the VRP formally, the following notations are introduced:

Notations 4.2.1. *Vehicle-, customer-, and route-related notations:*

m: *the number of customers.*

$M = \{m_1, \ldots, m_m\}$: *the set of customers.*

n: *the number of vehicles.*

$N = \{n_1, \ldots, n_n\}$: *the set of vehicles.*

x_i: *the capacity of vehicle* n_i $(1 \leq i \leq n)$.

d_j: *the amount of the demand of customer* m_j $(1 \leq j \leq m)$.

r: *the number of routes. If the problem contains only one depot (Single Depot VRP) and all the vehicles have exactly one route, then* $r = n$.

$R = \{R_1, \ldots, R_r\}$: *the set of routes.*

R_i: *the route of vehicle* n_i $(1 \leq i \leq n)$ *that is a vector of customers.*

$R_i = [d, m_{i,1}, m_{i,2}, \ldots, m_{i,k_i}, d]$, *where* d *is the depot (the initial and final location of any vehicle),* $m_{i,j}$ *is the* j^{th} *customer of vehicle* n_i*, and the number of customers who are serviced by vehicle* n_i *is* k_i*.*

E_i*: the set of the elements of* R_i*.* $E_i = \{d, m_{i,1}, m_{i,2}, \ldots, m_{i,k_i}\}$ *if* $R_i = [d, m_{i,1}, m_{i,2}, \ldots, m_{i,k_i}, d]$*.*

$C(R_i)$*: the cost of vehicle* n_i*'s route* $(1 \leq i \leq n)$*.*

Based on these parameters, the problem can be formally stated as follows. The overall cost is minimized

$$min \sum_{i=1}^{n} C(R_i) \tag{4.1}$$

subject to the following constraints:

$$E_i \cap E_j = \{d\}, \quad \forall\, 1 \leq i, j \leq n, i \neq j \tag{4.2}$$

$$\bigcup_{i=1}^{n} E_i = M \cup \{d\} \tag{4.3}$$

$$\sum_{j=1}^{|R_i|-2} d_j \leq x_i, \quad \forall\, 1 \leq i \leq n, m_j \in R_i. \tag{4.4}$$

The first constraint (4.2) ensures that each customer is visited by only one vehicle. Constraint (4.3) states that all the customers are visited. Finally, constraint (4.4) guarantees that no vehicle exceeds its capacity so that it goes above the sum of visited customers' demands. Other variants of VRP may expand or modify the above constraints (4.2)-(4.4).

4.3 Common solution methods for Vehicle Routing Problem

Finding the optimal solution of a VRP is a highly resource-intensive process since it is already NP-hard for one vehicle (the TSP). In practice, companies do not have the time to use exact algorithms for finding the optimal solution. Instead, logistic experts fall back on a near-optimal solution that can be computed in an acceptable time.

Usually, an initial solution of the VRP is created first and then improved in a later step. This first step, the initialization, can be a random solution generation or a heuristic-driven action.

In the case of a general VRP, the initialization phase can be divided into three sub-tasks:

- First, the elements of the set of customers (M) have to be assigned to depots. This phase is called clustering. Although this thesis deals with problems

with only one depot, the proposed method can be generalized for more general cases with more depots where these clustering methods are taken into account. Clustering methods that are more sophisticated than the random method (which selects one depot for a customer based on a random number between 1 and the number of the depots) are reviewed and compared in [146].

- Second, the customers belonging to the same depot have to be grouped into routes. This process is called routing. A popular, thoroughly documented, and easy to implement algorithm for determining the routes that belong to a depot is the Clarke and Wright (C&W) savings method [104] (see Appendix P). This algorithm can be met often in the literature, partly because of the underlying idea on which it is based and partly because it has been in existence for a long time. The grouping of customers into routes is based on their saving value.

  Definition 71. *Saving value:*

  $$S(m_a, m_b) = L(d, m_a) + L(d, m_b) - L(m_a, m_b), \tag{4.5}$$

  where m_a and m_b are customers assigned to the same depot, $L(a, b)$ is the distance between points a and b, and d is the depot.

  A saving matrix is created from these values for each pair of customers of the same depot. In the following step, the distances are calculated for the routes; the obtained values are the values of the cost function $C()$.

  Definition 72. *Cost function:*

  $$C(R_i) = L(d, m_{i,1}) + \sum_{p=1}^{k_i - 1} L(m_{i,p}, m_{i,p+1}) + L(m_{i,k_i}, d), \tag{4.6}$$

  if $R_i = [d, m_{i,1}, m_{i,2}, \ldots, m_{i,k_i}, d]$.

  After the saving matrix is computed, customers are organized in descending order - this is the so-called saving list - and grouped into routes so that the capacity constraint of the route is not violated.
  Several other - newer, more sophisticated - algorithms exist for this task; see comparisons of the C&W to genetic algorithms [147] or other heuristics like the ant colony algorithm and the Mole and Jameson algorithm in [148].

- The third and final subtask is scheduling, i.e., the ordering of customers along their route. This requires a solution for the Travelling Salesman Problem, which is often performed using heuristics, for example, the nearest neighbour heuristic (NNH) [104]. NNH selects the first customer of the root randomly, then periodically looks for the customer nearest to the previous one. This step results in an initial feasible solution.

When computational speed is preferred to the solution's quality, the random method can be used for each sub-task.

Different methods can be applied for improving the initial feasible solution, for example, tabu search [149], simulated annealing [150], intra- and inter-depot or route improvement, and genetic algorithm [104]. Improvement can be achieved by copying biological evolution, too, for solutions of VRP [151–155]. As was presented in detail in chapter 3, genetic algorithms work in this manner. Usually, GAs are also applied to VRP combined with other heuristic methods. These are reflected as Hybrid GAs [104, 156–161]. Several solution methods for different types of VRP with Time Windows are presented in detail in [162].

4.4 Fault-tolerant extension to the VRP solution

4.4.1 Basic assumptions

The proposed method for introducing fault tolerance is based on an existing VRP solution as its input. This is then improved by taking into account the possible unexpected breakdowns of the vehicles during their trips in an optimal way. The following basic assumptions (e.g., applicability constraints) are applied:

- *Single Depot.* It is a basic property of the considered VRP. In case of Multi-Depot problems, the offered method has to be extended in the first step by assigning the customers to depots. After that, the proposed algorithm can be applied without changes for each depot.

- *Customers do not have Time Windows.* This assumption narrows the offered method's applicability to problems where any customer can be visited anytime during the distribution service.

- *Customer demands are deterministic.* The proposed algorithm deals with DVRP, where events may happen dynamically after the start from the depot. However, all the properties of the problem - travelling costs, customer demands, etc. - are deterministic and do not change in time. In real life, it is often the case: the length - and thus, the cost - of each trip is determined by the route planning phase (before the start), and the customer demand changes very rarely after the vehicles start. The only stochastic factor is the location of vehicle breakdown: in the presented approach, it can happen with a uniform distribution over each customer location on the vehicle's route.

- *The vehicles start from the depot simultaneously.* The algorithm uses this assumption in the determination of the location of the vehicles at different time points. Unless this constraint, the location calculation has to be modified by taking into account the different starting times of the vehicles.

- *The time of unpacking is negligible.* This assumption is also used in the calculation of the location of the vehicles at different time points. When either the service time at the customers or the time of the repackaging at the broken-down vehicle is not negligible, it results in delay for the affected vehicles.

- *Vehicle breakdown always occurs at a customer, never along the route.* In the presented approach, the distribution of the possibility of vehicle breakdown is uniform over each customer location along the route. This approach approximates well the uniform distribution of vehicle failure on the whole of its route if the variance in the distances to neighbouring customer locations is small, and all the routes have similar lengths.

- *A vehicle breakdown is considered to be the only possible fault.* Although other types of fault - for example, road closures or a vehicle going the wrong way - may occur, only the breakdown of a vehicle is considered as a fault. The consequences of the handled fault type are that a subset of the customers has to be reallocated to another vehicle, and the size of the fleet of the vehicles decreases by 1.

- *Only one vehicle in total may break down.* Since the possibility of a vehicle breakdown is little in real life, the possibility of the breakdown of more than one vehicle during the same servicing cycle at a company is negligible. This constraint is reasonable for a small fleet and few customers.

- *Only one vehicle provides assistance; no cases are handled where several vehicles assist together.* This strong constraint could be released if the additional computation need of the routes' replanning at each assumed location of the breakdown for every vehicle can be satisfied.

- *Cooperative vehicles.* This is a basic feature of vehicles in a company's fleet, where they all are willing to cooperate. That is why the cooperation of the vehicles is not a strong constraint.

- *Similar vehicles with equal capacity.* Without the vehicles having equal capacities, certain situations may arise when a helper vehicle would need to visit the broken-down vehicle more than once to service all the remaining customers.

- *Equal and constant vehicle speed.* This assumption is far from reality; however, it is still applied, mostly in VRP literature, because of its influence on task complexity. It results that during a trip, the time that has passed is proportional to the distance traveled. This constraint causes that the vehicles' calculated locations at the assumed time of the breakdown only approximate the locations of a usual real-life case.

- *Symmetrical distances between points.* This assumption is often applied in VRP. In real life, this depends on the road map. The effect of this assumption is that inversions have no additional costs of the route schedule.

Prior - arbitrary - route planning is also assumed. Its result is used as the input of the proposed algorithm. For the sake of completeness, the presented algorithm also includes a route planning part (Step1-Step4 of subsection 4.4.3) that applies the Clarke and Wright savings method [104]; however, it is not an integral part of the fault-tolerant extension and can be substituted by any other VRP solution method that can be applied to solve VRP problems without time windows and with symmetrical distances between points.

4.4.2 Concepts and notations introduced for handling VRP solution in a fault-tolerant way

4.4.2.1 Notations related to the routes and their direction of execution

Any solution of a VRP without Time Windows can be the subject of the proposed fault-tolerant improvement method. The previously mentioned Clarke and Wright savings method (see section 4.3) was used in this thesis, followed by NNH without any improvement. The reason for choosing NNH was its simplicity - and thus its speed. This quick algorithm - that results in not always the optimum of the NP-hard TSP subproblem - can be substituted with another heuristic or metaheuristic algorithm. Since the presentation of this thesis's method needs an undirected route set - a solution of a VRP obtained by any solution method - as its input, it was allowed to choose any algorithm for the routing phase that produces an undirected route set. This work aims not to answer the question of how to determine the routes but to show in which direction the vehicles should start along these routes. That is why an algorithm was intended to be chosen that can be presented in an easily understandable way.

The obtained result of the VRP solution is an undirected route set

$$R = \{R_1, \ldots, R_r\} \tag{4.7}$$

with the following properties:

(a) each route is cyclic, starts and ends in the same depot

$$R_i = [d, m_{i,1}, m_{i,2}, \ldots, m_{i,k_i}, d], \tag{4.8}$$

where d is the depot, $m_{i,j}$ is the j^{th} customer of vehicle n_i, the number of customers who are serviced by vehicle n_i is k_i, and

(b) each customer is visited

$$\bigcup_{i=1}^{n} E_i = \{M, d\}. \tag{4.9}$$

Moreover, the route set satisfies (4.2) and (4.4), too.

The input of the fault-tolerant improvement algorithm is an undirected route set R in (4.7). This arbitrary solution of a VRP without Time Windows (all the customers can be visited any time during a trip) will be improved so that, on average, the best response is provided to an unexpected event. The algorithm is based on the fact that in an optimal route set of a VRP, each route can be represented as a cycle (which starts and ends in the depot), and each vehicle has two possibilities to start its journey: it can drive either to the first or to the last customer of the planned route.

The following two notations will be used for the "route direction compositions" of the route R_i of vehicle n_i (superscript "F" means forward, and superscript "B" stands for backward):

Notations 4.4.1. *"Route direction composition"-related notations:*

R_i^F: $R_i^F = R_i = [d, m_{i,1}, m_{i,2}, \ldots, m_{i,k_i}, d]$, where the first customer visited by vehicle n_i is $m_{i,1}$.

R_i^B: $R_i^B = [d, m_{i,k_i}, \ldots, m_{i,2}, m_{i,1}, d]$ is the route where the customers are visited in the opposite direction than in R_i^F, so vehicle n_i starts his route by visiting customer m_{i,k_i}.

With these notations, one execution of a VRP solution can be described by one *route direction composition.*

Definition 73. *Route direction composition:*

$$R^C = \{R_1^C, \ldots, R_r^C\}, \quad where \qquad R_i^C \in \{R_i^F, R_i^B\}, \quad \forall\, 1 \leq i \leq r. \tag{4.10}$$

In the case of r routes, there are 2^r possibilities for executing an existing VRP solution.

Definition 74. *Possible route executions ($V(R^C)$): The set of possible route direction compositions.*

This thesis aims to find the route direction composition R^C that results in a minimal loss on average in the case of an unexpected event.

4.4.2.2 The fault model of the thesis

When vehicles start deliveries, it may happen that a vehicle has to stop and finish executing its task before having serviced all the customers previously assigned to it. The immediate stop means the service failure of the vehicle. This service failure results from an error propagated to the vehicle's internal service interface; see Figure 4.1. The cause of the error is a fault (usually a physical fault, e.g., the fault of the engine or any other hardware fault related to the vehicle that becomes active). The related concepts were defined by Definitions 8-10.

The broken-down vehicle that leaves behind unserviced customers means a fault from the viewpoint of the delivery service. If the service's state's deviation from the desired state (the error of the service) - caused by the fault - reaches the delivery service's internal service interface, delivery service failure occurs. In case of the considered problem, it means that unserviced customers remain after the delivery service finishes.

This thesis proposes a method that prevents a single fault of the delivery service from causing service failure. All the assumed constraints of the offered method are summarized in subsection 4.4.1. Here, a single fault means that exactly one vehicle breaks down before it finishes its service. This fault results in a set of unserved customers with well-known demands and a fleet reduced by one vehicle. The place of the single breakdown is modelled by a random variable that has a uniform distribution over the customer locations (including the depot at the start) as follows:

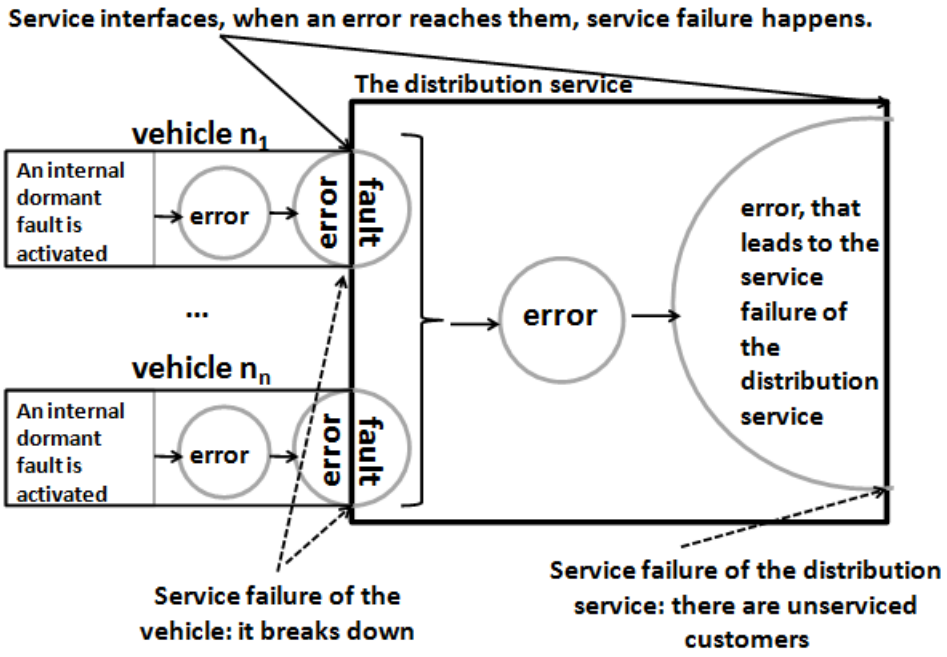


Figure 4.1. The chain of the handled dependability and security threats of a distribution service.

Using (4.8), if the probability of failure of vehicle n_i on its route between customer $m_{i,j}$ and $m_{i,j+1}$ is notated by $P_{n_i}(m_{i,j})$, it is assumed that

$$\sum_{i=1}^{r}(P_{n_i}(d) + \sum_{j=1}^{k_i} P_{n_i}(m_{i,j})) = 1, \tag{4.11}$$

and

$$P_{n_i}(d) = P_{n_i}(m_{i,j})), \quad \forall\ 1 \leq i \leq r \ \ and \ \ 1 \leq j \leq k_i. \tag{4.12}$$

Its consequence is that

$$P_{n_i}(d) = P_{n_i}(m_{i,j}) = 1/(m+n), \quad \forall\ 1 \leq i \leq r \ \ and \ \ 1 \leq j \leq k_i. \tag{4.13}$$

Taking into account "n" in (4.13) needs some reasoning. Breakdown may happen for any vehicle on any of its route segments (even when it starts from the depot and drives to its first customer). The total number of the problem's route segments is equal to $m+n$, as the example of Figure 4.2 demonstrates that. It has to be noted that if the breakdown of vehicle n_i occurs after it reaches its last customer (m_{i,k_i}), the fault does not lead to distribution service failure. This thesis aims to propose a fault-tolerant extension of VRP solutions, preventing one vehicle's breakdown from causing the distribution service's failure.

The requirement of reaching fault tolerance is that the fleet has more than one vehicle. It means that protective redundancy exists in the system that can take over the remaining tasks of the broken-down vehicle, thus trusted service can be delivered. If it is the case, 100% fault tolerance coverage can be reached (since a single fault was assumed).

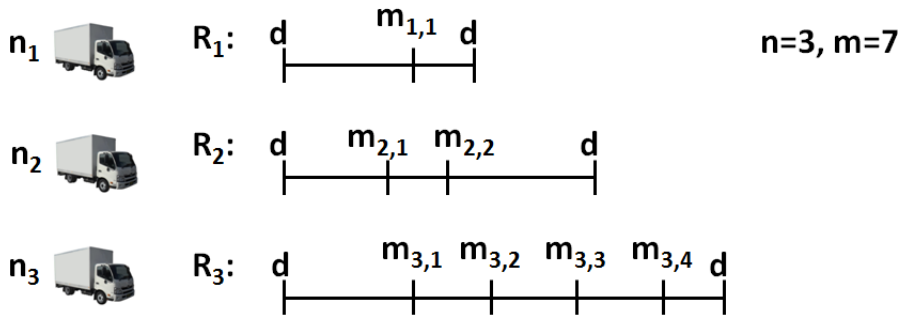


Figure 4.2. Example to explain the role of "$m + n$" in (4.13). The total number of route segments of this example is equal to $m + n = 10$.

Definition 75. *Coverage: A measure of the effectiveness of a fault-tolerant technique. The representativeness of the situations to which the system is subjected during its analysis compared to the actual situations that the system will be confronted with during its operational life [9].*

Formally, in the problem statement, a vehicle's breakdown appears as an additional customer set after the vehicles started. These customers have to be served by other vehicles than their original assignment shows. This way, recovery is made by reconfiguration [9]. As a result of the assistance provided, the helper vehicle's initially planned route must be modified, resulting in the dynamic/stochastic nature of the problem. The goal of this thesis's algorithm is to minimize the negative effect of such assistance by keeping the extra route length minimal on average in the case of an unexpected event. Of course, the extra route - the loss of a solution - is determined by the location of the breakdown, the selection of the helper vehicle, the helper vehicle's location, and the new route of the helper vehicle. Because any vehicle of the fleet can break down and the location of the breakdown can be anywhere on a vehicle's route, the goal was to find the optimal solution on average.

4.4.2.3 The average loss of a solution

For any unexpected event, the vehicle offering assistance with the minimum cost shall be found. For this, the concept of "route change cost" (*rcc*) is introduced.

Definition 76. *Route change cost (rcc): The difference between the total length of the modified route of the helper vehicle and the length of its original route.*

The main purpose of the proposed algorithm is to minimize *rcc*. Taking into account vehicle capacity constraints, the calculation of *rcc* can be performed as follows.

Suppose that vehicle n_b is the broken-down vehicle with its original route $R_b^C = R_b^F = [d, m_{b,1}, m_{b,2}, \ldots, m_{b,k_b}, d]$, and vehicle n_h is the helper one with its original route $R_h^C = R_h^B = [d, m_{h,k_h}, \ldots, m_{h,2}, m_{h,1}, d]$. Moreover, suppose that at the moment of breaking down, vehicle n_b has already serviced its customer $m_{b,i}$, and vehicle n_h has left its customer m_{h,k_h-j}. Furthermore, the location of the breakdown

is denoted as p. Now, the route of the helper vehicle has to be modified as follows:

$$R_h^{Cmod} = [d, m_{h,k_h}, \ldots, m_{h,k_h-j}] \wedge R_h^+, \tag{4.14}$$

where $\wedge$ denotes the concatenation operation, and R_h^+ is the new route of the helper vehicle after the moment of the breakdown of vehicle n_b. R_h^+ is an ordered list (vector) of the elements of the set $\{m_{h,k_h-j-1}, \ldots, m_{h,1}, m_{b,i+1}, \ldots, m_{b,k_b}, \{p|\emptyset\}\}$ finished by d (when the visit of the broken-down vehicle is not needed, p is not included in R_h^+ that is indicated by the element $\{p|\emptyset\}$). Thus the continuation of the route of the helper vehicle is constructed from the elements of the union of the remaining customers of the helper and the broken-down vehicles (plus, in some cases, the location of the broken-down vehicle).

Next, M_B^r and M_H^r are introduced.

Notations 4.4.2. *Notations related to the remaining customers:*

M_B^r: The set of the remaining customers of the broken-down vehicle at the moment of breakdown.

M_H^r: The set of the remaining customers of the helper vehicle at the moment of breakdown.

In the example above

$$M_B^r = \{m_{b,i+1}, \ldots, m_{b,k_b}\} \tag{4.15}$$

and

$$M_H^r = \{m_{h,k_h-j-1}, \ldots, m_{h,1}\}. \tag{4.16}$$

Because of the vehicle capacity constraints (4.4) of the VRP, the construction of R_h^+ is not trivial - this is described in detail in Step D of subsection 4.4.4.

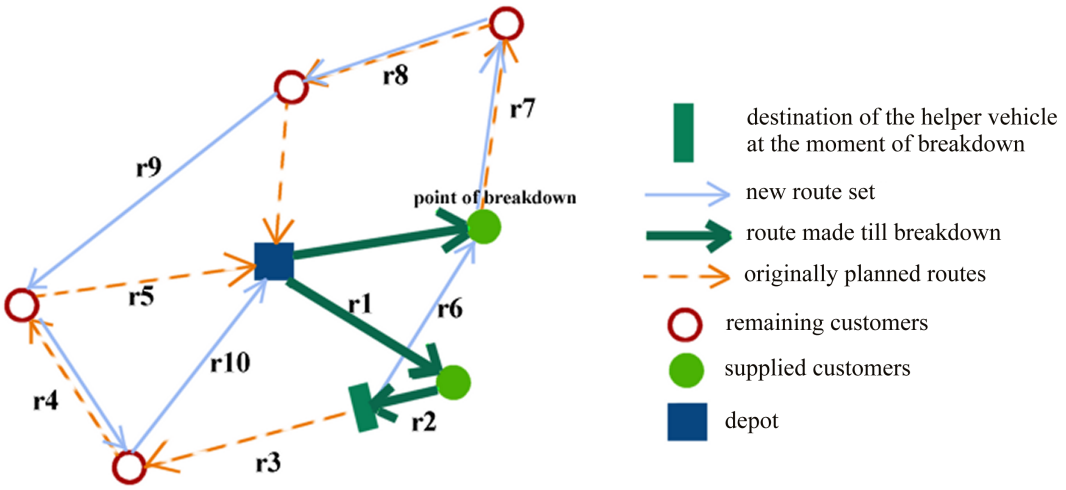


Figure 4.3. Calculation of route change cost (rcc_h) value - an example.

Finally, the route change cost of vehicle n_h is calculated as:

$$rcc_h = C(R_h^{Cmod}) - C(R_h^C). \tag{4.17}$$

Figure 4.3 shows the loss in a simple example. In this example, the helper vehicle has enough free capacity after servicing its first customer to take over the broken-down vehicle's remaining goods. The extra costs, i.e., the route change cost, of the helper vehicle is calculated as follows:

$$rcc_h = r1 + r2 + r6 + r7 + r8 + r9 + r4 + r10 - (r1 + r2 + r3 + r4 + r5) \tag{4.18}$$

When computing the average of the route change costs, it needs to be kept in mind that a breakdown can happen any time during a vehicle's trip. As the list of remaining customers has to be modified whenever a vehicle reaches a customer, all the time points when customers are reached are noted to calculate the route change cost of any unexpected event, supposing that a vehicle having just reached a customer breaks down. For each point, the helper vehicle is selected as follows:

$$n_i, \quad where \quad i = arg\ min_\ell(rcc_\ell).$$

At each reached customer m_j the calculated minimal route change cost value is denoted as $mrcc_j$. Having determined the helper vehicle with the minimal route change costs for all reached points, the average loss of a solution is defined as follows:

Definition 77. *Average loss of a solution ($avloss_k$):*

$$avloss_k = \frac{\sum_{l=1}^{m} mrcc_l}{m}, \quad where \quad 1 \leq k \leq 2^r. \tag{4.19}$$

Finally, the route direction composition R^{C_k} (see Definition 73) is selected from the possible route executions $V(R^C)$ that minimizes $avloss_k$ in (4.19). This is regarded as the best route execution on average of the solution of the original VRP for each breakdown.

4.4.3 The algorithm for determining the best route direction composition

It is supposed that there are m customers at known locations and one single depot (d). An algorithm consisting of 9 steps is proposed below for determining the best route direction composition (its flowchart is given in Appendix Q). The first 4 steps only calculate the initial VRP solution using the Clarke and Wright savings method [104]. These steps can be substituted by the steps of any other VRP solution method that can be applied to solve VRP problems without time windows and with symmetrical distances between points.

Step 1: Determine the distances between each customer pair and the distances between customers and the depot. Result: a distance matrix.

Step 2: Calculate the savings matrix based on the Clarke and Wright savings method.

Step 3: Arrange the customer pairs in descending order based on their saving values. This way, the so-called saving list is obtained.

Step 4: Process the saving list as Clarke and Wright suggested: Start with the customers at the beginning of the saving list, process the list to its end, and organize the customers into routes until capacity constraints allow. After that, the creation of a new route begins.

At the end of Step 4, a solution of the VRP is obtained in the form of a set of routes (a set of cycles) consisting of r routes.

Step 5: Create a new direction composition (R^C) for the set of routes.

Step 6: Arrange customers in increasing order based on the time when the servicing vehicle reached them. This way, the list of arrivals is obtained.

Step 7: Process the list of arrivals from its beginning to its end. The last customer of any vehicle is ignored because if a vehicle breaks down (on its way back to the depot) after servicing the last customer, no assistance is required. Calculate for each customer in the list how much a breakdown costs at that point.

In particular, perform the sub-steps below first for all vehicles as they start from the depot, then for each (m_b) element of the list of arrivals except the last customers of any vehicle (the servicing vehicle for customer m_b is n_b):

> **Step 7.1**: Calculate the location of the other vehicles at the time instant when vehicle n_b arrives at customer m_b.
>
> **Step 7.2**: Determine the route change cost (rcc) value - as illustrated in Figure 4.3 and in (4.18) - for each vehicle (except for n_b) as if it would service all the remaining customers of vehicle n_b. Next, determine the minimum of the resulted rcc values. This is the cheapest possible rcc for the considered possible breakdown point.

Step 8: Summarize the costs calculated in Step 7.2 for the whole list of arrivals (including the starting point - the depot - for all vehicles) and divide it by the length of the list. This is the average loss of the considered directed route set.

Step 9: If there is a non-investigated element of the directed route set $V(R^C)$, then go to Step 5, else select the directed route set with the minimal average loss as the best solution.

The result of the algorithm is a route direction composition $R^C = \{R_1^C, \ldots, R_r^C\}$, where $R_i^C \in \{R_i^F, R_i^B\}$, $\forall\ 1 \leq i \leq r$ and for which the average loss value in (4.19) is

the minimum average loss value.

The number of computational steps of the proposed algorithm is $O(r * m^2 * 2^r)$ (i.e., it is exponential in the number of routes r) - assuming that an initial VRP solution is created before, for example by the Clarke and Wright algorithm, which has $O(m^2 * log(m))$ complexity -, where m is the number of the customers. The execution of the proposed algorithm is demonstrated on a simple example in subsection 4.5.1.

4.4.4 The proposed execution - determination of the helper vehicle and its modified route

Having determined the route direction composition which reacts the best to a vehicle breakdown on average, the next steps should be carried out for the execution to be optimal:

Step A: Start the vehicles from the same (the only one) depot at the same time.

Step B: Unless an unexpected event happens:
for each vehicle:
if there are remaining customers:
drive to the following one on the route and service it;
goto Step B;
if there is no more customer to service:
drive to the depot;
end;
If an unexpected event happens and it is the first time:
goto Step C.
else **Error Stop**.

Step C: Determine the actual location of all the vehicles of the fleet. Determine the set of the remaining customers of the broken-down vehicle M_B^r, introduced in (4.15). Determine the actual free capacity of all other vehicles n_i:

$$x_i - \sum_{j|m_j \in M_H^r} d_j, \quad where \quad H = i \quad (see \quad (4.16)). \tag{4.20}$$

Step D: For all vehicles (n_i) except for the broken-down one, do:
Starting from the actual location of n_i, order the elements of the sets M_B^r and M_H^r, $H = i$ into the route RF (which is composed of the remaining customers of n_i and the remaining customers of the broken-down vehicle) using NNH. Finish the route by d. Determine the first customer (m_f) whose demand cannot be satisfied because of the actual quantity of the goods carried by the vehicle:

$$\begin{aligned} m_f, \quad f = min(j \mid \sum_{k=1}^{j} d_{f_k} > A, \quad where \quad RF = [q, m_{f_1}, \ldots, m_{f_l}, d], \\ m_{f_k} \in M_B^r \bigcup M_H^r \quad and \quad l = |M_B^r| + |M_H^r|). \end{aligned} \tag{4.21}$$

In the equation, q denotes the location of n_i, and A is the amount of goods carried by n_i at the moment of breaking down.
Starting from the end of RF and stepping backward determine the first customer of RF (m_g) whose demand can not be satisfied because of the capacity of the vehicle:

$$m_g, \quad g = max(l - j \mid \sum_{k=0}^{j} d_{f_{l-k}} > x_i, \quad where \quad RF = [q, m_{f_1}, \ldots, m_{f_l}, d]). \quad (4.22)$$

In the equation, q denotes the location of n_i and x_i is the capacity of n_i.
If m_g is not found, $m_g = q$.
Insert the broken-down vehicle's location (p) into the route between m_g and m_f with the shortest extra cost (route length).

Calculate rcc for n_i.

Step E: Select the vehicle with the minimal rcc as the helper vehicle and modify its route according to the result of the calculations above. Go to Step B.

Remarks
There are two extreme cases in the execution of this algorithm:

- The amount of goods on the helper vehicle is sufficient for servicing the remaining customers (the goods on the broken-down vehicle do not need to be used). In this case, the faulty vehicle does not have to be visited.
- The helper vehicle's capacity is not sufficient for servicing the remaining customers with visiting the broken-down vehicle only once. The proposed algorithm does not deal with this case: if a fleet composed of homogeneous vehicles is supposed, this situation cannot occur. In the worst case, the helper vehicle services all its original customers, drives to the broken-down vehicle, takes over the remaining goods, and services the broken-down vehicle's remaining customers.

The proposed algorithm minimizes the extra route length resulting from the modification of the helper vehicle's route without influencing the goodness of the initial route set unless a breakdown of a vehicle happens. The reason for choosing NNH in Step D was its simplicity and speed since when the breakdown occurs, the immediate answer - that is based on online computations - is crucial. An example for the execution of the algorithm is given in Appendix R.

4.5 The efficiency of the fault-tolerant extension

I have implemented the described algorithm in C language and used this program to evaluate its results. For demonstrating its efficiency, two types of case studies are presented as examples. By them, the execution of the directed route set the proposed method chooses is compared to the other directed route sets of the same VRP solution. The first example is a simple one that helps understand simply and

Table 4.1. Location and demand of the customers and the depot of the simple example.

	Depot	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9
X coord.	35	41	35	55	55	15	25	20	10	55
Y coord.	35	49	17	45	20	30	30	50	43	60
Demand (unit)	0	10	7	13	19	26	3	5	9	16

transparently how the proposed execution works. The second group of examples is taken from Solomon's instances [163], which are used as a standard for verifying and comparing VRP solver algorithms.

4.5.1 Simple VRP example for analyzing the fault tolerance of the developed algorithm

The simple example

In the simple example, 9 customers have to be serviced from a common depot. The location and demand of the depot and the 9 customers (m_1-m_9) are highlighted in Table 4.1. Moreover, the location of the depots and the customers is illustrated in 2D in Figure 4.4. The depot is signed with the largest circle. The capacity of each vehicle is equally 50 units.

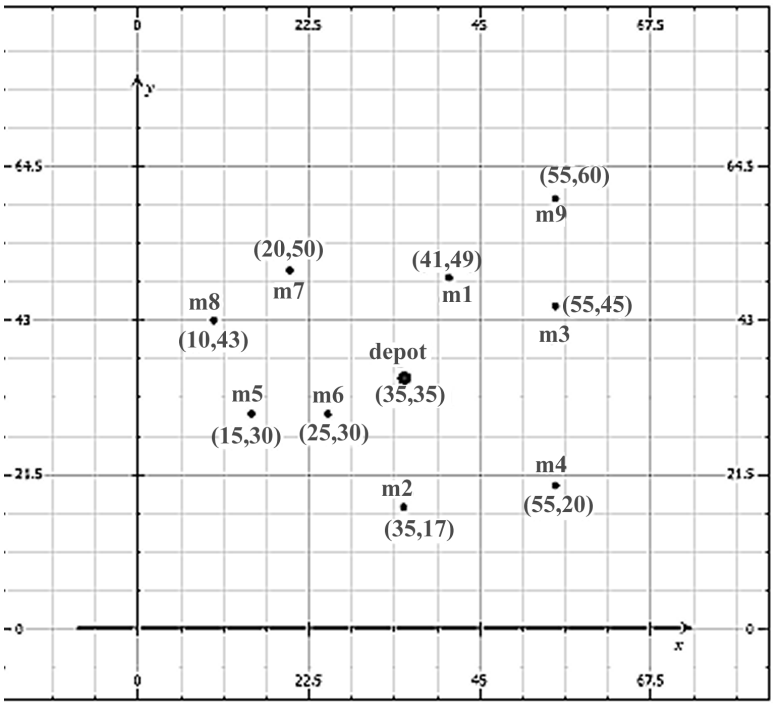


Figure 4.4. Location of the customers and the depot of the simple example.

The proposed method applied for the simple example

First, the distance matrix (bottom triangle matrix in Figure 4.5) is calculated based on *Step* 1 of the algorithm presented in subsection 4.4.3. Second, the saving matrix (upper triangle matrix in Figure 4.5; where the columns and rows represent the customers, the 0^{th} element is the depot) is determined (*Step* 2).

half part of savings matrix

0	0	0	0	0	0	0	0	0	0
15	0	1	23	8	3	2	15	10	30
18	32	0	6	23	15	13	3	8	3
22	14	34	0	22	0	0	8	3	39
25	32	20	25	0	4	5	0	1	17
20	32	23	42	41	0	21	21	33	2
11	24	16	33	31	10	0	12	18	1
21	21	36	35	46	20	20	0	35	17
26	31	36	45	50	13	19	12	0	10
32	17	47	15	40	50	42	36	48	0

half part of distance matrix

Figure 4.5. Distance and saving matrix of the simple example.

After this, the saving list is built up (*Step* 3), and the Clarke and Wright savings method is applied (*Step* 4). As its result, three routes are obtained, as illustrated in Figure 4.6. The obtained route lengths are 69, 67, and 63 units.

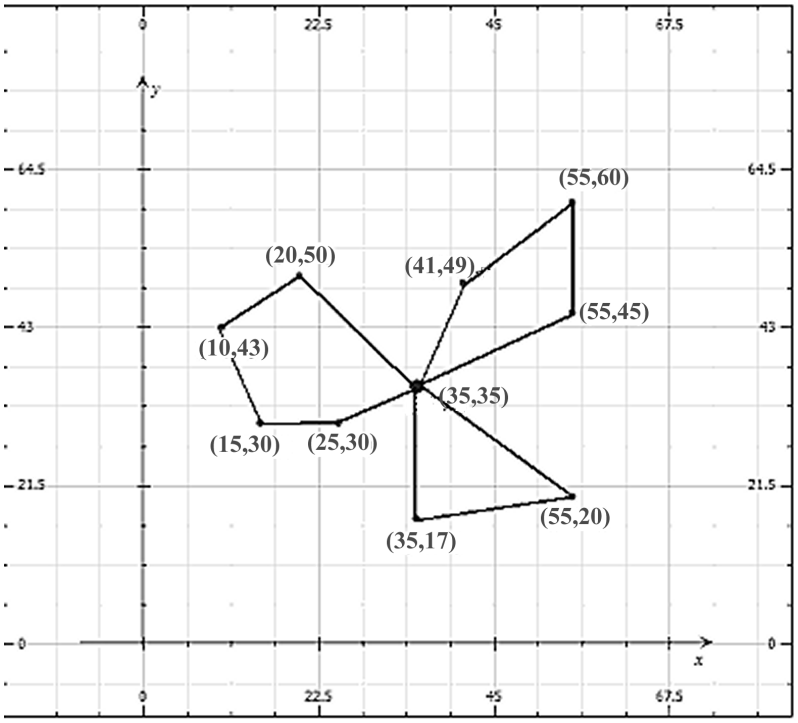


Figure 4.6. VRP solution for the simple example by Clarke and Wright savings method.

One route direction composition (R^C) - see (4.10) - of the example can be seen in Table 4.2. There are three routes; the first one visits customer m_3 first, then customer m_9, and finally customer m_1 before getting back to the depot. The second group, named "Distances", shows the length of the routes that have to be covered by the vehicles between the depot and the customers, cumulative (the total distance left before getting back to the depot is shown at the end of the rows). For example, the first vehicle has to cover 37 distance units from the depot to arriving at customer m_9, and the total length of its route is 69 units.

Table 4.2. One route direction composition, its distance data, and the list of arrivals of the simple example.

Routes (illustrated by ordered customers)					
the route of vehicle n_1	m_3	m_9	m_1		
the route of vehicle n_2	m_7	m_8	m_5	m_6	
the route of vehicle n_3	m_2	m_4			
Distances (time units, the last element belongs to the depot)					
of vehicle n_1	22	37	54	69	
of vehicle n_2	21	33	46	56	67
of vehicle n_3	18	38	63		

The list of arrivals		
Customer	total distance from the depot	route /vehicle/
m_2	18	n_3
m_7	21	n_2
m_3	22	n_1
m_8	33	n_2
m_9	37	n_1
m_4	38	n_3
m_5	46	n_2
m_1	54	n_1
m_6	56	n_2

Table 4.2 also shows the third group, "the list of arrivals", below the distance data. This is an ordered list of customers, ordered by the time when customers are visited. The three columns are showing the customer ID (first column), the total distance a vehicle has to cover from the depot until it arrives at this customer on its route (second column), and the vehicle ID (third column). Except for the vehicles' final customers, the route change cost (*rcc*, see (4.17)) is calculated at each point of the list of arrivals for all vehicles but the one whose customer is in the list's actual position (*Step* 6 *and Step* 7). This calculation starts in the depot (before processing the list), as Table 4.3 illustrates (*rcc* stands for "route change cost").

As the example contains 3 routes (3 vehicles), there can be 3 variations in the broken-down vehicle. The first row of Table 4.3 represents the case when vehicle n_1 breaks down. It can be seen that in this case, vehicle n_2 should service 7 customers (since vehicle n_2 originally had 4 customers and vehicle n_1 had 3), or vehicle n_3

Table 4.3. Route change cost (*rcc*) calculation for a route direction composition in the depot.

Broken-down vehicle	Helper vehicle	left nodes for the helper	rcc	the best rcc for one reached point
n_1	n_2	$m_6, m_5, m_8, m_7, m_1, m_3, m_9$	78.3	77.0
	n_3	m_1, m_3, m_9, m_4, m_2	77.0	
n_2	n_1	$m_6, m_5, m_8, m_7, m_1, m_3, m_9$	76.3	68.8
	n_3	$m_6, m_5, m_8, m_7, m_2, m_4$	68.8	
n_3	n_1	m_1, m_3, m_9, m_4, m_2	71.0	64.8
	n_2	$m_6, m_5, m_8, m_7, m_2, m_4$	64.8	

Table 4.4. Route change cost (*rcc*) calculation for a route direction composition after driving 33 distance units. Letter p indicates visiting the broken-down vehicle.

Broken-down vehicle	Helper vehicle	left nodes for the helper	rcc	the best rcc for one reached point
n_2	n_1	$m_9, m_1, m_6, \boldsymbol{p}, m_5$	65.0	61.2
	n_3	$m_4, m_6, \boldsymbol{p}, m_5$	61.2	

should service 5 customers. In the next column, the *rcc* value is presented for the two vehicles. The other two cases can be interpreted in the same way. As vehicles go further on their routes, circumstances change. After 33 distance units (the 4^{th} element of the list of arrivals), the *rcc* calculation is illustrated in Table 4.4, where the letter "p" indicates visiting the broken-down vehicle in the remaining route.

Since vehicle n_2 arrives at customer m_8 after driving 33 distance units on its route, this point represents the case when vehicle n_2 breaks down (somewhere between customer m_8 and m_5), and its remaining two customers (m_5 and m_6) should be serviced by either vehicle n_1 or vehicle n_3. It can be seen that vehicle n_1 has a higher *rcc* value than vehicle n_3. Naturally, in this case, vehicle n_3 will be the helper vehicle. *Rcc* is calculated for all elements of the list of arrivals (including the starting point of each vehicle as illustrated in Table 4.3), except for the vehicles' final customer. In the example, the final customers are elements 6, 8, and 9 of the arrival list. In *Step* 8, the minimum values of each iteration of the calculation (the helpers' *rcc* values) are averaged. This value is calculated for all route direction compositions.

The obtained results for the simple example

In this example (because of the 3 routes), 8 route direction composition variations can be determined. Two of them are illustrated in Table 4.5. It can be seen that in the second route set, the final two routes are travelled in reverse order.

The averaged *rcc* values are calculated for all of the 8 possible travelling orders, and the minimal one is selected as the best solution to the problem. The averaged

Table 4.5. Two route direction compositions of 8 total of the simple example.

Directed route set I.	
Vehicle	the ordered sequence of the customers
n_1	m_3, m_9, m_1
n_2	m_7, m_8, m_5, m_6
n_3	m_2, m_4

Directed route set II.	
Vehicle	the ordered sequence of the customers
n_1	m_3, m_9, m_1
n_2	m_6, m_5, m_8, m_7
n_3	m_4, m_2

Table 4.6. The efficiency of the proposed algorithm regarding the simple example.

Averaged route change costs for the route direction composition variation number							
I.	II.	III.	IV.	V.	VI.	VII.	VIII.
57.53	57.16	56.87	57.34	**53.82**	54.81	55.75	55.60
The smallest averaged route change cost - the result of the algorithm: **53.82**							
The highest averaged route change cost - the worst possible choice: 57.53							
How much (in %) the proposed solution is better related to the worst possible case: 6.4%							
The mean averaged route change cost: 56.11							
How much (in %) the proposed solution is better related to the average: **4.1**%							

rcc values of the 8 cases of the simple example are illustrated in Table 4.6; the details can be seen in Appendix S. Even in the simple example, a solution can be chosen for the case of an unexpected event that is better than the other possibilities (*Step* 9 of the algorithm presented in subsection 4.4.3). Table 4.6 shows that the cost of the best result is only 93.55% of the solution with the highest cost. If the average cost of all possible route direction compositions is considered and the proposed algorithm is applied, the planned choice of starting directions improves the result by 4.1%. Note that in all cases, the same route set (the result of a common VRP solution method) is used; the only difference is in the chosen direction of execution.

4.5.2 Efficiency analysis of the proposed algorithm on the Solomon instances

The impact of the proposed algorithm was also investigated by using standardized VRP test instances. Solomon instances [163] were chosen for the test; this data set is a common test set in the VRP literature.

The Solomon instances

M.M. Solomon generated 6 sets of VRP-TW instances (R1, R2, C1, C2, RC1, and RC2). Each problem has one depot and 100 customers with well-determined locations, demands, time windows, and service times. This thesis does not deal with time windows and service times; only the location data and the demands defined by the instances are used. The 6 groups of the problems differ - besides the time-related data - in the customers' locations' relation - except groups R1 and R2 - ("R" stands for randomized and "C" denotes clustered placement). Another difference between the 6 instance groups is the capacity of the vehicles (in groups R1, C1, and RC1, each vehicle has a capacity of 200, in C2, the capacity is 700, finally, in groups R2 and RC2, the value of the capacity is 1000). Instances of a group differ only in the time-related parameters; the customers' locations in a group are identical.
For example, the RC202 instance (that is an element of the RC2 instance group) contains a mix of clustered and randomly generated customers and allows long routes to be created. The number of customers is 100, and the vehicle capacity is 1000. More specific data (e.g., locations, demands) can be found in [163]. The location of the 100 customers (small circles) and the depot (marked with the largest circle) can be seen in Figure 4.7.

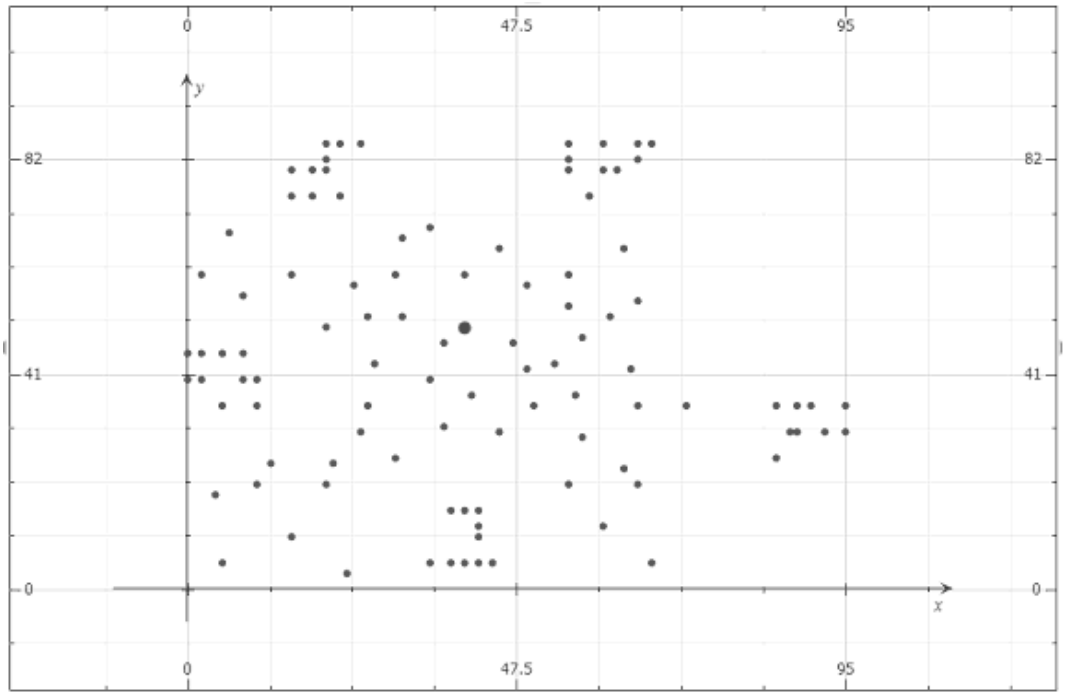


Figure 4.7. Location of the depot and the 100 customers of Solomon's RC202 instance.

The results of the proposed method for the Solomon instances

First, the results of the proposed algorithm are detailed for Solomon's RC202 instance, then the results for all the six groups of Solomon's instances (without time-specific data) are summarized.

Executing *Step* 1−*Step* 4 of the algorithm of subsection 4.4.3 for the RC202 instance results in the initial route set illustrated in Figure 4.8.

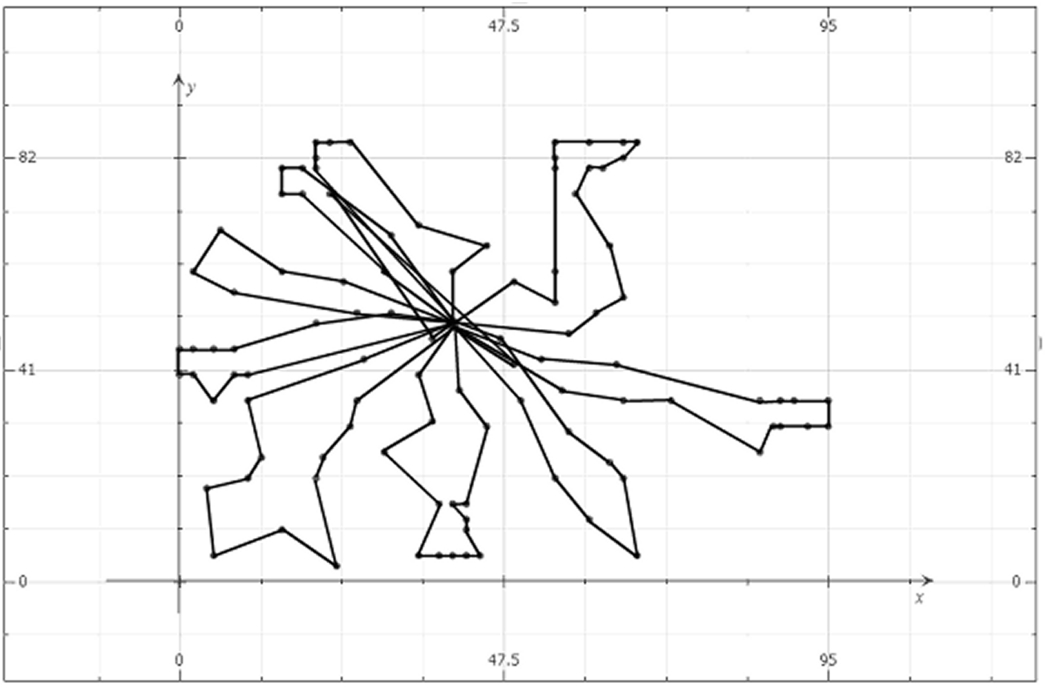


Figure 4.8. The route set, obtained for Solomon's RC202 instance.

This route set is composed of 10 routes. This means that the algorithm has to deal with 1024 differently directed route compositions. Among the obtained solutions, the minimal averaged route change cost has a value of 53.29, the maximal value is 65.38, the average value of the route change costs is 58.22, and the standard variance of the 1024 cases is 2.22. This means that, when an unexpected event occurs, the extra cost of the best solution - the cost originating from helping the broken-down vehicle - is 81.5% of the extra cost of the worst solution and 91.5% of the extra cost of the average of all the cases. The obtained values - completed with the results got for the other instance groups of Solomon - are presented in Table 4.7.

The algorithm was tested on all types of Solomon's instances [163]. The result of the fault-tolerant route planning algorithm on the groups of Solomon instances is summarized in Table 4.7. The columns of the table contain the minimal and the maximal *rcc* values (see 4.17), the standard variance and the mean of the *rcc* values for the different route direction compositions, the ratio of the best and the worst case, and the ratio of the best and the average case per Solomon instance groups. The histograms of the averaged minimal *rcc* of the possible route direction compositions for each instance group can be found in Appendix T.

The results show that the application of the algorithm improves the efficiency of route execution in all cases. Of course, the improvement is the largest - even 10%-15% - compared to the worst possible response to the unexpected event. However, the algorithm also ensures a 4%-8% improvement compared to the average in the case of a vehicle breakdown. The results show that the improvement is bigger when there are clustered customer locations and is smaller for random locations. Its reason

Table 4.7. The efficiency of the algorithm for the different Solomon instance groups.

	Name of the Solomon instance group					
	C1	C2	R1	R2	RC1	RC2
The smallest averaged route change cost; the result of the algorithm	**42.17**	**49.79**	**62.39**	**62.39**	**54.91**	**53.29**
The highest averaged route change cost; the worst possible choice	47.27	59.40	67.41	67.41	64.41	65.38
How much (in %) the obtained solution is better related to the worst possible case	10.8%	16.2%	7.4%	7.4%	14.7%	18.5%
The mean averaged route change cost	44.68	54.48	65.05	65.05	59.39	58.22
The standard variance of averaged route change costs	1.14	2.03	0.82	0.82	1.75	2.22
How much (in %) the obtained solution is better related to the average	**5.6%**	**8.6%**	**4.1%**	**4.1%**	**7.5%**	**8.5%**

can be that clustered customers determine more strictly the route that they should be assigned to, and an inefficient selection of a helper vehicle causes longer extra routes than in the case of random - thus, not so strictly separable - locations.

4.6 The contribution of this chapter

The contribution of this chapter is threefold: (1) it proposes an algorithm that finds the route direction composition of any solution of an arbitrary Capacitated Single Depot Vehicle Routing Problem without Time Windows, that tolerates the best the breakdown of one vehicle, on average, assuming the constraints of subsection 4.4.1, (2) it gives an algorithm that determines the helper vehicle and the route modifications for the cases when a vehicle of the fleet breaks down during route execution and (3) it determines the efficiency of the proposed fault-tolerant extension on a simple example and the well-known VRP benchmark problems of Solomon.
When a vehicle breaks down, the proposed method counts on minimizing the extra cost of the breakdown by applying a greedy method: inserting the remaining customers into the closest route (keeping to the constraint of using exactly one helper vehicle). The total replanning of the routes - which is the common method found in literature - is NP-hard.
For the investigated problems, the proposed algorithm ensures 4%-8% improvement compared to the average in the case of a vehicle breakdown. These problems include both random and clustered customer locations. The average efficiency improves

when the customers are clustered.
Due to the variations in the number of customers, computation times differ highly. The calculation of the results in the simple example presented in subsection 4.5.1 took 0.487 seconds, while for Solomon's instances, it took 81 seconds on average using an HP Compaq tc4200 tablet PC with Intel Pentium M 2.00 GHz processor and 1 GB RAM. Because these calculations have to be carried out before the route execution during the route planning process, the delay in computation does not influence the logistic system's performance.

4.6.1 The problem class that can be solved by the developed algorithms

The proposed algorithms can be applied to any Capacitated Single Depot Vehicle Routing Problem without Time Windows with the constraints of subsection 4.4.1. It means that the customer demands are known even in the route planning phase; at most one vehicle breaks down, the vehicles are cooperative and have the same capacity, they start from the common depot simultaneously, the time of unpacking is negligible, and the distances between points are symmetric. The other constraints of subsection 4.4.1 influence the quality of the solution provided by the proposed algorithm; however, they do not influence the covered problem class. When resolving these simplifications, the computational need increases. Both the consequences and solution proposals for the cases of the resolutions of the applied constraints are briefly described in subsection 4.4.1.

Chapter 5

Conclusions

This chapter concludes this thesis: summarizes the previously presented work's scientific contribution and suggests directions for future research.

5.1 New scientific results

In this section, the new scientific contributions are summarized.

Thesis 1. *I have created a heuristic algorithm to solve an extended FJSP scheduling problem for mass production. The algorithm decomposes the initial complex problem into subproblems and solves them one-by-one. The trickiest part of the algorithm is that it assigns the operations to jobs only during its scheduling phase. Finally, it builds up the solution of the initial problem from the solutions of the subproblems.*

1.1 I have compared the results of the heuristic algorithm with the result of a MILP solver on 1000 random inputs. In about half of the cases, the heuristic solver found the optimum. The relative error was at most 2% in 80% of the cases and at most 5% in 95% of the cases. The computation time of the heuristic solver was 2 minutes *altogether for the* 1000 inputs. That means 0.12 seconds for one instance on average. The exact solver's calculation usually took more time: there were cases when 10000 seconds were not enough for getting the optimal schedule for *one* instance. Based on these results, it can be stated that the heuristic algorithm is efficient.

1.2 I have analyzed how much the heuristic algorithm can support the exact (CPLEX) solver. The standalone exact solver's efficiency was compared with different combinations of the heuristic algorithm and the exact solver. Applying the heuristic algorithm's result as the exact solver's upper bound was usually much more efficient than the standalone execution of the exact solver. The best results were obtained when the lower bound of the exact solver was increased by a logarithmic step size (calculated with the help of the result of the heuristic solver).

The corresponding publication is [124]; details can be found in Chapter 2.

Thesis 2. *I have developed and improved a genetic algorithm (GA) that can solve an extended FJSP scheduling problem with transportation devices (the considered problem is much more general than the problem of Thesis 1). I have investigated the algorithm's parameter settings and determined its efficient values for the analyzed problem (population size: 100, crossover rate: 0.3, mutation rate: 0.38). The improvement covers the creation of the initial population, the design of the genetic operators and the selection mechanism, and the development of the fitness function.*

2.1 I have compared and combined the GA with other scheduling methods. I have determined a coarse lower bound for the problem, and I have applied simulated annealing (SA) both with short and long runtimes (SA-short and SA-long) to solve the problem. I have also combined GA with SA. For the considered problem, GA was able to approach the calculated lower bound. SA-short found worse results than GA but did it much quicker. SA-long was similar to GA in both quality and computation time. When the result of SA-short was placed into the first generation of the GA, a quicker initial improvement was experienced; however, the final results were not better than without SA.

2.2 I have tested the GA on some well-known FJSP benchmark problems. I have improved the GA on these test problems, and finally, the algorithm approached (even, in some cases, it reached) their known best solution.

The corresponding publications are [90, 164, 165]; details can be found in Chapter 3.

Thesis 3. *I have developed an algorithm that determines the route execution directions for a solution of a Capacitated Single Depot Vehicle Routing Problem without Time Windows that minimizes the extra cost of a recourse action in case of a vehicle breakdown, on average.*

3.1 I have suggested an efficient route execution - including the recourse action - for the cases when one vehicle breaks down.

3.2 I have analyzed the efficiency of the proposed algorithm both for a simple example and for the instances of the classic, popular VRP benchmark: Solomon's test set.

(a) In the case of the simple example, the cost of the best result was only 93.55% of the solution with the highest cost. Considering the average cost of all possible route direction compositions and applying the algorithm, its suggestion for start directions improved the result by 4.1%.
(b) For Solomon's instances, I have got 10%-15% improvement when the cost of the selected route direction composition was compared to the worst possible response to the unexpected event. The result of my algorithm also ensured a 4%-8% improvement compared to the average in the case of a vehicle breakdown. I have also experienced that the improvement is bigger when customer locations are clustered and is smaller for random locations.

The corresponding publication is [125]; details can be found in Chapter 4.

5.2 Suggestions for future research

This section summarizes the suggestions for future research related to the thesis points.

1. The most significant question in the continuation of the work presented in Chapter 2 is how to separate a complex problem into subproblems. In the chapter, I gave two possible necessary conditions for separability: (1) the operations that can be performed by machines that can substitute each other, have to belong to the same subproblem, and (2) in the process graph there can not be two nodes of the same subproblem that are connected through a node of another subproblem. The investigation of the conditions of separability can be the basis of further research, including determining different problem classes classified by the applicable separation methods.
Another improvement can be resolving the constraint on the number of operation types that a resource can perform. In the presented heuristic algorithm, each resource can perform at most two different operation types. This constraint results from the applied simplifications, and its resolution requires significant modification of the proposed algorithm.
Instead of the applied CPLEX on the MILP model, another exact method can be used to solve the problem. For example, the creation of a CP model and its solution by the IBM ILOG CP Optimizer may have better performance in producing the problem's exact solution than CPLEX.
The impact of another combination type of the heuristic and the exact solver could also be analyzed: a solution found by the solver can be attempted to improve by the Improvement phase of the heuristic algorithm.

2. One of the future research directions related to the genetic algorithm of Chapter 3 could be the improvement of its efficiency on such FJSP problems that can be represented by a set of simple sequential graphs. It is possible that more efficient encoding - and thus genetic operators - could be found, taking into account the intention to maintain the representability of parallel execution of the different branches of the processes' and-subgraphs.
The presented GA can be modified by handling higher capacity values than 1 of the transportation devices. The GA's modification is easy to handle different capacity values; however, its impact on the optimality is more complex: there are cases when some operations should be delayed because batching them with later operations finally leads to better results.

3. The evident way to improve the fault-tolerant VRP-related work presented in Chapter 4 is to bring it closer to real circumstances: the applied constraints should be eliminated (these constraints were presented in section 4.4.1).
Releasing the assumption of equal and constant speed of the vehicles, the location calculation becomes more complex. However, with a precise forecast of the vehicle speeds, locations could be calculated with good quality.
If vehicles also differ in their capacity, there can be cases when the helper vehicle has to visit the broken-down vehicle more than once to service all the

remaining customers. These visits can be inserted into the route of the helper vehicle similar way as Step D in the algorithm of subsection 4.4.4 did.
Releasing the constraint that only one vehicle provides assistance, the routes' replanning at each assumed location of the breakdown becomes much more complex.
The problem approximates the reality better if the vehicle breakdown's assumed location can be anywhere along the route. The proposed algorithm can be modified easily to calculate each vehicle's location with more refined resolution along their route; however, with the precision, the computation need also increases.
When the customers' service time or the time of the repackaging at the broken-down vehicle is not negligible, it results in delay for the affected vehicles and influences the calculation of the vehicle locations.
If the vehicles do not start simultaneously from the depot, the location calculation must be modified by taking into account the different start times of the vehicles.
If the number of depots is more than one (thus, the problem becomes a Multi-Depot problem), the offered method has to be extended in the first step by assigning the customers to depots. After that, the proposed algorithm can be applied without changes for each depot.
All the other assumptions of section 4.4.1 are essential for the proposed algorithm: when time windows are applied, or the distances between points are not symmetrical, the inversion of the cyclic routes may not result in the same feasible route set. It means that even without a breakdown, the different route direction compositions may have a different sum of lengths (thus, different costs).

Appendices

Appendix A

Abbreviations

AJSP	Assembly Job-Shop Problem
AJSP	Job-Shop Scheduling with Processing Alternatives
CP	Constraint Programming
CVRP	Capacitated Vehicle Routing Problem
DVRP	Dynamic Vehicle Routing Problem
FIFO	First In First Out
FJSP	Flexible Job-Shop Problem
FSP	Flow-Shop Problem
GA	Genetic Algorithm
GLPK	GNU Linear Programming Kit
GRASP	Greedy Randomized Adaptive Search Procedure
HFS	Hybrid Flow-Shop Problem
ICT	Information and Communication Technology
IPODS	Integrated Production and Outbound Distribution Scheduling
JBX	Job-Based Crossover
JSP	Job-Shop Problem

LS	Local Search
MD-VRP	Multi-Depot Vehicle Routing Problem
MD-VRPI	Multi-Depot Vehicle Routing Problem with Inter-Depot Routes
MEMSY	Modular Expandable Multiprocessor SYstem
MILP	Mixed-Integer Linear Programming
MIP	Mixed-Integer Programming
MMJSP	Multi-Mode Job-Shop Problem
MPM-JSP	Job-Shop Problem with Multi-Purpose Machines
MS string	Machine Selection string
MULFLEX-SP	Multi-Resource Shop Scheduling with Resource Flexibility
NNH	Nearest Neighbour Heuristic
NP	Nondeterministic Polynomial Time
OR	Operations Research
OS string	Operation Sequence string
OSP	Open-Shop Problem
POX	Precedence Preserving Order-Based Crossover
PSN	Pocket Switched Network
PVRP	Periodic Vehicle Routing Problem
QoS	Quality of Service
SA	Simulated Annealing
SVRP	Stochastic Vehicle Routing Problem
TS	Tabu Search
TSP	Travelling Salesman Problem

UML	Unified Modeling Language
VRP	Vehicle Routing Problem
VRPM	Vehicle Routing Problem with Multiple Use of Vehicles
VRPPD	Vehicle Routing Problem with Pickup and Delivery
VRPRB	Vehicle Routing Problem with Route Balancing
VRP-TW	Vehicle Routing Problem with Time Windows

Appendix B

The formal description of the general production planning problem

The simple problem and the schedule in Figure B.1 are referred to during the introduction of the notations for better understanding.

Notations B.0.1. ***Input****-related notations:*

$M = \{m_1, ..., m_s\}$*: the set of the workflow types also called process models. In the example problem of Figure B.1, there are two process models:* m_1 *and* m_2*.*

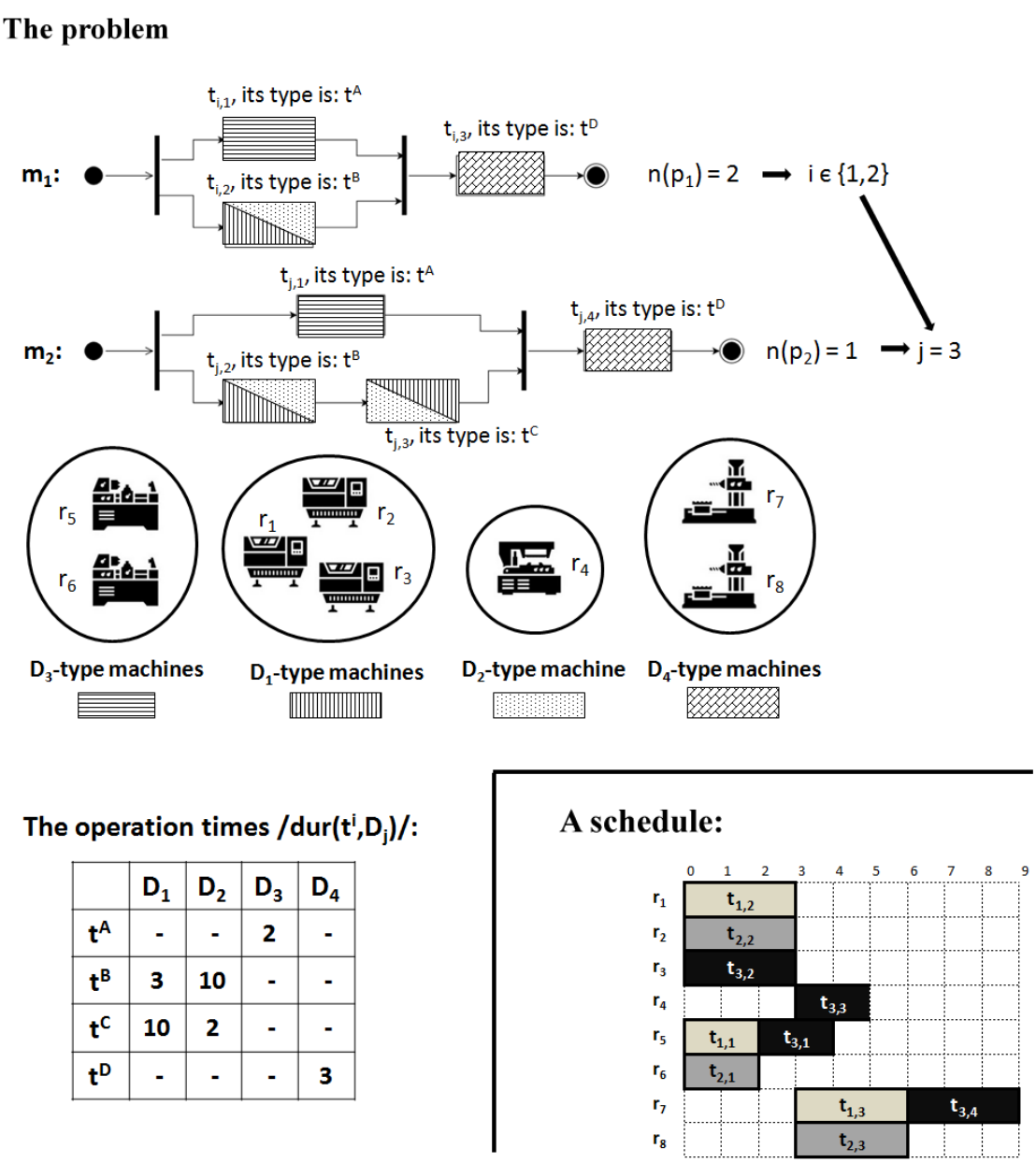
$W = \{w_1, ..., w_n\}$*: the set of the workflows also called processes. The problem of Figure B.1 has three workflows: two of them have workflow type* m_1*, and the third one has workflow type* m_2*.*

$T_i = \{t_{i,1}, ..., t_{i,m}\}$*: the set of the operations (also called tasks) of process* w_i*. The problem of Figure B.1 has* 10 *operations. The operations of process* w_1 *are* $t_{1,1}$*,* $t_{1,2}$*, and* $t_{1,3}$*, the operations of process* w_2 *are* $t_{2,1}$*,* $t_{2,2}$*, and* $t_{2,3}$*, finally, the operations of process* w_3 *are* $t_{3,1}$*,* $t_{3,2}$*,* $t_{3,3}$*, and* $t_{3,4}$*.*

$T = \cup_{i=1}^{n} T_i$*: the set of the operations of all processes. In the case of the problem of Figure B.1,* $T = \{t_{1,1}, t_{1,2}, t_{1,3}, t_{2,1}, t_{2,2}, t_{2,3}, t_{3,1}, t_{3,2}, t_{3,3}, t_{3,4}\}$*.*

$Pre(t_{i,j}) \subset T_i$*: the subset of the operations of process* w_i *(it can even be an empty set) whose elements are direct prior operations to operation* $t_{i,j}$ *in process* w_i*. This set may have more than one element because of the possible parallel structure of the processes. In the case of the problem of Figure B.1, e.g.,* $Pre(t_{3,2}) = Pre(t_{1,1}) = \emptyset$*,* $Pre(t_{3,3}) = \{t_{3,2}\}$*, and* $Pre(t_{3,4}) = \{t_{3,1}, t_{3,3}\}$*.*

$allPre(t_{i,j}) \subset T_i$*: the subset of the operations of process* w_i *(it can even be an empty set) whose elements are prior operations to operation* $t_{i,j}$ *in process* w_i*. E.g., if* $t_{i,b} \in Pre(t_{i,c})$ *and* $t_{i,a} \in Pre(t_{i,b})$*, then* $t_{i,a} \in allPre(t_{i,c})$*. In the case of the problem of Figure B.1,* $allPre(t_{3,1}) = allPre(t_{3,2}) = \emptyset$*,* $allPre(t_{3,3}) = \{t_{3,2}\}$*, and* $allPre(t_{3,4}) = \{t_{3,1}, t_{3,2}, t_{3,3}\}$*, for example.*



	D_1	D_2	D_3	D_4
t^A	-	-	2	-
t^B	3	10	-	-
t^C	10	2	-	-
t^D	-	-	-	3

Figure B.1. A simple scheduling problem and its schedule. They are applied during the explanation of the introduced notations.

$R = \{r_1, ..., r_u\}$: *the set of the resources. In Figure B.1, there are* 8 *resources:* $R = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8\}$.

$D = \{D_1, ..., D_y\}$: *the set of the resource types. In Figure B.1, there are* 4 *resource types:* $D = \{D_1, D_2, D_3, D_4\}$.

$n(D_l) \in \mathbb{Z}$: *the number of the resources of resource type* D_l. *As two examples,* $n(D_1) = 3$ *and* $n(D_2) = 1$ *in the problem of Figure B.1.*

$RT(r_k) \in D$: *the resource type of resource* r_k. *For example,* $RT(r_6) = D_3$ *in the problem of Figure B.1.*

$setup(D_l, t^S, t^U) \in \mathbb{N}$: *the setup time required to set a resource with resource type* D_l *from its state that can perform a* t^S*-type operation to a new state that makes the resource capable of performing a* t^U*-type operation.*

$A = \{t^A, t^B, ..., t^Z\}$: *the set of the operation types. In the problem of Figure B.1, there are* 4 *operation types:* $A = \{t^A, t^B, t^C, t^D\}$.

$TT(t_{i,j}) \in A$: *the operation type of operation* $t_{i,j}$. *For example,* $TT(t_{2,3}) = t^D$ *and* $TT(t_{3,3}) = t^C$ *in the problem of Figure B.1.*

$capable : D \times A \to \{0, 1\}$: *a function that describes whether a resource with a certain resource type can carry out a certain operation type. As two examples,* $capable(D_1, t^A) = 0$ *and* $capable(D_1, t^B) = 1$, *considering the problem of Figure B.1.*

$dur(t^S, D_l) \in \mathbb{N}$: *the duration of carrying out an operation with operation type* t^S *by a* D_l*-type resource. If* $capable(D_l, t^S) = 0$, *then* $dur(t^S, D_l) = \infty$. *For example, in the problem of Figure B.1,* $dur(t^B, D_2) = 10$ *and* $dur(t^D, D_2) = \infty$.

$durRe(t^S, D_l) \in \mathbb{N}$: *the duration; after that, a resource with resource type* D_l *can be used again for performing a new operation after starting a* t^S*-type operation. If* $capable(D_l, t^S) = 0$, *then* $durRe(t^S, D_l) = \infty$. *This notion is introduced because of the existence of transportation devices. It is the sum of the operation time and a specialized setup time that follows the operation. Because there is no transportation resource in the problem of Figure B.1, in the example, the values of function* $durRe()$ *are equal to the values of function* $dur()$ *for each operation type - resource type pair.*

$P = \{p_1, ..., p_s\}$: *the set of the product types. In the problem of Figure B.1, there are* 2 *product types:* p_1 *and* p_2.

$n(p_l) \in \mathbb{Z}$: *the number of the products with product type* p_l *that has to be produced. As it can be seen in Figure B.1,* $n(p_1) = 2$ *and* $n(p_2) = 1$ *in the example.*

$I = \{I_1, ..., I_n\}$: *the set of the product items. As in the example of Figure B.1, there are two product items (*I_1 *and* I_2*) with product type* p_1 *and one product item (*I_3*) with product type* p_2.

$PT(I_v) \in P$: *the product type of product item* I_v. *For example,* $PT(I_3) = p_2$ *in the example of Figure B.1.*

$WF(I_v) \in W$: *the workflow that produces product* I_v. *In the problem of Figure B.1,* $WF(I_1) = w_1$, $WF(I_2) = w_2$, *and* $WF(I_3) = w_3$.

$WM(w_v) \in M$: *the workflow model of* w_v *workflow. In the problem of Figure B.1,* $WM(w_1) = WM(w_2) = m_1$, *and* $WM(w_3) = m_2$.

$WM(p_l) \in M$: *the workflow model that produces products with product type* p_l. *In the problem of Figure B.1,* $WM(p_1) = m_1$, *and* $WM(p_2) = m_2$.

Notations B.0.2. ***Schedule****-related variables:*

$X(t_{i,j}) \in R$: *the resource which is assigned to operation* $t_{i,j}$ *by the scheduler. For example,* $X(t_{1,2}) = r_1$ *in the schedule of Figure B.1.*

$start(t_{i,j}) \in \mathbb{N}$: *the start time of operation* $t_{i,j}$ *of process* w_i. *As an example,* $start(t_{3,3}) = 3$ *in Figure B.1.*

$end(t_{i,j}) \in \mathbb{N}$: *the finish time of operation* $t_{i,j}$ *of process* w_i. *As an example,* $end(t_{1,3}) = 6$, *based on Figure B.1.*

$makespan(W) = \max(end(t_{i,j})) - \min(start(t_{k,l}))$ *for all* $t_{i,j}, t_{k,l} \in T$. *In the example of Figure B.1* $makespan(W) = end(t_{3,4}) - start(t_{2,1}) = 9 - 0 = 9$.

The **input-related constraints** are:

$$WF(I_v) \equiv WF(I_q) \Leftrightarrow v = q, \forall\ 1 \leq v, q \leq n.$$

This constraint ensures that each product item has its own, separate workflow that the item's production is based on.

$$WM(p_l) \equiv WM(p_k) \Leftrightarrow l = k, \forall\ 1 \leq l, k \leq s.$$

This constraint ensures that each product type has its own, separate workflow model that the production of the products with this product type is based on.

$$n(p_l) = |\bigcup_{i=1}^{n} I_i,\ where\ PT(I_i) = p_l|, \forall\ 1 \leq l \leq s.$$

The above constraint defines the calculation of the cardinality of the set of products with a certain product type.

Based on the above 3 constraints, the following constraint can be gained:

$$\sum_{l=1}^{s} n(p_l) = |W|.$$

The constraint prescribes that each workflow produces exactly one product.

$$For\ \forall\ r_k, 1 \leq k \leq u,\ \exists\ D_l, 1 \leq l \leq y,\ such\ that\ RT(r_k) = D_l, and$$
$$n(D_l) = |\bigcup_{i=1}^{u} r_i,\ where\ RT(r_i) = D_l|, \forall\ 1 \leq l \leq y.$$

The above constraint defines the calculation of the number of resources with a certain resource type.

$$dur(t^S, D_l) \leq durRe(t^S, D_l),\ \forall\ t^S \in A,\ D_l \in D.$$

This constraint prescribes that a resource can not be ready to perform any next operation before its ongoing operation is finished. It is worth noting that practically the differentiation between functions $dur()$ and $durRe()$ is required because of the possible existing of transportation machines: after a transportation device transports a product to the destination, ensuring that the product is ready for the following operation, the machine still requires time to get back to its initial location for starting a new similar transportation operation.

The constraints for determining $start(t_{i,j})$ and $end(t_{i,j})$ for all $t_{i,j} \in T$ are:

$$X(t_{i,j}) \neq \emptyset, \forall\ t_{i,j} \in T$$

This constraint means that each operation has to be carried out by a resource.

$$end(t_{i,j}) = start(t_{i,j}) + dur(TT(t_{i,j}), RT(X(t_{i,j}))),\ \forall\ t_{i,j} \in T$$

The above constraint represents the connection between the start and the finish time of an operation regarding the related operation time.

$$\begin{aligned} start(t_{l,m}) \geq\ & start(t_{i,j}) + durRe(TT(t_{i,j}), RT(X(t_{i,j}))) \\ & + setup(X(t_{i,j}), TT(t_{i,j}), TT(t_{l,m})) \quad or \\ & start(t_{l,m}) + durRe(TT(t_{l,m}), RT(X(t_{l,m}))) \\ & + setup(X(t_{l,m}), TT(t_{l,m}), TT(t_{i,j})) \leq start(t_{i,j}), \end{aligned}$$

$$\forall\ t_{i,j}, t_{l,m} \in T, \quad if \quad t_{i,j} \neq t_{l,m} \quad and \quad X(t_{i,j}) \equiv X(t_{l,m}))$$

This constraint specifies that the operations allocated to the same resource must not overlap each other.

$$end(t_{i,a}) \leq start(t_{i,b}),\ \forall\ 1 \leq i \leq n,\ \forall\ t_{i,a}, t_{i,b} \in T_i,\ if\ t_{i,a} \in allPre(t_{i,b})$$

The final constraint describes that an operation of a process must not start before its prior operations of the considered operation's process are not finished.

For the problem with the process set W, the solution with minimal $makespan(W)$ is intended to be found.

Appendix C

The specific model of Chapter 2

Here the problem of Chapter 2 is described formally, using the notations of Appendix B.

As Figure 2.2 of subsection 2.2 shows, there are 2 workflow types in the considered problem: $M = \{m_1, m_2\}$. Process model m_1 is executed $n(p_1)$ times, and process model m_2 is executed $n(p_2)$ times. It means that $|W| = n(p_1) + n(p_2)$. $WM(w_v) = m_1,\ if\ 1 \leq v \leq n(p_1)$, and $WM(w_v) = m_2,\ if\ n(p_1) + 1 \leq v \leq n(p_1) + n(p_2)$. The structure of the process models - highlighting the workflows' operations and based on Figure 2.2 - are illustrated in Figure C.1. The reason for using non-consecutive indexes for the operations is that the same indexes are used for the same operations in the extended workflows of Chapter 3.

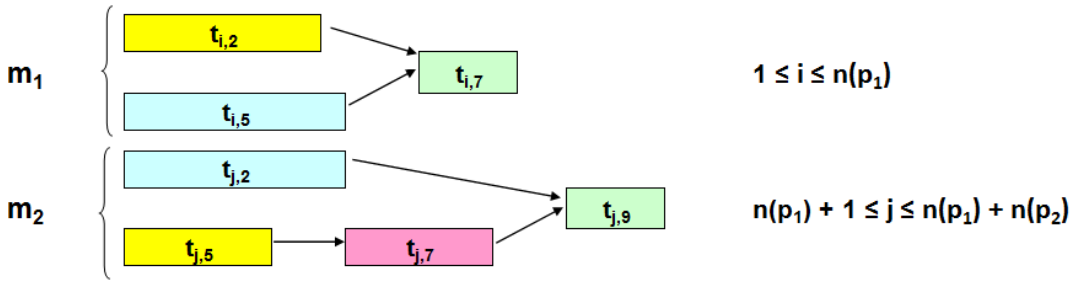


Figure C.1. The process models of Chapter 2.

The set of the operations of workflow w_i - that has workflow type m_1 - is: $T_i = \{t_{i,2}, t_{i,5}, t_{i,7}\}, \forall i$ where $WM(w_i) = m_1$.

The structure of workflow w_i - that follows process model m_1 - is determined by the sequence of its operations as follows:

$$\left.\begin{aligned} Pre(t_{i,2}) = Pre(t_{i,5}) = \emptyset, \\ Pre(t_{i,7}) = \{t_{i,2}, t_{i,5}\}, \end{aligned}\right\} \forall i \text{ where } WM(w_i) = m_1.$$

The set of the operations of workflow w_j - that has workflow type m_2 - is: $T_j = \{t_{j,2}, t_{j,5}, t_{j,7}, t_{j,9}\}, \forall j$ where $WM(w_j) = m_2$.

The structure of workflow w_j - that follows process model m_2 - is determined by the sequence of its operations as follows:

$$\left.\begin{aligned} Pre(t_{j,2}) = Pre(t_{j,5}) &= \emptyset, \\ Pre(t_{j,7}) &= \{t_{j,5}\}, \\ Pre(t_{j,9}) &= \{t_{j,2}, t_{j,7}\}, \end{aligned}\right\} \forall j \text{ where } WM(w_j) = m_2.$$

The types of operations are:

$$\left.\begin{aligned} TT(t_{i,5}) = TT(t_{j,2}) &= t^A, \\ TT(t_{i,2}) = TT(t_{j,5}) &= t^B, \\ TT(t_{j,7}) &= t^C, \\ TT(t_{i,7}) = TT(t_{j,9}) &= t^D, \end{aligned}\right\} \forall i \text{ where } WM(w_i) = m_1 \text{ and } \forall j \text{ where } WM(w_j) = m_2.$$

Based on Figure 2.2, the resources of the problem are:
$R = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8\}$.

The types of these resources are:
$RT(r_1) = RT(r_2) = RT(r_3) = D_1$,
$RT(r_4) = D_2$,
$RT(r_5) = RT(r_6) = D_3$,
$RT(r_7) = RT(r_8) = D_4$.

It means that the numbers of the resources with the same resource type are: $n(D_1) = 3$, $n(D_2) = 1$, $n(D_3) = 2$ and $n(D_4) = 2$.

The resource-related data are summarized in Table C.1.

Table C.1. Resource types and their properties: operation time for the resource type - operation type pairs.

	t^A	t^B	t^C	t^D
D_1	∞	$dur(t^B, D_1)$	$dur(t^C, D_1)$	∞
D_2	∞	$dur(t^B, D_2)$	$dur(t^C, D_2)$	∞
D_3	$dur(t^A, D_3)$	∞	∞	∞
D_4	∞	∞	∞	$dur(t^D, D_4)$
t^A, t^B, t^C, t^D are operation types				
D_1, D_2, D_3, D_4 are resource types				

Appendix D

Demonstration: preemptive solution of a small problem

Let us consider a problem whose parameters are determined as follows:

- $n(p_1) = 7$,
- $n(p_2) = 13$.

It means that 20 is the total number of products that have to be produced. Each of them requires the execution of a t^B-type operation, but a t^C-type operation has to be carried out only for 13 pieces of them.
The cardinality of the sets of different machines are:

- $\alpha = 1$ (the number of the D_1-type machines that make only t^B-type operations),
- $\beta = 2$ (the number of the D_1-type machines that make both t^B-type operations and t^C-type operations),
- $\gamma = 1$ (the number of the D_2-type machines that carry out only t^C-type operations).

The time-related parameters are:

- $setup(D_1, t^B, t^C) = 2$ (the setup time),
- $dur(t^B, D_1) = 7$ (the processing time of a t^B-type task on a D_1-type machine),
- $dur(t^C, D_1) = 14$ (the processing time of a t^C-type task on a D_1-type machine),
- $dur(t^C, D_2) = 8$ (the processing time of a t^C-type task on a D_2-type machine).

After solving the equation system (2.1)-(2.3), the result is:

- $x = 305/38 \approx 8.026$,
- $y = 4059/399 \approx 10.173$,
- $v = 3921/798 \approx 4.914$,
- and $w = 189/76 \approx 2.487$.

Appendix E

Demonstration: rounding the small problem of Appendix D

After truncating the variables of the preemptive solution, the obtained numbers are:

- $x = 8$, so, the number of the t^C-type operations on the D_2-type machine is 8,
- $y = 10$, it means that the one D_1-type machine that carries out only t^B-type operations has 10 pieces of t^B-type operations,
- $v = 4$, so the number of the t^B-type operations on the two D_2-type machines that perform both t^B-type operations and t^C-type operations is 4 per machine,
- and $w = 2$, it means that the number of the t^C-type operations on the two D_2-type machines that perform both t^B-type operations and t^C-type operations is 2 per machine.

The job-assignment after the truncation is illustrated in Figure E.1. Worth noting that there are fewer operations assigned to the machines than the process executions require.

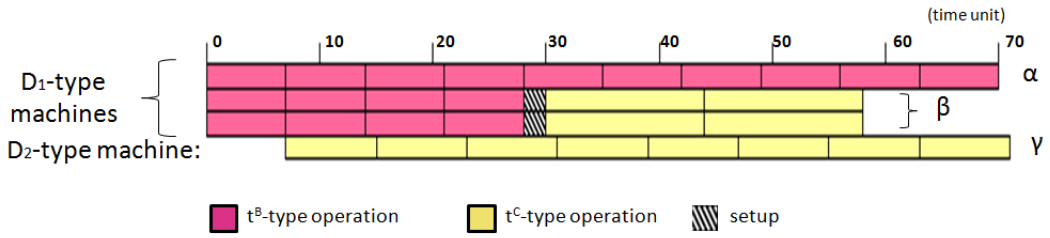


Figure E.1. Job assignment after the truncation phase of the rounding procedure for the introduced small problem.

After counting the assigned t^B-type operations, their number is obtained as $10 + 2 \cdot 4 = 18$. Since $n(p_1) = 7$ and $n(p_2) = 13$, the total number of the t^B-type operations of the processes is 20. It means that the first D_1-type machine has

11 t^B-type operations (instead of the previously assigned 10 tasks), the second D_1-type machine (that performs both t^B-type operations and t^C-type operations) gets 5 t^B-type operations (instead of the previously assigned 4 t^B-type tasks), and the 4 pieces of the t^B-type operations of the third D_1-type machine (that also performs both t^B-type operations and t^C-type operations) remains unchanged.
Similarly, the number of the assigned t^C-type operations is $2 \cdot 2 + 8 = 12$. Since $n(p_2) = 13$, the number of the t^C-type operations on the D_2-type machine is increased from 8 to 9, and the number of the t^C-type operations on the two D_2-type machines (they perform both t^B-type operations and t^C-type operations) remains 2 per machine.

The rounding procedure results in the job-assignment that is shown in Figure E.2.

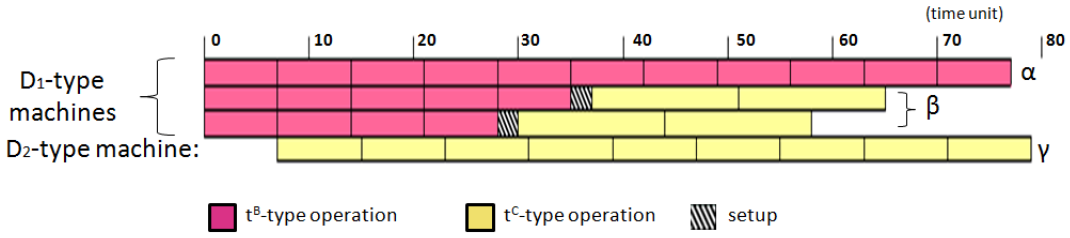


Figure E.2. Job assignment after the rounding procedure for the introduced small problem.

Appendix F

Critical path, block, and neighbourhood in Tabu Search, based on Hurink et al. [1]

In the paper of Hurink et al., a tabu search algorithm is presented, where the neighbours of a solution are generated by repositioning operations of some blocks of the critical path.

Definition 78. *Critical path A critical path of the graph is the shortest path from the earliest operation to the latest operation. Let us consider the directed graph model of a schedule: $G = (V, E)$, where V is the set of operations and $E = \{(v_i, v_j), v_i \in V$ and $v_j \in V$ where v_j is the direct subsequent operation of v_i in their common process, or in the queue of their common machine$\}$ is the set of directed arcs. Let each $e_{i,j} = (v_i, v_j)$ arc be weighted by the difference between the start time of the operation v_j and the finish time of operation v_i. Starting from the operation which has the latest finish time; all the paths are tracked backward in the graph and collected in a path-set as follows:*

- *if the following arc (during the backward tracking) of a path passes from a machine to another machine and the weight of the arc is not equal to zero, then the path is deleted from the path set,*
- *if the following arc (during the backward tracking) of a path remains on the same machine and the arc's weight is not equal to the appropriate setup time, then the path is deleted from the path set.*

Finally, the paths in the path-set are the critical paths.

Definition 79. *Block of a critical path A sequence of successive nodes if (1) all of the nodes of this sequence are on the same machine, (2) the length of this sequence is at least two, and (3) the sequence can not be enlarged by one operation without conflicting the first property.*

During the neighbour-generation process, a random operation from a block of a critical path of the iteration's initial schedule is selected and moved to a new place, following the next strategy:

- either another machine is selected randomly that can execute the operation, and the selected operation is inserted into the operation-queue of this machine into a feasible position,
- or the operation (if it is not the first operation of the block) is moved just before all other operations of the block,
- or the operation (if it is not the last operation of the block) is moved just after all other operations of the block.

The algorithm maintains a fix-sized tabu list that is empty in the beginning and follows FIFO manner. In each iteration, the best neighbour that is not an element of the tabu list is selected as the solution of the iteration. After getting the solution of an iteration, this solution is pushed into the tabu list. The tabu list avoids selecting the same solution in the near future, thus avoids getting into a cycle. The Tabu Search algorithm performs several iterations and returns the found best solution as its output.

Appendix G

The Tabu Search algorithm of Chapter 2

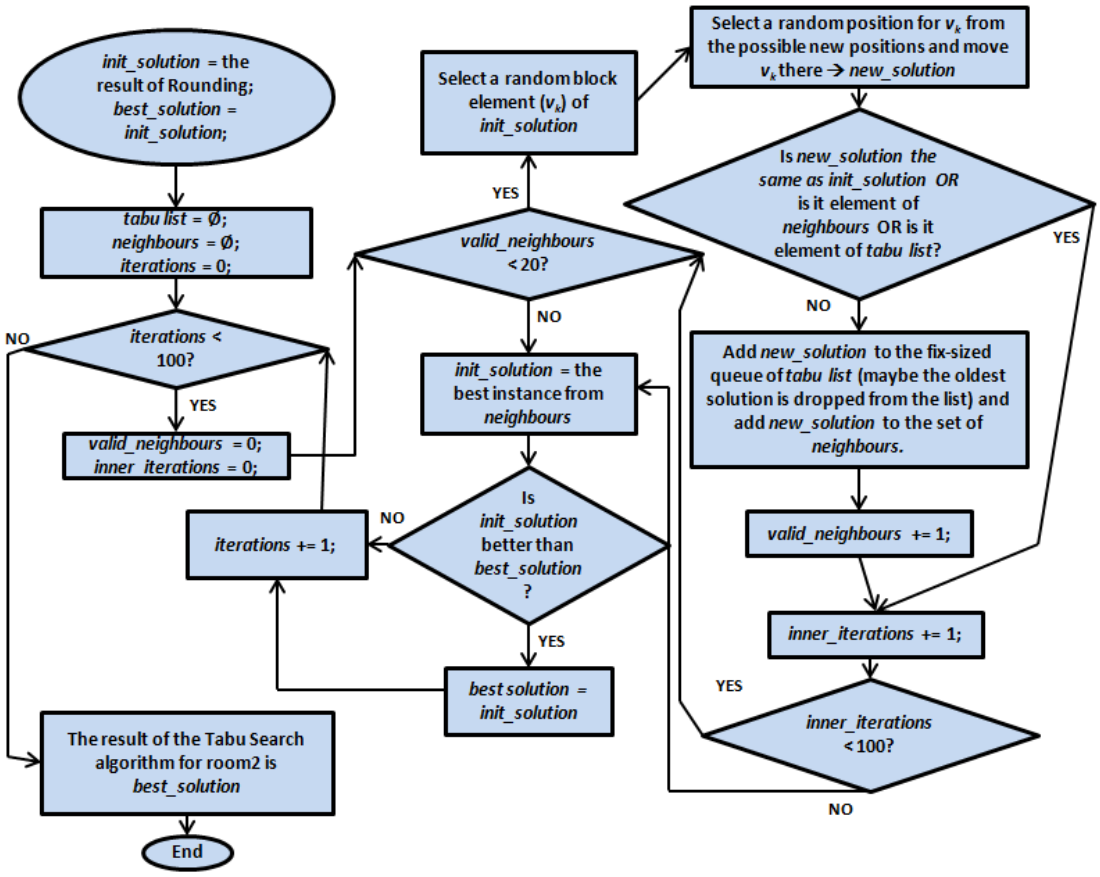


Figure G.1. The implemented Tabu Search algorithm.

Appendix H

The MILP model of the introduced problem

H.1 Sets and Parameters

$\mathcal{T} = \{0, 1, \ldots, T\}$ denotes the set of time steps. $\mathcal{T}' = \mathcal{T} \backslash \{T\}$ and $\mathcal{T}^- = \mathcal{T} \backslash \{0\}$.

Table H.1 shows the time-related parameters (operation time and setup time) of the problem.

Further parameters describe the amount of raw material for the t^A-type operations and the t^B-type operations and the number of desired final products, see Table H.2.

H.2 Variables

The decision variables used in this model are all binary or integer. They are divided into two categories: stock variables and flow variables. Considering the production as a directed graph, the stock variables s are the nodes. They describe the amount of material available for production. The resources (machines) are the arcs, which correspond to the flow variables f. Both variables are time-indexed since they describe a dynamic setting that changes from time point to time point. The final class of variables represents certain time points in the schedule, such as the end of the production. The details can be found in Tables H.3, H.4, and H.5.

H.3 Constraints

The constraints related to the operation execution are described in detail in the following subsections.

H.3.1 t^A-type operations

The t^A-type operations are executed by resources r_5 and r_6, using the raw material of t^A-type operations.

name	index i	description /$dur(t^X, D_i)$ denotes the operation time of a t^X-type operation with a D_i-type resource, following the notations introduced in Appendix B./
$p_{r_i}^{t^A}$	5&6	Shows the processing time of the r_i resource for each t^A-type operation. It is equal to $dur(t^A, D_3)$.
$p_{r_i}^{t^B}$	1–4	Shows the processing time of the r_i resource for executing a t^B-type operation. It is equal to $dur(t^B, D_1)$ if $i \in \{1, 2, 3\}$, and it is equal to $dur(t^B, D_2)$ if $i = 4$.
$p_{r_i}^{t^C}$	1–4	Shows the processing time of the r_i resource for executing a t^C-type operation. It is equal to $dur(t^C, D_2)$ if $i = 4$, and it is equal to $dur(t^C, D_1)$ if $i \in \{1, 2, 3\}$.
$p_{r_i}^{t^D}$	7&8	Shows the processing time of r_i for each t^D-type operation. It is equal to $dur(t^D, D_4)$.
u_{r_i}	1–4	Shows the setup time for r_i for changing between a t^B-type operation and a t^C-type operation. It is equal to $setup(D_1, t^B, t^C)$ if $i \in \{1, 2, 3\}$, and it is equal to $setup(D_2, t^B, t^C)$ if $i = 4$.

Table H.1. Time-related parameters of the model.

name	description
m^{t^A}	raw material input for the t^A-type operations
m^{t^B}	raw material input for the t^B-type operations
$n(p_1)$	the desired number of products of the first type (p_1)
$n(p_2)$	the desired number of products of the second type (p_2)

Table H.2. Production input and output parameters.

name	range	description
$s^{t^A raw}(t)$	$\mathbb{Z}_+$	raw material for t^A-type operation at time t
$s^{t^A finished}(t)$	$\mathbb{Z}_+$	finished products of a t^A-type operation at time t
$s^{t^B raw}(t)$	$\mathbb{Z}_+$	raw material for t^B-type operation at time t
$s^{t^B finished}(t)$	$\mathbb{Z}_+$	finished products of a t^B-type operation at time t
$s^{t^C finished}(t)$	$\mathbb{Z}_+$	finished products of a t^C-type operation at time t
$s^{p_1}(t)$	$\mathbb{Z}_+$	finished first type (p_1) products at time t
$s^{p_2}(t)$	$\mathbb{Z}_+$	finished second type (p_2) products at time t

Table H.3. Stock variables (associated with nodes in the production graph).

name	i	range	description
$f_{r_i}^{t^A}(t)$	5&6	$\{0,1\}$	r_i resource starts processing at time t
$f_{r_i}^{t^B}(t)$	1-4	$\{0,1\}$	r_i resource starts a t^B-type operation at time t
$f_{r_i}^{t^C}(t)$	1-4	$\{0,1\}$	r_i resource starts a t^C-type operation at time t
$f_{r_i}^{p_1}(t)$	7&8	$\{0,1\}$	r_i resource starts finishing a first type (p_1) product at time t
$f_{r_i}^{p_2}(t)$	7&8	$\{0,1\}$	r_i resource starts finishing a second type (p_2) product at time t
$f_{r_i}^{t^D}(t)$	7&8	$\{0,1\}$	r_i resource starts finishing a t^D-type operation (an assembly task) at time t

Table H.4. Flow variables (associated with arcs in the production graph).

name	i	range	description
C_{max}		$\mathbb{Z}_+$	makespan
ℓ_{r_i}	7&8	$\mathbb{Z}_+$	time point of the end of manufacturing on r_i resource

Table H.5. Time variables.

The initial conditions are:

$$s^{t^A raw}(0) = m^{t^A},$$

$$s^{t^A finished}(0) = 0,$$

which place all the raw material of t^A-type operations in front of r_5 and r_6 and assumes an empty stock of finished products of the t^A-type operations at time point 0. The boundary conditions enforce no production in the very last time point T:

$$f_{r_5}^{t^A}(T) = f_{r_6}^{t^A}(T) = 0.$$

There are two nodes associated with the execution of t^A-type operations; both of them give rise to flow conservation constraints. The raw material of t^A-type operations of time point t is moved to the two resources r_5, r_6, and the remaining raw material is moved to the next time point:

$$s^{t^A raw}(t) = s^{t^A raw}(t+1) + f_{r_5}^{t^A}(t) + f_{r_6}^{t^A}(t), \quad \forall\, t \in \mathcal{T}'.$$

The flow conservation at the node at the other end of r_5, r_6 takes the finished materials and either stores them as a stock or moves them forward for the final assembly:

$$s^{t^A finished}(t-1) + f_{r_5}^{t^A}(t-p_{r_5}^{t^A}) + f_{r_6}^{t^A}(t-p_{r_6}^{t^A}) = s^{t^A finished}(t) + f_{r_7}^{t^D}(t) + f_{r_8}^{t^D}(t), \quad \forall\, t \in \mathcal{T}^-.$$

Note that here and in all the following equations, if the time index of a variable is not in $\mathcal{T}$, the variable is replaced by the value 0. In the above equation, it means that $f_{r_5}^{t^A}(t - p_{r_5}^{t^A})$ is replaced by 0, whenever $t - p_{r_5}^{t^A} \notin \mathcal{T}$.

The production within r_5, r_6 takes a certain number of time steps, and only one item (raw material) can be processed within that time span:

$$\sum_{\Delta \in \{0,\ldots,p_{r_i}^{t^A}-1\}} f_{r_i}^{t^A}(t-\Delta) \leq 1, \quad \forall t \in \mathcal{T}, i = 5, 6.$$

H.3.2 t^B- and t^C-type operations

First, a t^B-type operation is executed on the initial raw material by one of the resources $r_1, \ldots, r_4$. Then, the t^B-type operation's resulted products are either moved for final assembly, or they are further processed, and a t^C-type operation is executed on them by one of the resources $r_1, \ldots, r_4$. As an initial condition, all the initial raw material is placed in front of machines $r_1, \ldots, r_4$ at time point 0:

$$s^{t^B raw}(0) = m^{t^B}.$$

There is no possibility for executing a t^C-type operation at time point 0:

$$f_{r_1}^{t^C}(0) = \ldots = f_{r_4}^{t^C}(0) = 0.$$

There is an empty stock of products of executed t^B-type and t^C-type operations at time point 0:

$$s^{t^B finished}(0) = s^{t^C finished}(0) = 0.$$

As boundary conditions, it is required that no processing takes place in the very last time point T:

$$f_{r_1}^{t^B}(T) = \ldots = f_{r_4}^{t^B}(T) = f_{r_1}^{t^C}(T) = \ldots = f_{r_4}^{t^C}(T) = 0.$$

The raw material of a t^B-type operation is either processed by one of the machines $r_1, \ldots, r_4$, or it is moved to the next time point as raw material:

$$s^{t^B raw}(t) = s^{t^B raw}(t+1) + f_{r_1}^{t^B}(t) + \ldots + f_{r_4}^{t^B}(t), \quad \forall t \in \mathcal{T}'.$$

The products of finished t^B-type operations are either stored or processed further by a t^C-type operation on one of the machines $r_1, \ldots, r_4$, or moved for final assembly to the machines r_7, r_8:

$$\begin{aligned} &s^{t^B finished}(t-1) + f_{r_1}^{t^B}(t - p_{r_1}^{t^B}) + \ldots + f_{r_4}^{t^B}(t - p_{r_4}^{t^B}) \\ &= s^{t^B finished}(t) + f_{r_1}^{t^C}(t) + \ldots + f_{r_4}^{t^C}(t) + f_{r_7}^{p_1}(t) + f_{r_8}^{p_1}(t), \quad \forall t \in \mathcal{T}^-. \end{aligned}$$

The products of finished t^C-type operations are either stored or moved for final assembly to the machines r_7, r_8:

$$\begin{aligned} &s^{t^C finished}(t-1) + f_{r_1}^{t^C}(t - p_{r_1}^{t^C}) + \ldots + f_{r_4}^{t^C}(t - p_{r_4}^{t^C}) \\ &= s^{t^C finished}(t) + f_{r_7}^{p_2}(t) + f_{r_8}^{p_2}(t), \quad \forall t \in \mathcal{T}^-. \end{aligned}$$

Let it be either a t^B-type operation or a t^C-type operation within an $r_1, \ldots, r_4$ resource, its execution takes a certain number of time steps. Moreover, only one

item (in case of a t^B-type operation, a raw material of a t^B-type operation; in case of a t^C-type operation, a product of a prior t^B-type operation) can be processed within that time span:

$$\sum_{\Delta \in \{0,\dots,p_{r_i}^{t^X}-1\}} f_{r_i}^{t^X}(t-\Delta) \leq 1, \quad \forall t \in \mathcal{T}, i = 1,\dots,4, \quad \forall t^X \in \{t^B, t^C\}.$$

When changing either from a t^B-type operation to a t^C-type operation or from a t^C-type operation to a t^B-type operation, the setup time must be obeyed:

$$f_{r_i}^{t^X}(t) + f_{r_i}^{t^Y}(t+\Delta) \leq 1, \quad \forall \Delta \in \{0,\dots,p_{r_i}^{t^X} + u_{r_i} - 1\}, i = 1,\dots,4,$$
$$\forall t^X, t^Y \in \{t^B, t^C\}, \quad t^X \neq t^Y.$$

H.3.3 t^D-type operations

Two resources (r_7 and r_8) assemble the products in the final step, resulting in products of product types p_1 and p_2.

As initial conditions, it is assumed that the stock of final products of both p_1 and p_2 product types is empty:

$$s^{p_1}(0) = s^{p_2}(0) = 0.$$

The flow of the products of finished t^B-type, t^C-type, and t^A-type operations through resources r_7, r_8 is initially empty; moreover, there is no flow in the very last time point, too:

$$f_{r_i}^{p_1}(t) = f_{r_i}^{p_2}(t) = f_{r_i}^{t^D}(t) = 0, \quad i = 7,8, \quad \forall t \in \{0, T\}.$$

The final stocks of the products of product type p_1 and the products of product type p_2 must meet the required demand:

$$s^{p_j}(T) = n(p_j), \quad \forall p_j \in \{p_1, p_2\}.$$

Both the stock of p_1-type products and the stock of p_2-type products are increased by adding finished products coming out of r_7, r_8 resources:

$$s^{p_j}(t-1) + f_{r_7}^{p_j}(t - p_{r_7}^{t^D}) + f_{r_8}^{p_j}(t - p_{r_8}^{t^D}) = s^{p_j}(t), \quad \forall p_j \in \{p_1, p_2\}, \quad \forall t \in \mathcal{T}^-.$$

Only one product can be handled during the processing time while performing either a t^A-type operation or a t^B-type operation or a t^C-type operation by resource r_7 or r_8:

$$\sum_{\Delta \in \{0,\dots,p_{r_i}^{t^D}-1\}} f_{r_i}^{Y}(t-\Delta) \leq 1, \quad \forall t \in \mathcal{T}, \quad \forall Y \in \{t^D, p_1, p_2\}, \quad i = 7,8.$$

A product consists of an outcome of a t^A-type operation assembled with either an outcome of a t^C-type operation (on which a t^B-type operation was carried out

earlier) or an outcome of a t^B-type operation. The two parts have to be assembled at the same time:

$$f_{r_i}^{t^D}(t) = f_{r_i}^{p_2}(t) + f_{r_i}^{p_1}(t), \quad \forall\, t \in \mathcal{T}, i = 7, 8.$$

H.4 Makespan and Objective

The makespan is the time when the last product is completed on either an r_7 or an r_8 resource:

$$(t + p_{r_i}^{t^D}) \cdot f_{r_i}^{t^D}(t) \leq \ell_{r_i}, \quad \forall\, t \in \mathcal{T}, i = 7, 8,$$

and

$$\ell_{r_i} \leq C_{max}, \quad \forall\, i = 7, 8.$$

The objective is to minimize the total makespan C_{max}.

Appendix I

The calculation of the coarse upper and lower bounds for subsection 2.4.1

The calculation of the coarse upper bound ($\mathcal{T}^{\mathrm{ub}}$) that is applied in Method1 - using the notations of Appendix H - is:

$$\begin{aligned}\mathcal{T}^{\mathrm{ub}} := \; & \lceil m^{t^A} \cdot ((p_{r_5}^{t^A})^{-1} + (p_{r_6}^{t^A})^{-1})^{-1} \rceil + \\ & \lceil m^{t^B} \cdot ((p_{r_1}^{t^B})^{-1} + \ldots + (p_{r_4}^{t^B})^{-1})^{-1} \rceil + \\ & \quad \max\{u_{r_1}, \ldots, u_{r_4}\} + \\ & \quad \lceil n(p_2) \cdot ((p_{r_1}^{t^C})^{-1} + \ldots + (p_{r_4}^{t^C})^{-1})^{-1} \rceil + \\ & \lceil m^{t^A} \cdot ((p_{r_7}^{t^D})^{-1} + (p_{r_8}^{t^D})^{-1})^{-1} \rceil.\end{aligned}$$

The calculation of the coarse lower bound ($\mathcal{T}^{\mathrm{lb}}$) that is applied in Method2 - using the notations of Appendix H - is:

$$\begin{aligned}\mathcal{T}^{\mathrm{lb}} := \; & \min\{\lfloor m^{t^A} \cdot ((p_{r_5}^{t^A})^{-1} + (p_{r_6}^{t^A})^{-1})^{-1} \rfloor, \\ & \quad \lfloor m^{t^B} \cdot ((p_{r_1}^{t^B})^{-1} + \ldots + (p_{r_4}^{t^B})^{-1})^{-1} \rfloor + \\ & \quad \quad \lfloor n(p_2) \cdot ((p_{r_1}^{t^C})^{-1} + \ldots + (p_{r_4}^{t^C})^{-1})^{-1} \rfloor, \\ & \quad \lfloor m^{t^A} \cdot ((p_{r_7}^{t^D})^{-1} + (p_{r_8}^{t^D})^{-1})^{-1} \rfloor\}.\end{aligned}$$

Appendix J

The specific model of Chapter 3

Here the problem of Chapter 3 is described formally, using the notations of Appendix B.

As Figure 3.1 of subsection 3.1 shows, there are 2 workflow types in the considered problem: $M = \{m_1, m_2\}$. These are extended variants of the workflow types of Chapter 2. Process model m_1 is executed $n(p_1)$ times, and process model m_2 is executed $n(p_2)$ times. It means that $|W| = n(p_1) + n(p_2)$. $WM(w_v) = m_1,\ if\ 1 \leq v \leq n(p_1)$, and $WM(w_v) = m_2,\ if\ n(p_1) + 1 \leq v \leq n(p_1) + n(p_2)$. Moreover, its consequence is that $P = \{p_1, p_2\}$, so, there are 2 types of products ($s = 2$), where $WM(p_1) = m_1$ and $WM(p_2) = m_2$.
The structure of the process models - highlighting the workflows' operations and based on Figure 3.1 - is illustrated in Figure J.1.

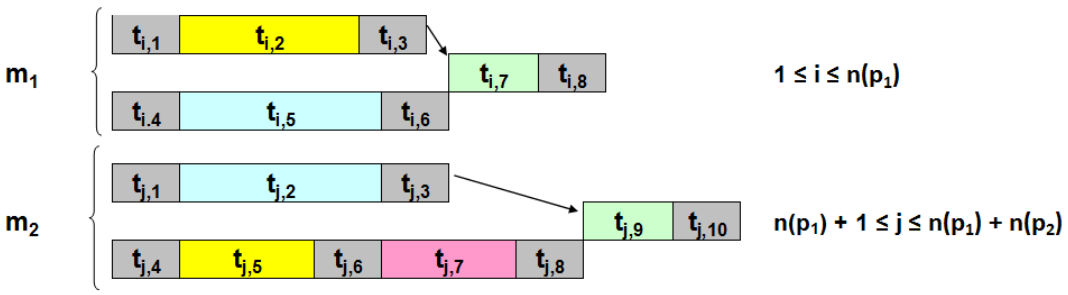


Figure J.1. The process models of Chapter 3.

The set of the operations of workflow w_i - that has workflow type m_1 - is: $T_i = \{t_{i,1}, t_{i,2}, t_{i,3}, t_{i,4}, t_{i,5}, t_{i,6}, t_{i,7}, t_{i,8}\}, \forall i$ where $WM(w_i) = m_1$.

The structure of workflow w_i - that follows process model m_1 - is determined by the sequence of its operations as follows:

$$\left.\begin{aligned} Pre(t_{i,1}) = Pre(t_{i,4}) = \emptyset, \\ Pre(t_{i,2}) = \{t_{i,1}\}, \\ Pre(t_{i,3}) = \{t_{i,2}\}, \\ Pre(t_{i,5}) = \{t_{i,4}\}, \\ Pre(t_{i,6}) = \{t_{i,5}\}, \\ Pre(t_{i,7}) = \{t_{i,3}, t_{i,6}\}, \\ Pre(t_{i,8}) = \{t_{i,7}\}, \end{aligned}\right\} \forall i \text{ where } WM(w_i) = m_1.$$

The set of the operations of workflow w_j - that has workflow type m_2 - is: $T_j = \{t_{j,1}, t_{j,2}, t_{j,3}, t_{j,4}, t_{j,5}, t_{j,6}, t_{j,7}, t_{j,8}, t_{j,9}, t_{j,10}\}, \forall j$ where $WM(w_j) = m_2$.

The structure of workflow w_j - that follows process model m_2 - is determined by the sequence of its operations as follows:

$$\left.\begin{aligned} Pre(t_{j,1}) = Pre(t_{j,4}) = \emptyset, \\ Pre(t_{j,2}) = \{t_{j,1}\}, \\ Pre(t_{j,3}) = \{t_{j,2}\}, \\ Pre(t_{j,5}) = \{t_{j,4}\}, \\ Pre(t_{j,6}) = \{t_{j,5}\}, \\ Pre(t_{j,7}) = \{t_{j,6}\}, \\ Pre(t_{j,8}) = \{t_{j,7}\}, \\ Pre(t_{j,9}) = \{t_{j,3}, t_{j,8}\}, \\ Pre(t_{j,10}) = \{t_{j,9}\}, \end{aligned}\right\} \forall j \text{ where } WM(w_j) = m_2.$$

The types of operations are:

$$\left.\begin{aligned} TT(t_{i,5}) = TT(t_{j,2}) = t^A, \\ TT(t_{i,2}) = TT(t_{j,5}) = t^B, \\ TT(t_{j,7}) = t^C, \\ TT(t_{i,7}) = TT(t_{j,9}) = t^D, \\ TT(t_{i,4}) = TT(t_{j,1}) = t^E, \\ TT(t_{i,1}) = TT(t_{j,4}) = t^F, \\ TT(t_{i,3}) = t^G, \\ TT(t_{j,6}) = t^H, \\ TT(t_{j,8}) = t^I, \\ TT(t_{i,6}) = TT(t_{j,3}) = t^J, \\ TT(t_{i,8}) = TT(t_{j,10}) = t^K, \end{aligned}\right\} \forall i \text{ where } WM(w_i) = m_1 \text{ and } \forall j \text{ where } WM(w_j) = m_2.$$

Operation types t^E, t^F, t^G, t^H, t^I, t^J, and t^K are transportation operation types.

Based on Figure 3.1, the resources of the problem are:
$R = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9, r_{10}, r_{11}, r_{12}, r_{13}, r_{14}, r_{15}\}$.

The types of these resources are:
$RT(r_1) = RT(r_2) = RT(r_3) = D_1$,
$RT(r_4) = D_2$,
$RT(r_5) = RT(r_6) = D_3$,
$RT(r_7) = RT(r_8) = D_4$,
$RT(r_9) = D_5$,
$RT(r_{10}) = D_6$,
$RT(r_{11}) = D_7$,
$RT(r_{12}) = D_8$,
$RT(r_{13}) = D_9$,
$RT(r_{14}) = D_{10}$,
$RT(r_{15}) = D_{11}$.
Resource types D_5-D_{11} are transportation resource types, and the operations of the process models they are capable of operating are transportation tasks.

The numbers of the resources with the same resource type are:
$n(D_1) = 3$, $n(D_2) = 1$, $n(D_3) = 2$, $n(D_4) = 2$, $n(D_5) = 1$, $n(D_6) = 1$, $n(D_7) = 1$, $n(D_8) = 1$, $n(D_9) = 1$, $n(D_{10}) = 1$ and $n(D_{11}) = 1$.

The resource-related data are summarized in Table J.1.

Table J.1. Resource types and their properties: operation time for the resource type - operation type pairs.

	t^A	t^B	t^C	t^D	t^E	t^F
D_1	∞	$dur(t^B, D_1)$	$dur(t^C, D_1)$	∞	∞	∞
D_2	∞	$dur(t^B, D_2)$	$dur(t^C, D_2)$	∞	∞	∞
D_3	$dur(t^A, D_3)$	∞	∞	∞	∞	∞
D_4	∞	∞	∞	$dur(t^D, D_4)$	∞	∞
D_5	∞	∞	∞	∞	∞	$dur(t^F, D_5)$
D_6	∞	∞	∞	∞	$dur(t^E, D_6)$	∞

	t^G	t^H	t^I	t^J	t^K
D_7	∞	$dur(t^H, D_7)$	∞	∞	∞
D_8	∞	∞	$dur(t^I, D_8)$	∞	∞
D_9	$dur(t^G, D_9)$	∞	∞	∞	∞
D_{10}	∞	∞	∞	$dur(t^J, D_{10})$	∞
D_{11}	∞	∞	∞	∞	$dur(t^K, D_{11})$

t^A, t^B, t^C, t^D, t^E, t^F, t^G, t^H, t^I, t^J, t^K are operation types
D_1, D_2, D_3, D_4, D_5, D_6, D_7, D_8, D_9, D_{10}, D_{11} are resource types

Appendix K

Parameter values of the deterministic problem of Chapter 3

Here the parameter values of the deterministic problem of Chapter 3 are presented formally, using the notations of Appendix B.

In the deterministic problem, $n(p_1) = 15$ and $n(p_2) = 20$. It means that 15 products with product type p_1 and 20 products with product type p_2 have to be produced. So, the scheduling problem is to schedule 35 workflows altogether: $|W| = 35$, with the structure that $|\bigcup_{i=1}^{n} w_i$, where $WM(w_i) = m_1| = 15$, and $|\bigcup_{i=1}^{n} w_i$, where $WM(w_i) = m_2| = 20$.

The operation times of the problem are presented in Table K.1.

Table K.1. Deterministic operation times of the problem.

Resource type × Task type $D_l \times t^S$	Operation time $dur(t^S, D_l)$	Duration after the resource is ready again $durRe(t^S, D_l)$
$D_1 \times t^B$	6	6
$D_1 \times t^C$	10	10
$D_2 \times t^C$	6	6
$D_2 \times t^B$	10	10
$D_3 \times t^A$	6	6
$D_4 \times t^D$	4	4
$D_5 \times t^F$	3	6
$D_6 \times t^E$	5	10
$D_7 \times t^H$	3	6
$D_8 \times t^I$	5	10
$D_9 \times t^G$	4	8
$D_{10} \times t^J$	3	6
$D_{11} \times t^K$	4	8

The setup times of the problem are:
$setup(D_1, t^B, t^C) = setup(D_1, t^C, t^B) = 5$
$setup(D_2, t^C, t^B) = setup(D_2, t^B, t^C) = 6.$

Appendix L

Computational results for Chapter 3

L.1 Results obtained by the unaided Genetic Algorithm for the deterministic problem

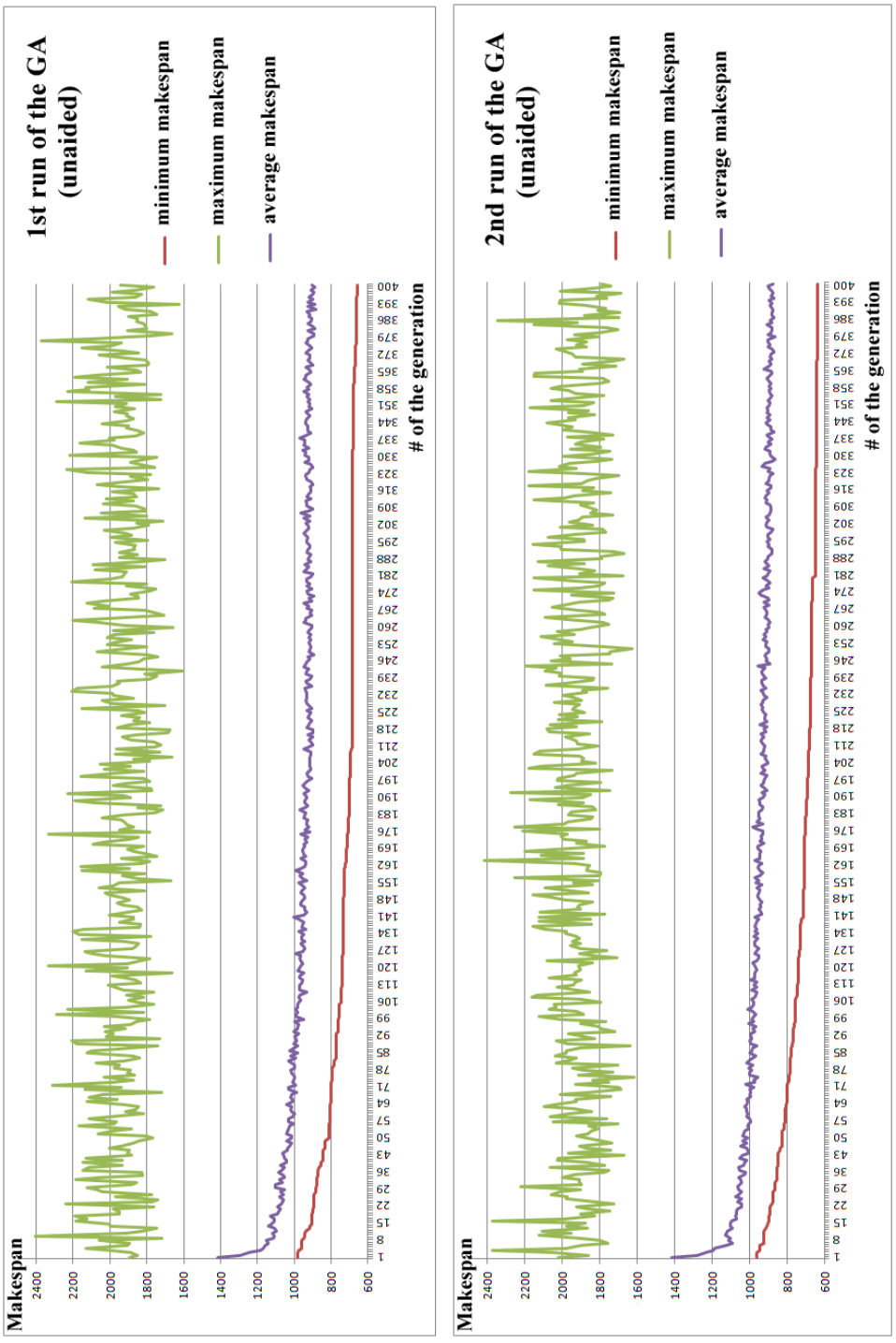


Figure L.1. Results for the best makespans related to the generation counter in case of the deterministic problem. Run 1-2.

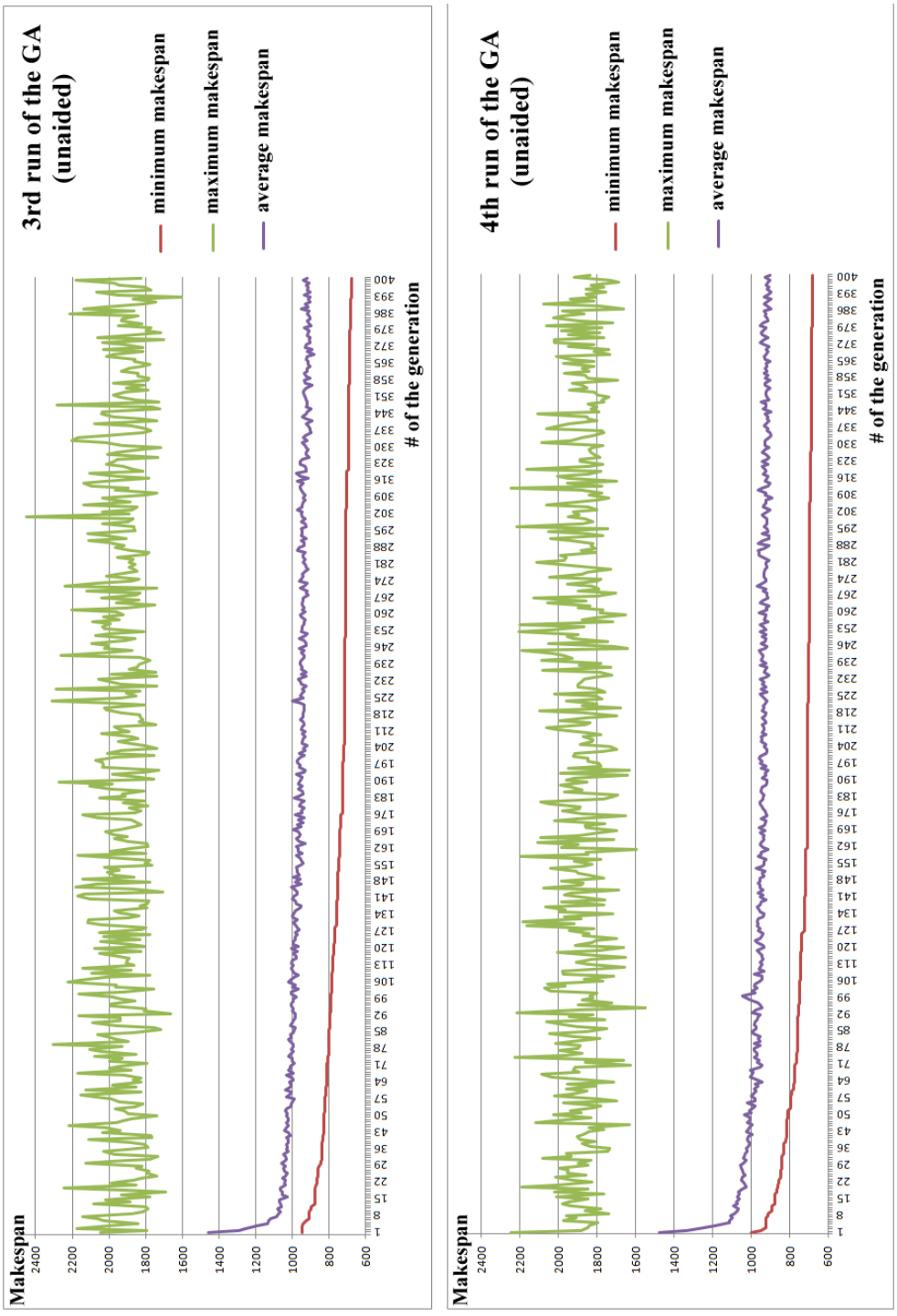


Figure L.2. Results for the best makespans related to the generation counter in case of the deterministic problem. Run 3-4.

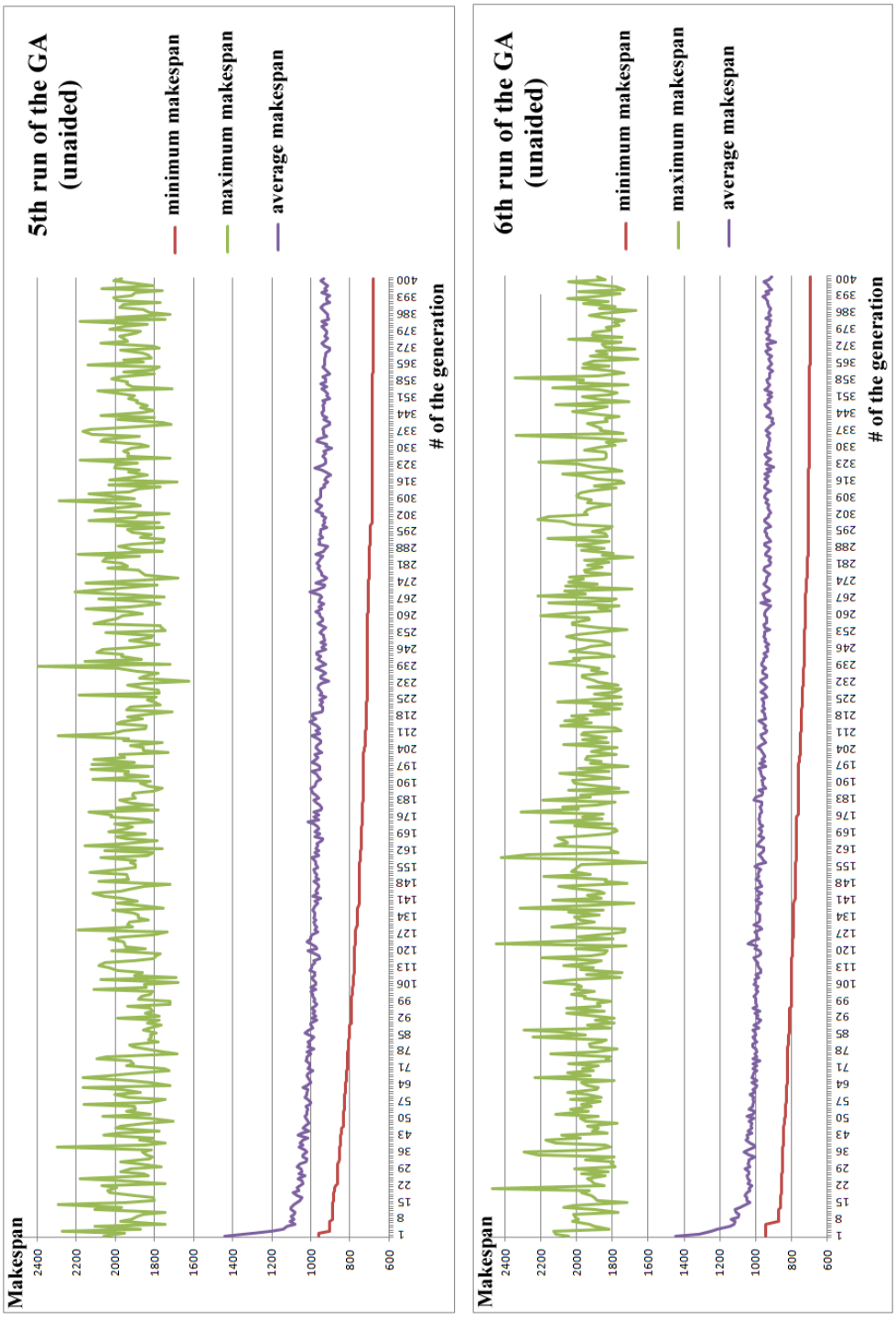


Figure L.3. Results for the best makespans related to the generation counter in case of the deterministic problem. Run 5-6.

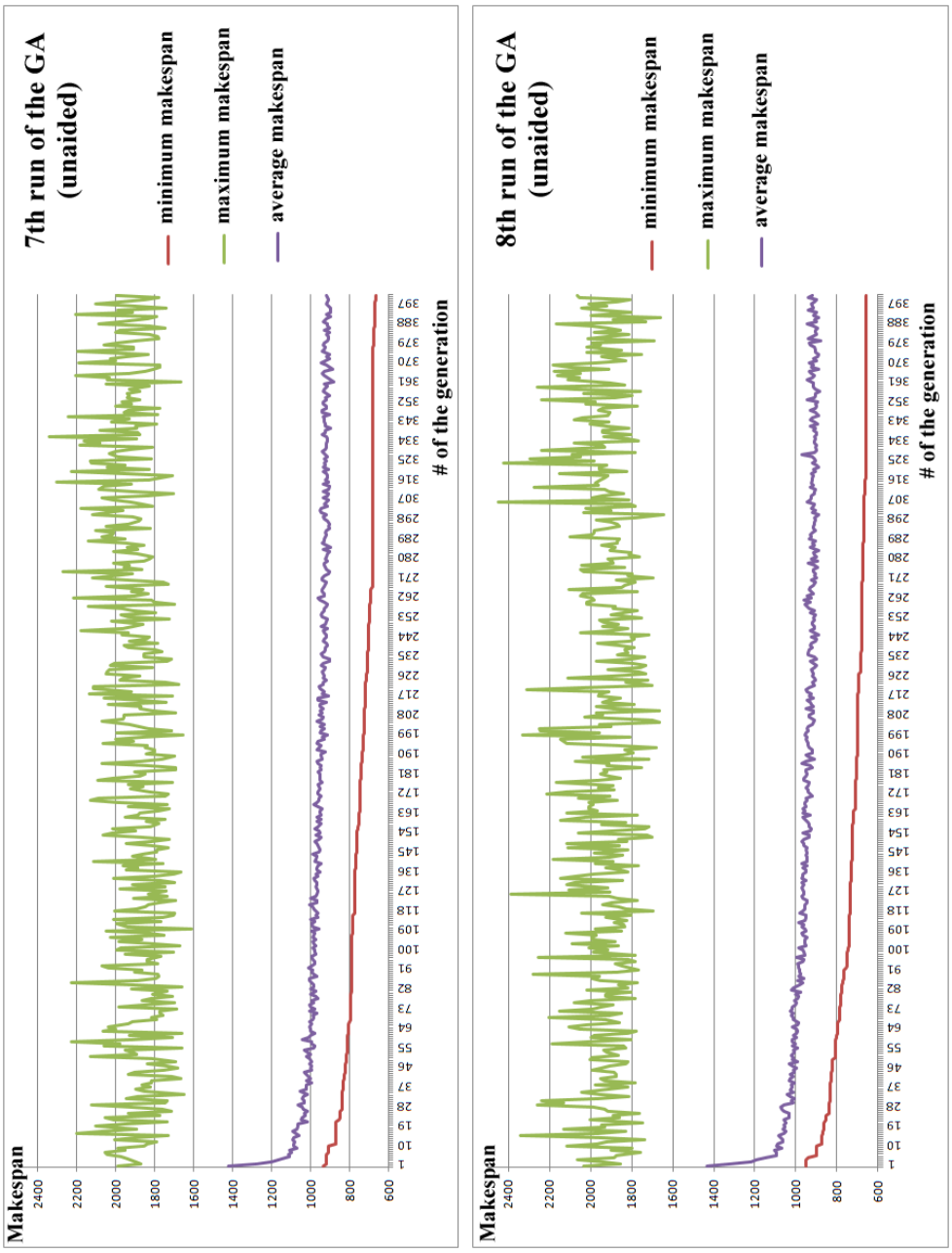


Figure L.4. Results for the best makespans related to the generation counter in case of the deterministic problem. Run 7-8.

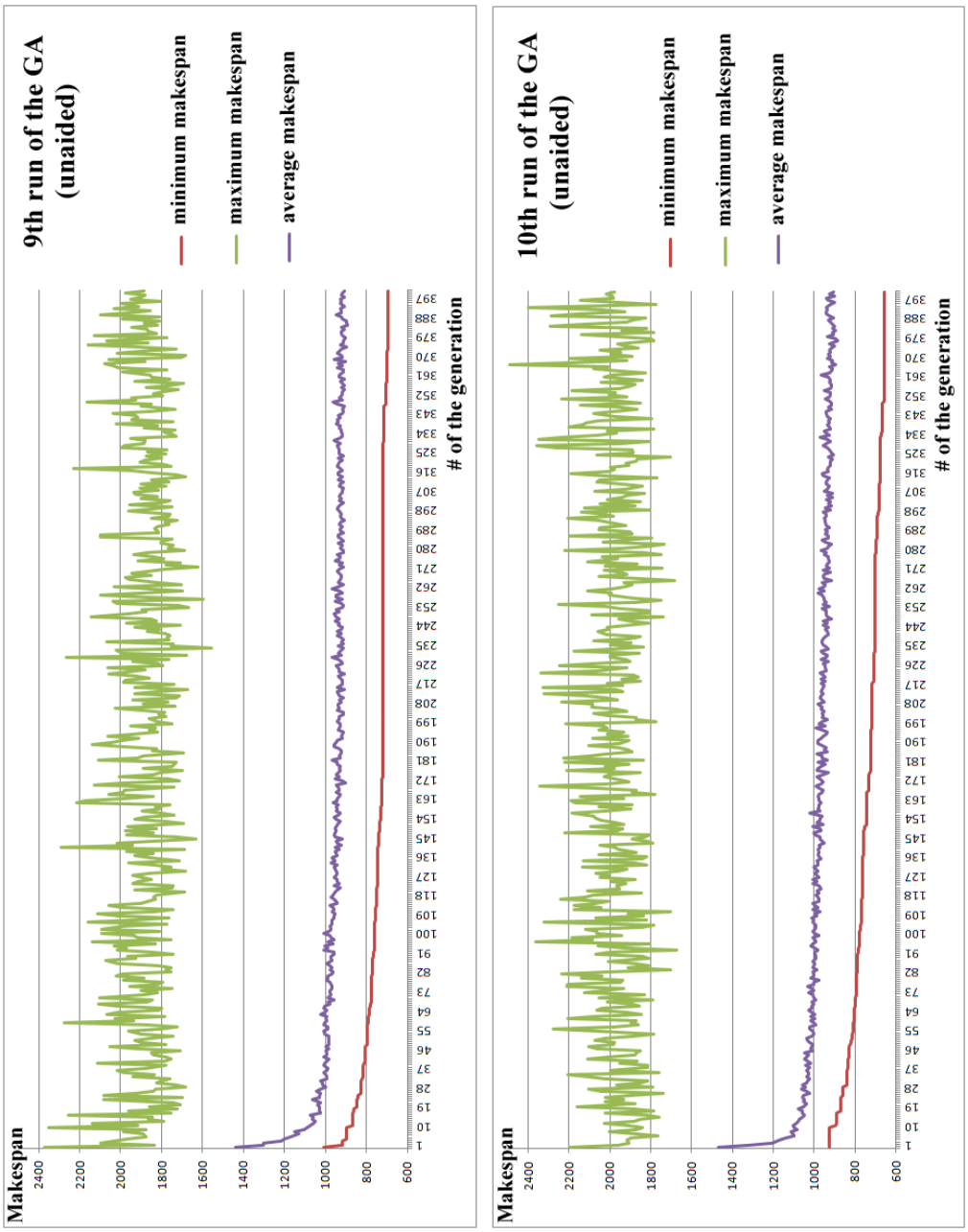


Figure L.5. Results for the best makespans related to the generation counter in case of the deterministic problem. Run 9-10.

L.2 Results obtained by Simulated Annealing for the deterministic problem

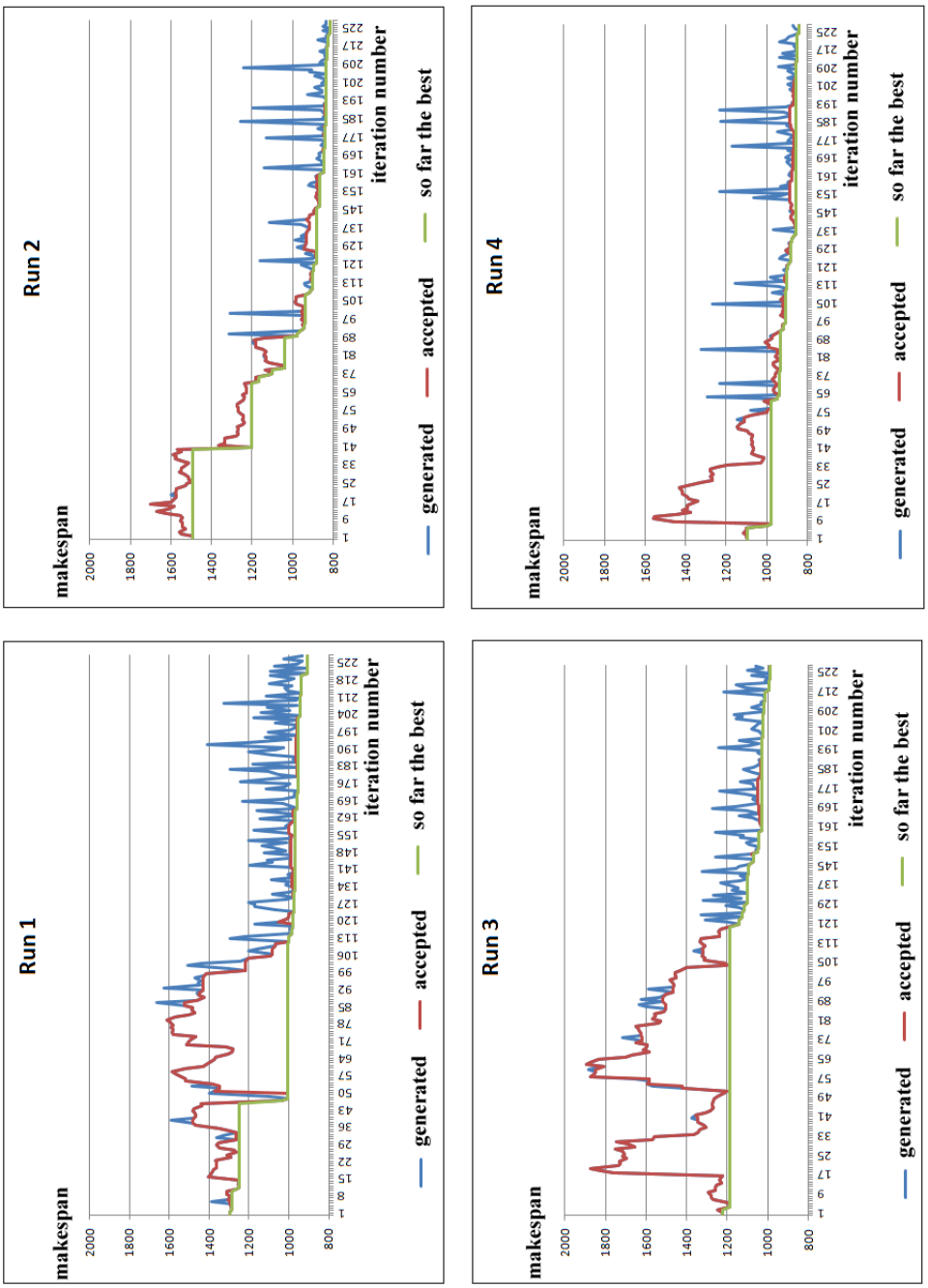


Figure L.6. Results for the best makespans related to the iteration number in case of the deterministic problem. Run 1-4.

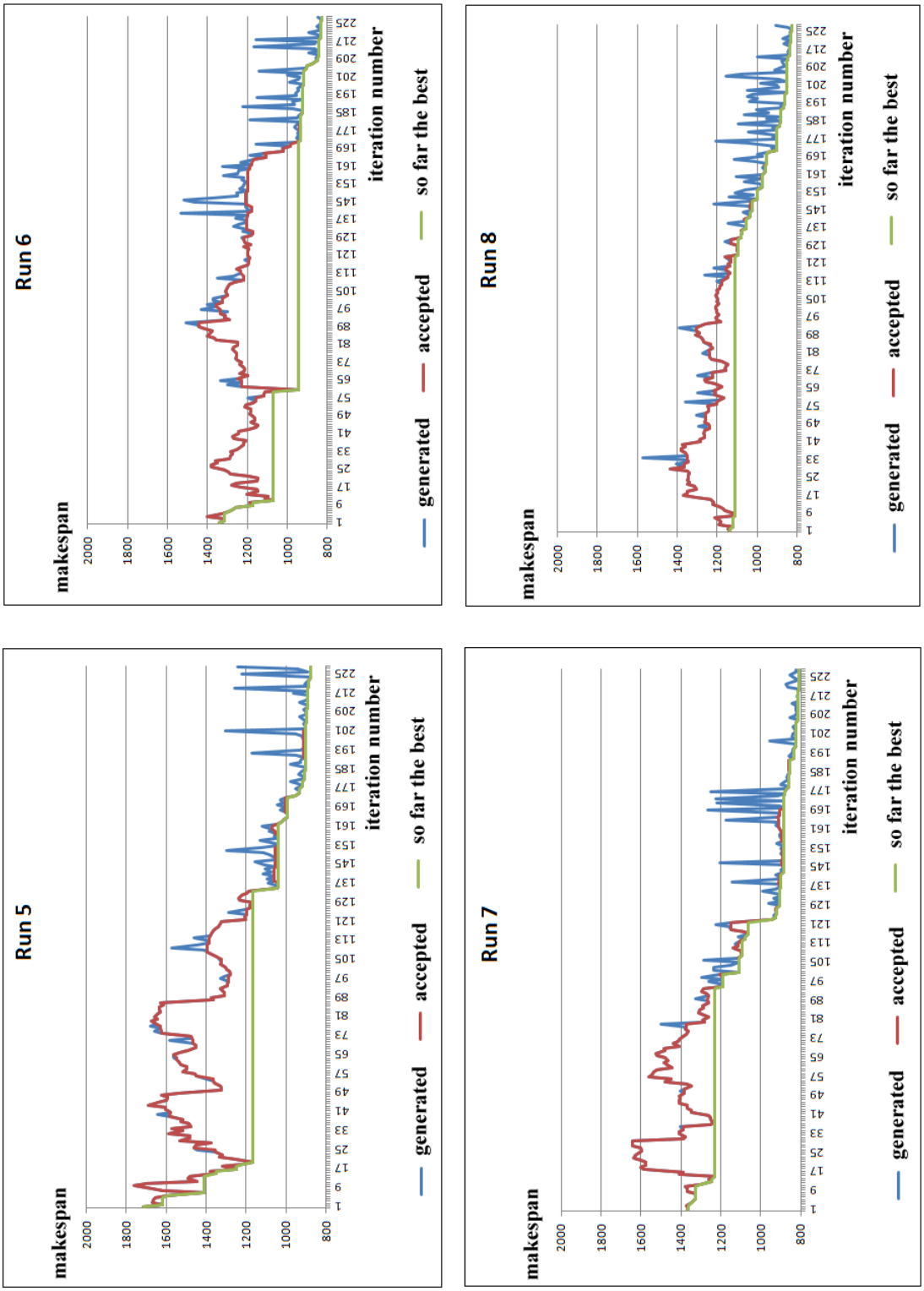


Figure L.7. Results for the best makespans related to the iteration number in case of the deterministic problem. Run 5-8.

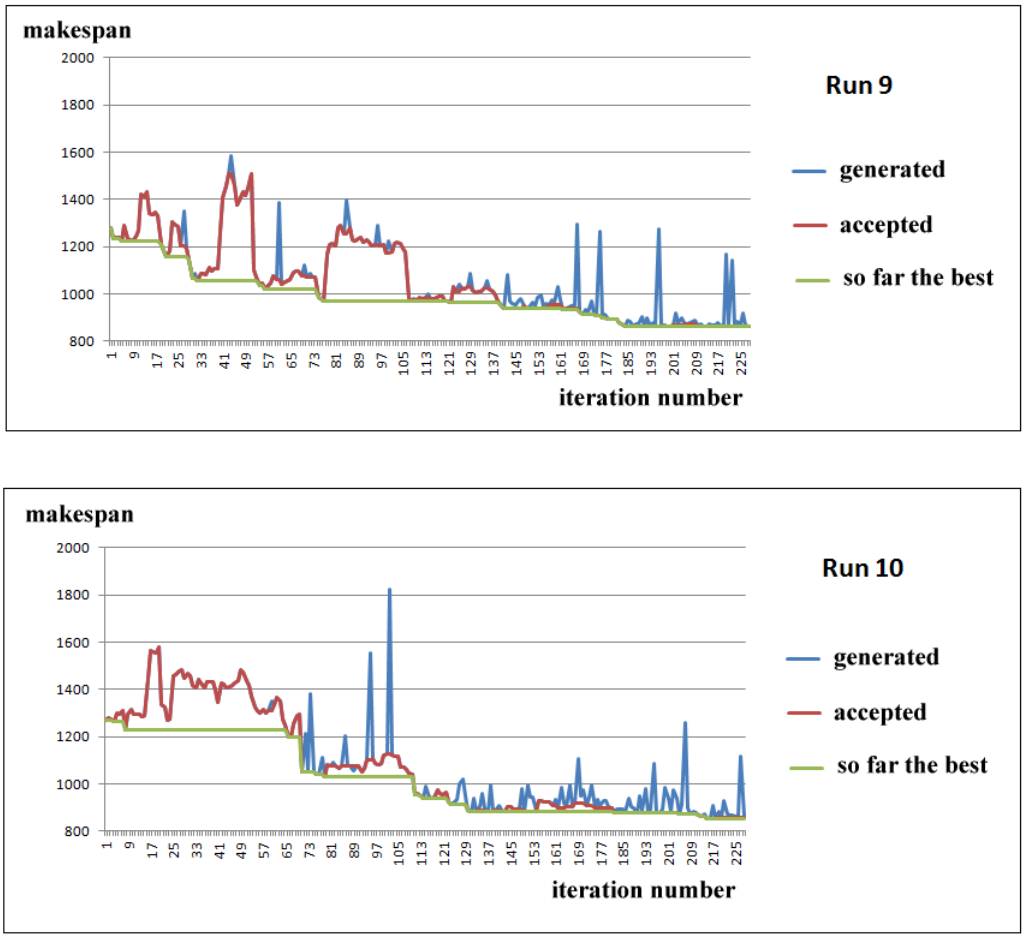


Figure L.8. Results for the best makespans related to the iteration number in case of the deterministic problem. Run 9-10.

L.3 Results obtained by the Simulated Annealing-aided Genetic Algorithm for the deterministic problem

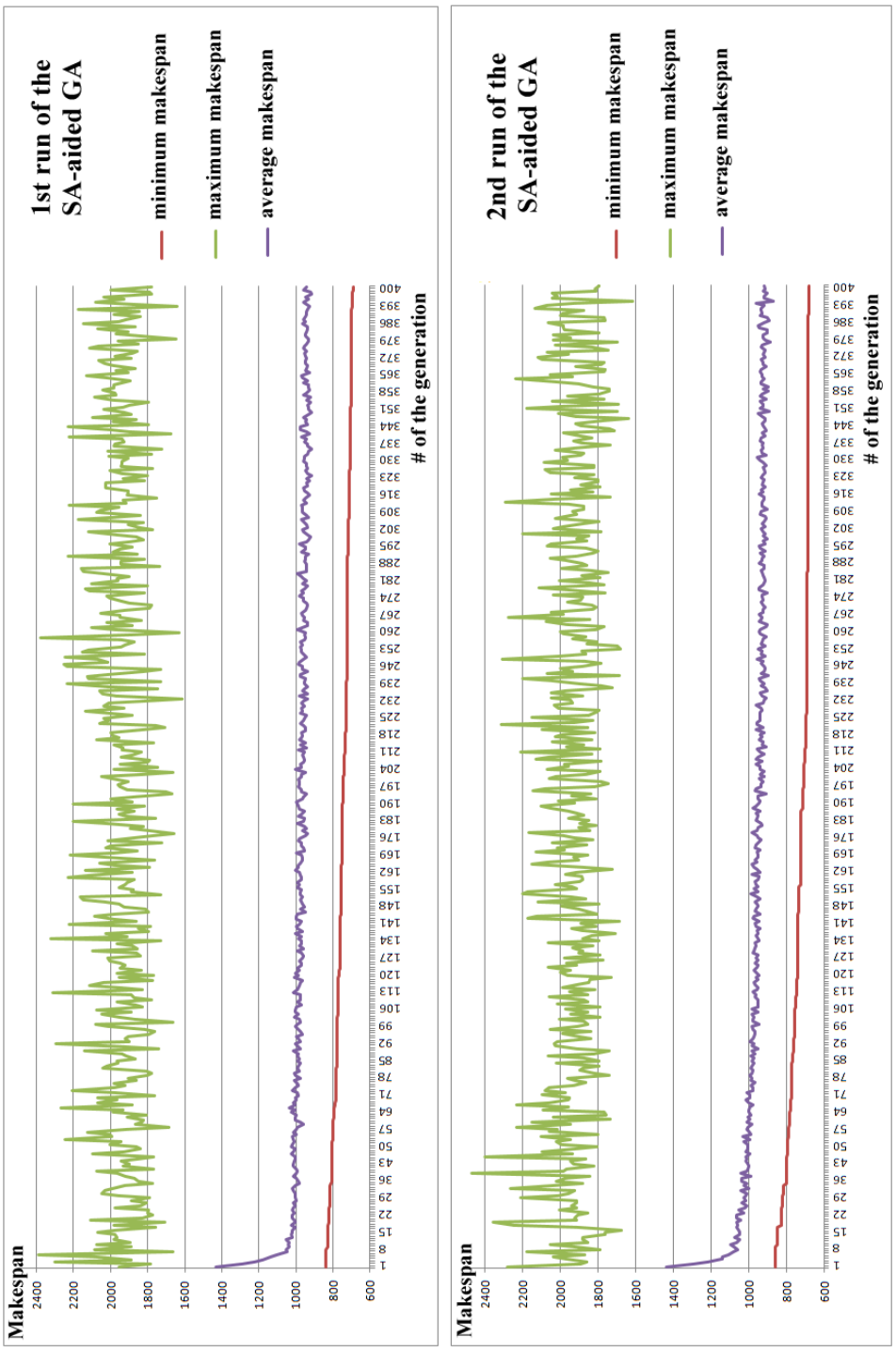


Figure L.9. Results for the best makespans related to the generation counter in case of the deterministic problem. Run 1-2.

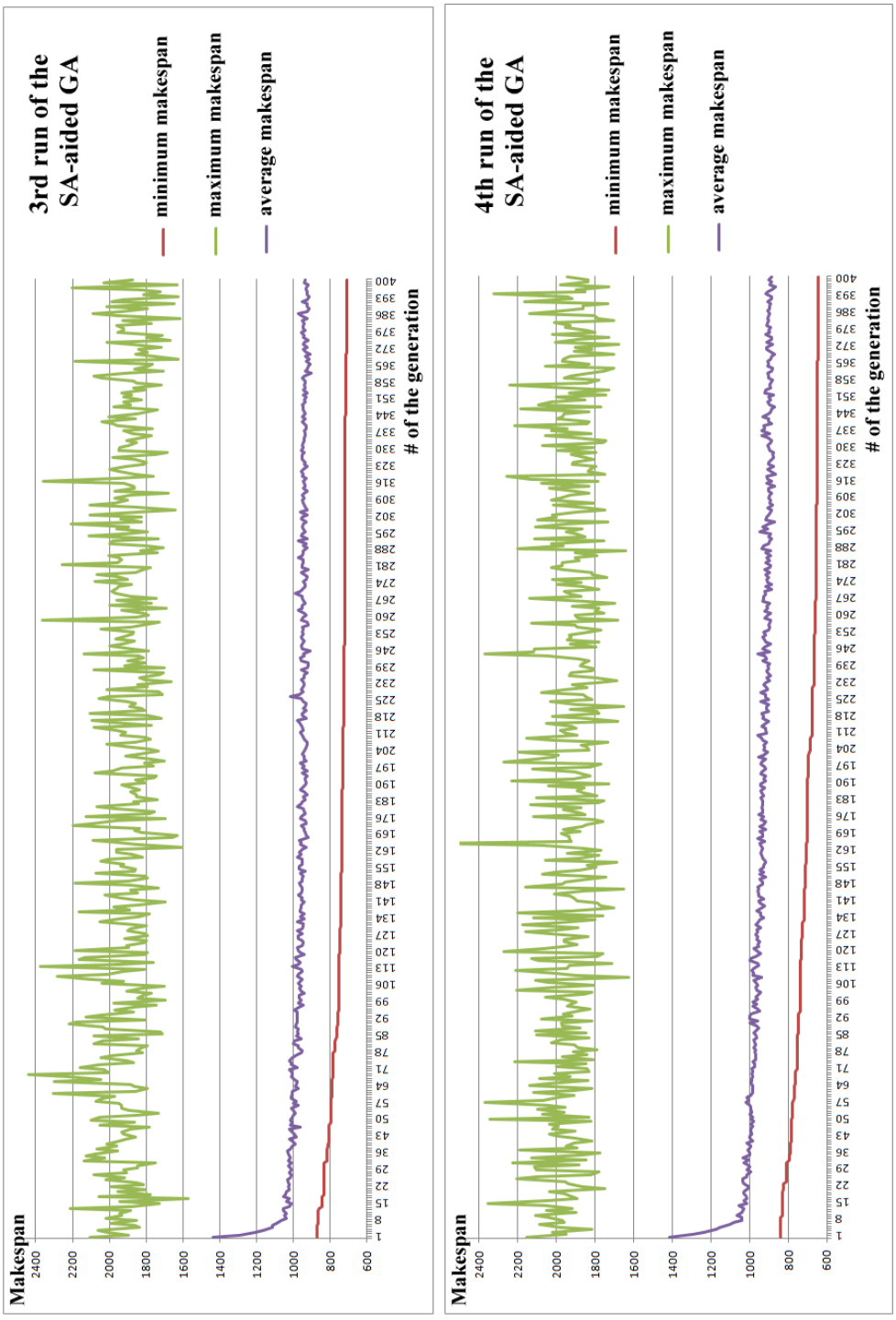


Figure L.10. Results for the best makespans related to the generation counter in case of the deterministic problem. Run 3-4.

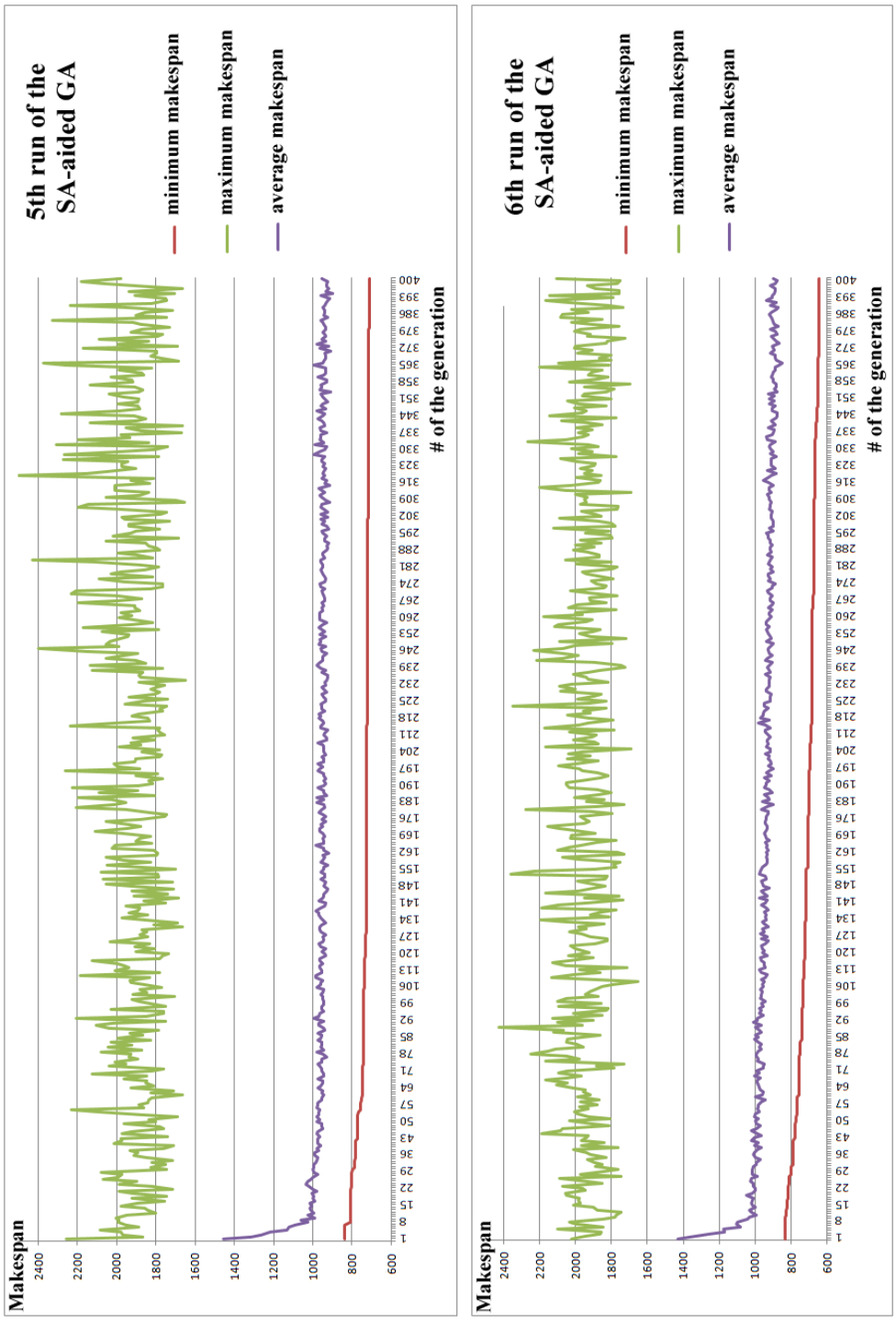


Figure L.11. Results for the best makespans related to the generation counter in case of the deterministic problem. Run 5-6.

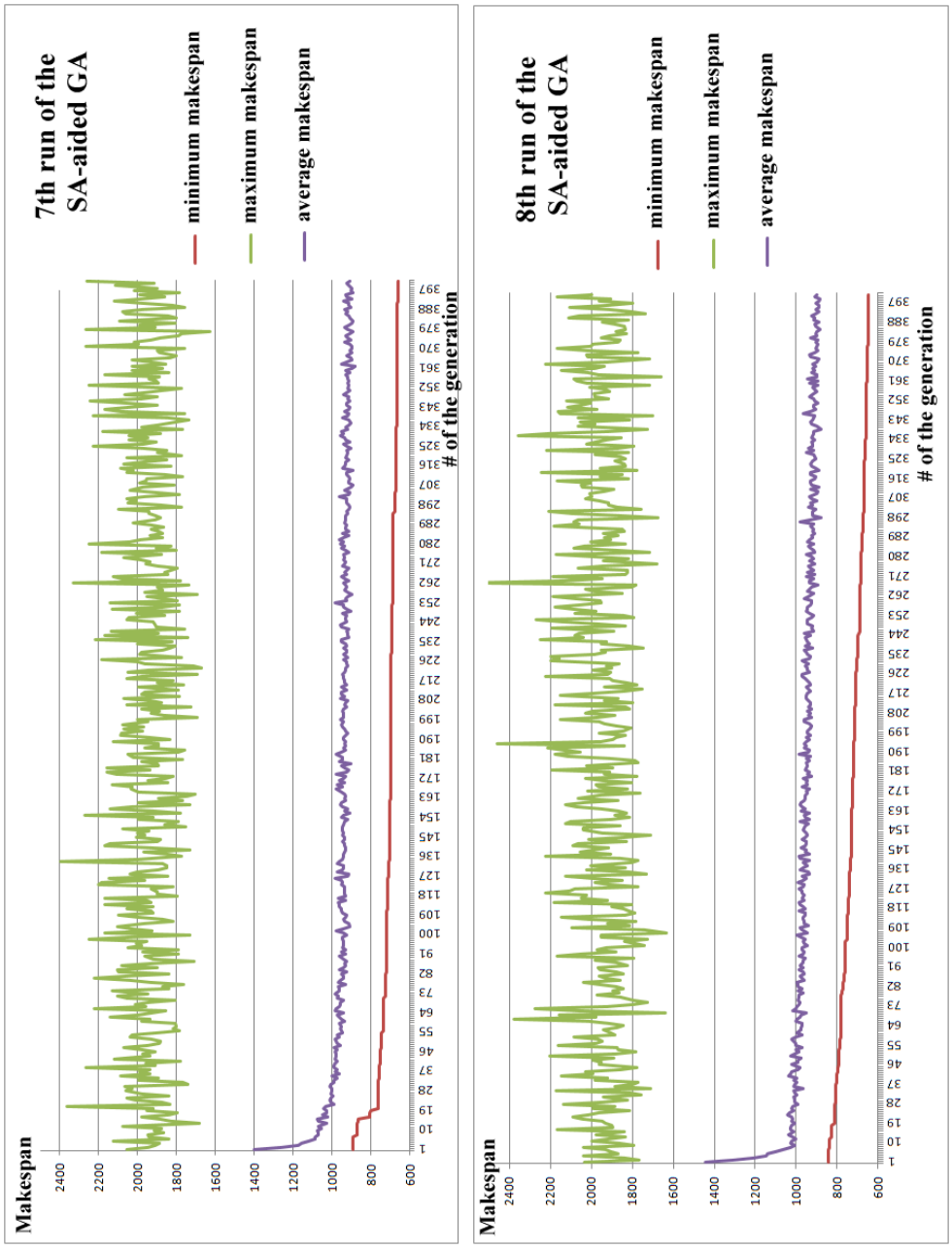


Figure L.12. Results for the best makespans related to the generation counter in case of the deterministic problem. Run 7-8.

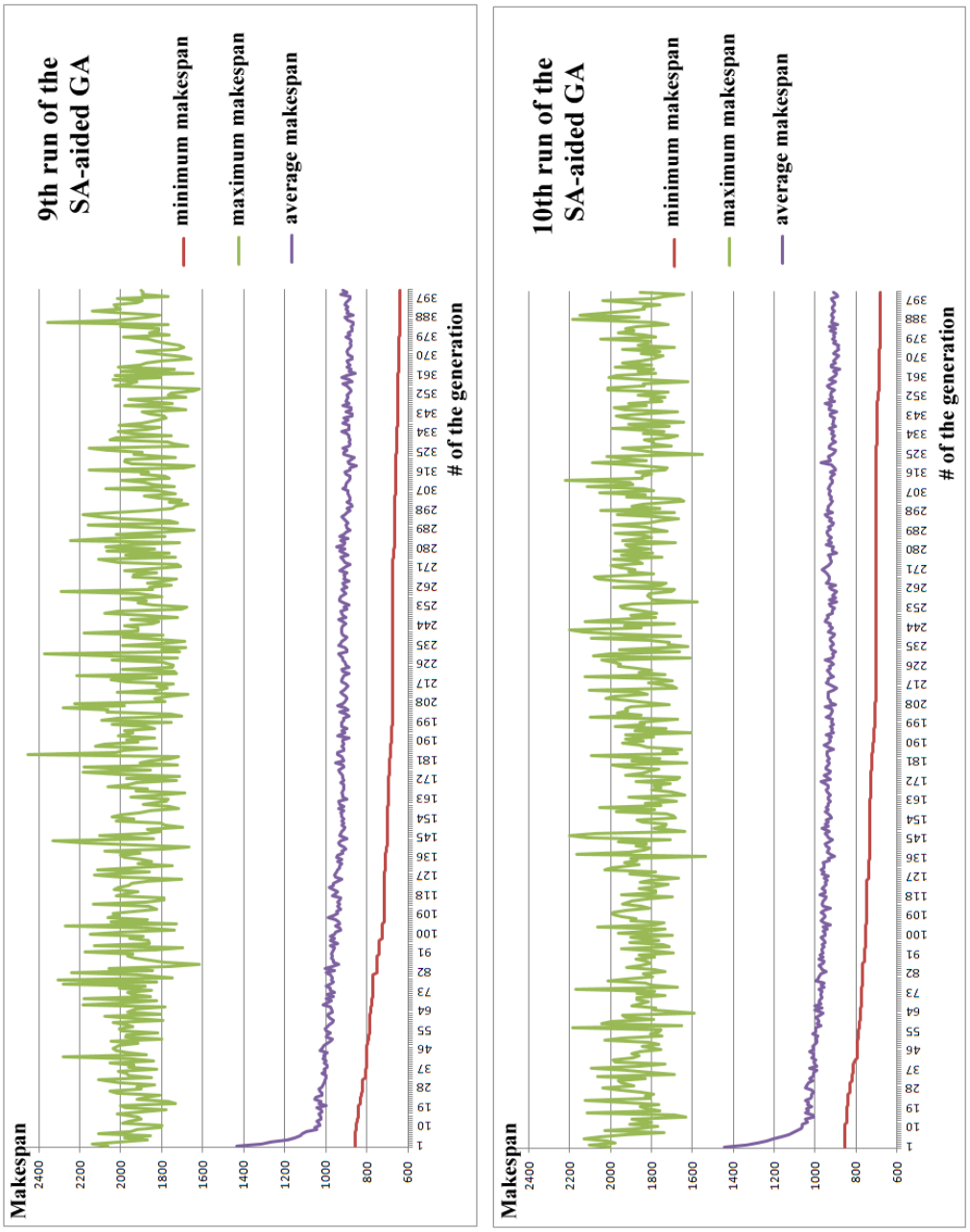


Figure L.13. Results for the best makespans related to the generation counter in case of the deterministic problem. Run 9-10.

Appendix M

The detailed improvement steps of the genetic algorithm of Subsection 3.6

First improvement: using heuristic - that works from job to job - to create the initial population

The original GA gave significant importance to suddenness because the main purpose was to cover the whole search space. However, I was not satisfied with the results' quality, and I thought it could be improved. The purpose of the first improvement is to fill the initial population with good-quality instances that are obtained quickly by a heuristic. This way, the GA can start from a better state than in the original case.
Analyzing the input of the $mk01$ problem - that has 6 machines, 10 jobs, 5-6 operations per job and the average number of machines per operation (that can perform the operation) is 2 -, some bottlenecks were noticed: there are "critical operations" and because of them, "critical machines". These operations can determine the priority of their processes during scheduling (instead of the random sequence of the processes).

Definition 80. *Critical machine: Let T^{one} be the set of all of those operations of the problem that can be performed by only one resource. If the operation times of the elements of T^{one} are summed per machine, the machine with the highest (nonzero) sum is a critical machine (res_1). If the sum of the machine with the second-highest value is within 80% of the sum of res_1, then this machine (res_2) is also critical.*

Definition 81. *Critical operation: Let avg be the average of the operation times of the elements of T^{one}. Each element of T^{one}, whose operation time equals or higher than avg and the only machine that can perform it is res_1 or res_2, is a critical operation.*

Let T^C denote the set of the critical operations of the problem. For each element of T^C, both the sum of the minimum operation times of its preceding operations in its process ($sumPrev$) and the sum of the minimum operation times of its subsequent operations in its process ($sumFollower$) are calculated. The averages of these values

on the T^C set are also determined: $sumPrevAvg$ and $sumFollowerAvg$. Based on these values, the critical operations are classified into the following priority classes:

- I. (weight: 3): $sumPrev = 0$ and $sumFollower \geq sumFollowerAvg$.
- II. (weight: 1): ($sumPrev = 0$ and $sumFollowerAvg > sumFollower > 0$) or ($sumPrevAvg > sumPrev > 0$ and $sumFollower \geq sumFollowerAvg$).
- III. (weight: 0): ($sumPrevAvg > sumPrev > 0$ and $sumFollowerAvg > sumFollower > 0$) or ($sumPrev \geq sumPrevAvg$ and $sumFollower \geq sumFollowerAvg$) or ($sumPrev = 0$ and $sumFollower = 0$).
- IV. (weight: -1): ($sumPrev \geq sumPrevAvg$ and $sumFollowerAvg > sumFollower > 0$) or ($sumPrevAvg > sumPrev > 0$ and $sumFollower = 0$).
- V. (weight: -3): $sumPrev \geq sumPrevAvg$ and $sumFollower = 0$.

Then, a weighted sum of the number of the critical operations is calculated for each process. The weights depend on the priority class; their values are highlighted in the previous enumeration. The processes are classified based on their weighted sum, as follows:

- Class 1., if the weighted sum is greater than or equal to 3.
- Class 2., if the weighted sum is between 0 and 3.
- Class 3., if the weighted sum is equal to 0.
- Class 4., if the weighted sum is between -3 and 0.
- Class 5., if the weighted sum is less than or equal to -3.

During scheduling the processes, the heuristic method follows their increasing order by their class numbers (the order of the processes with equal class numbers is random). This method ensures that an operation that can be performed by only one machine (thus, it is harder to schedule) and whose process still requires a long operation time after performing this operation is scheduled earlier - and can be performed earlier - on average. To maintain the genetic variability, maybe not all the instances of the initial population must be generated by this heuristic. Experiments were done on creating different sized parts of the initial population by the heuristic method (the creation of the remaining instances is unchanged, following the method of subsection 3.3). Because of the long runtime of the GA, the generation number was decreased to 800. The results obtained by running the GA until its 800^{th} generation on the $mk01$ test instance of Brandimarte are shown in Table M.1.

The results justify the efficiency of the heuristic method. The best results were obtained when the whole initial population was created by the heuristic. From now on - in each further improvement - the initial population is created by this heuristic. However, since the obtained best makespan is far from the optimum, further improvement is necessary.

Table M.1. The makespans, obtained on the $mk01$ problem by the GA after the first improvement, with different numbers of instances created by the heuristic in the 50-sized initial population.

The number of instances created by the heuristic in the 50-sized initial population	Results of the GA*			
	optimum	minimum	maximum	average
1	40	49	60	54
5	40	48	59	52.5
50	40	46	54	50.7

*from 10 runs

Table M.2. The makespans, obtained on the $mk01$ problem by the GA after the second improvement.

Results of the GA*			
optimum	minimum	maximum	average
40	42	44	43.1

*from 10 runs

Second improvement: modifying the heuristic that creates the initial population to schedule from operation to operation in a greedy way

The first improvement applied a heuristic that orders the jobs and schedules all operations of a job before the next job's operations are scheduled. It excludes the creation of some good solutions; the proof for that can be found in Appendix N. The second improvement modifies the heuristic in this way:
The scheduling is done from operation to operation. In each step, all the possible subsequent, unscheduled operations of the processes are collected, and the operation that results in the minimum makespan after its schedule is selected and becomes scheduled. If more than one such operations exist, one of them is chosen randomly. Because the runtime of the heuristic is short, 1000 instances are generated by the heuristic method, and the initial population is filled by the best 50 of them. The number of the generations remains 800. Table M.2 shows the obtained results on the $mk01$ problem of Brandimarte.

These results are convincing: the best makespan improved from 46 to 42. Because of this, this variant of the algorithm was run on other test instances, too (see Table M.3).

The results are better in the case of $mk01$ and $mk03$; however, they are far from the optimum in the case of $mk02$ and $mk04$. It necessitates further improvements.

Third improvement: crossover applies scheduling from operation to operation instead of from job to job

Because of the success of the heuristic, the scheduling method used in the crossover is also changed to operate from operation to operation (instead of its original way,

Table M.3. The makespans, obtained on different test problems of Brandimarte by the GA after the second improvement.

		Results of the GA		
name of the test instance	optimum	minimum	maximum	average
mk02	26	32	35	33.9
mk03	204	210	217	213.9
mk04	60	74	77	75.6

Table M.4. The makespans, obtained on the $mk02$ problem by the GA after the third improvement.

Results of the GA*			
optimum	minimum	maximum	average
26	32	35	33.6

*from 10 runs

which operated from job to job - see subsection 3.3). The resource allocation of the modified crossover remains: the operations of the randomly chosen half of the child's processes follow the resource allocation of the first parent, and the remaining operations are scheduled on the same resource as they were in the second parent. At the same time, the order of their scheduling changes. Instead of ordering the processes and scheduling all the operations of a process before handling the subsequent process's operations, the scheduling is now made from operation to operation. In each step, the operation that results in minimum makespan after its schedule is selected (independently from its process).
Because the previous results obtained on the $mk02$ problem were far from the optimum, this modification of the GA is tested on this test instance. The heuristic of the second improvement - that fills the initial population by the best 50 instances of the created 1000 solutions - is also used, and the number of the generations is 800. The results are summarized in Table M.4.

The improvement that was reached is minimal: the best makespan remained 32, and the average of the makespans decreased from 33.9 to 33.6. That is why I decided to improve the GA further.

Fourth improvement: modified fitness and controlled selection

So far, the fitness of a solution was its makespan, and the selection method of the parents to recombine was random. The third improvement modifies these properties of the GA.
Three alternative fitness functions were developed:

- **(1)** Both the makespan of the whole schedule (C_{max}) and each machine's completion time (C_i in case of resource r_i) influence the fitness value. The fitness value of a schedule is $C_{max} + (1 - y)$, where $y = \frac{\sum_{i=1}^{u} (\frac{C_i}{C_{max}})^2}{u}$ and u is the number of machines.

Table M.5. The makespans, obtained on the $mk02$ problem by different variants of the GA after the fourth improvement.

		Results of the GA		
combination	optimum	minimum	maximum	average
(1)+(a)	26	33	35	34
(1)+(b)	26	33	35	34
(2)+(a)	26	32	34	33.17
(2)+(b)	26	33	35	34.17
(3)+(a)	26	33	35	33.67
(3)+(b)	26	33	35	33.84

- **(2)** The fitness calculation of (1) is extended by also taking the resources' load into account. If the sum of the idle times of machine r_i is $idle_i$ and the number of the machines is u, then the fitness value of a schedule is $C_{max}+(1-y)+(1-z)$, where $z = \frac{1}{u} * \sum_{i=1}^{u} (\frac{idle_i}{\max\limits_{j=1\ldots u} idle_j})^2$ and y is calculated the same way as in the case of (1).

- **(3)** The fitness calculation of (2) is extended by the different weighting of the critical and not critical machines (for the definition of critical machines, see Definition 80). Here, y and z values are calculated separately, both for the critical machines (y_c and z_c) and the non-critical machines (y_d and z_d). Then, $y = \alpha * y_c + (1 - \alpha) * y_d$ and $z = \alpha * z_c + (1 - \alpha) * z_d$, where $\alpha = 1/3$ was used. Finally, the fitness value of a schedule is calculated the same way as in the case of (2): $C_{max} + (1 - y) + (1 - z)$.

The controlled selection was realized in two different ways:

- **(a)** both parents were selected from the elites

- **(b)** one parent was selected from the elites, and the other parent was chosen from the instances outside of the elites

All the 6 possible combinations of the 3 fitness calculations and the 2 controlled selections were examined; Table M.5 shows the results. During the examinations, all the previous improvements were also applied (the initial population was filled by the best 50 instances of the 1000 solutions generated by the heuristic and the crossover applied scheduling from operation to operation). The GA was run through 800 generations.

The best results were obtained when calculating the fitness value involves the load of the machines, and both parents to recombine were chosen from the elites. Then, the best makespan was the same as previously (32); however, the makespans' average improved from 33.6 to 33.17. Unfortunately, the obtained best result is not close enough to the optimum, so further improvements were needed. In the further works, the most efficient combination ((2)+(a)) of this phase was applied.

Table M.6. The makespans, obtained on the $mk02$ problem by the GA after the fifth improvement attempt.

	Results of the GA		
optimum	minimum	maximum	average
26	42	49	45.67

Table M.7. The makespans, obtained on the $mk02$ problem by the GA - with the original 0.18 mutation rate - after the sixth improvement attempt.

	Results of the GA		
optimum	minimum	maximum	average
26	33	36	33.92

Fifth attempt for improvement: modified mutation - inserting each rescheduled operation into the earliest feasible position

The mutation of the GA operates by rescheduling each operation of a randomly chosen process. So far, for each rescheduled operation, a random resource was selected from the set of the machines that can perform the operation, and the operation was inserted into a random, feasible position in the machine's operation queue. This attempt selects the earliest feasible position of the chosen machine's operation queue for the insertion instead of a random one. All the other methods and parameters of the (2)+(a) variant of the previous improvement phase are kept. Table M.6 shows the results.

This attempt resulted in much worse results than the previous runs, so this modification of the mutation was rejected. The selection of the new positions of the rescheduled operations of the mutation phase remains random in the further improvement attempts.

Sixth attempt for improvement: using heuristic for crossover

So far, the crossover worked as follows: the resource allocation for the child instance's operations in the case of the - randomly chosen - half of the processes followed the resource allocation of the first parent, and the remaining operations were allocated based on the second parent. This modification selects the 40% of the processes randomly and allocates their operation by a heuristic: the machine that results in the minimum makespan after scheduling the operation will be chosen for each operation of these processes. If more than one such machines exist, one of them is selected randomly. Random 30% of the child's processes follow the first parent's resource allocation, and the remaining operations are allocated based on the second parent. The scheduling is made from operation to operation; the presented modification of the crossover extends the best method of the fourth improvement. The results are presented in Table M.7.

Table M.8. The makespans, obtained on the *mk*02 problem by the GA - with mutation rate 0.1 - after the sixth improvement attempt.

	Results of the GA		
optimum	minimum	maximum	average
26	32	34	33.5

The test was also performed using a mutation rate of 0.1 (instead of the initial - 0.18 - mutation rate of the GA) to qualify the applied mutation. It gave the results of Table M.8.

These results indicate that the sixth attempt for improvement does not help much. Moreover, the present mutation operator of the GA seems to be inefficient. Thus, the further improvements are based on the best variant of the fourth improvement and focus on the improvement of the mutation genetic operator.

Seventh improvement: a complete review of the mutation genetic operator

The design of the initial GA's mutation operator - that reschedules each operation of a randomly selected process - seems to be inefficient. The seventh improvement replaces this method with a completely new mutation that picks an operation from the machine that causes the makespan (whose completion time is equal to the makespan of the schedule) randomly. (This is similar to the neighbour creation of Hurink et al. [1]; however, here, the picked operation is not necessarily part of the critical path.) The picked operation and its subsequent operations both on its machine and in its process are deleted from the schedule. This deletion is continued recursively for each deleted operation (so, every subsequent node of the selected node is deleted from the directed graph of the schedule). Then each deleted operation is rescheduled into the remaining schedule. The operation is selected in each iteration, which results in the schedule with minimum makespan after its insertion. For the originally picked operation, the machine is selected that can perform the operation and is the earliest with its finish. All the other operations are allocated to the machine, which they were deleted from. The whole method is executed 20 times on the initial instance, and the best result is chosen as the mutated instance.
Finally, this mutation was used in the GA in four different ways:

- **a. Random**: the new mutation is applied to any instance of the 50-sized population with a 40% probability.
- **b. Roulette**: 20 instances are selected to be mutated from the population by the roulette wheel method.
- **c. Best**: the new mutation is applied to the best 20 instances of the 50-sized population.
- **d. Worst**: the new mutation is applied to the worst 20 instances of the 50-sized population.

Table M.9. The makespans, obtained on the $mk02$ problem by the GA - with the different mutation variants - after the seventh improvement.

		Results of the GA		
the selection method for the mutation	optimum	minimum	maximum	average
Random	26	31	32	31,2
Roulette	26	31	32	31,8
Best	26	30	32	30.8
Worst	26	34	35	34.4

The results of the GA, which extends the algorithm of the fourth improvement by the new mutation, are presented in Table M.9. For the sake of shorter runtime, the number of generations was decreased to 200 from 800.

The best makespan obtained after the seventh improvement decreased to 30 (from 32); moreover, 32 was the worst makespan in the case of the first three variants. The best results were obtained when the mutation was applied to the best 20 instances of the population. This variant was chosen for the GA.

Appendix N

Why scheduling from operation to operation can be more efficient than from job to job scheduling?

Let us consider the problem of Figure N.1!

Process (workflow)	Operation	Machines and processing time	
		r_1	r_2
w_1	$t_{1,1}$	10	100
	$t_{1,2}$	100	10
w_2	$t_{2,1}$	100	2
	$t_{2,2}$	10	100
w_3	$t_{3,1}$	100	9
	$t_{3,2}$	11	100

Figure N.1. The problem.

If the schedule of its operations is created from job to job (no operation of the following jobs is scheduled before all the operations of the current job are scheduled), 6 different sequences of its 3 processes can be created. Figure N.2 shows the results of these 6 schedules when a greedy scheduler is applied.

However, if the problem is scheduled from operation to operation, better results can be reached. An example for this is illustrated in Figure N.3, where the sequence of the operations is $t_{1,1}, t_{3,1}, t_{2,1}, t_{3,2}, t_{1,2}, t_{2,2}$ during the greedy scheduling. Here, the obtained makespan is 31, while the best makespan obtained by scheduling from job to job was 32.

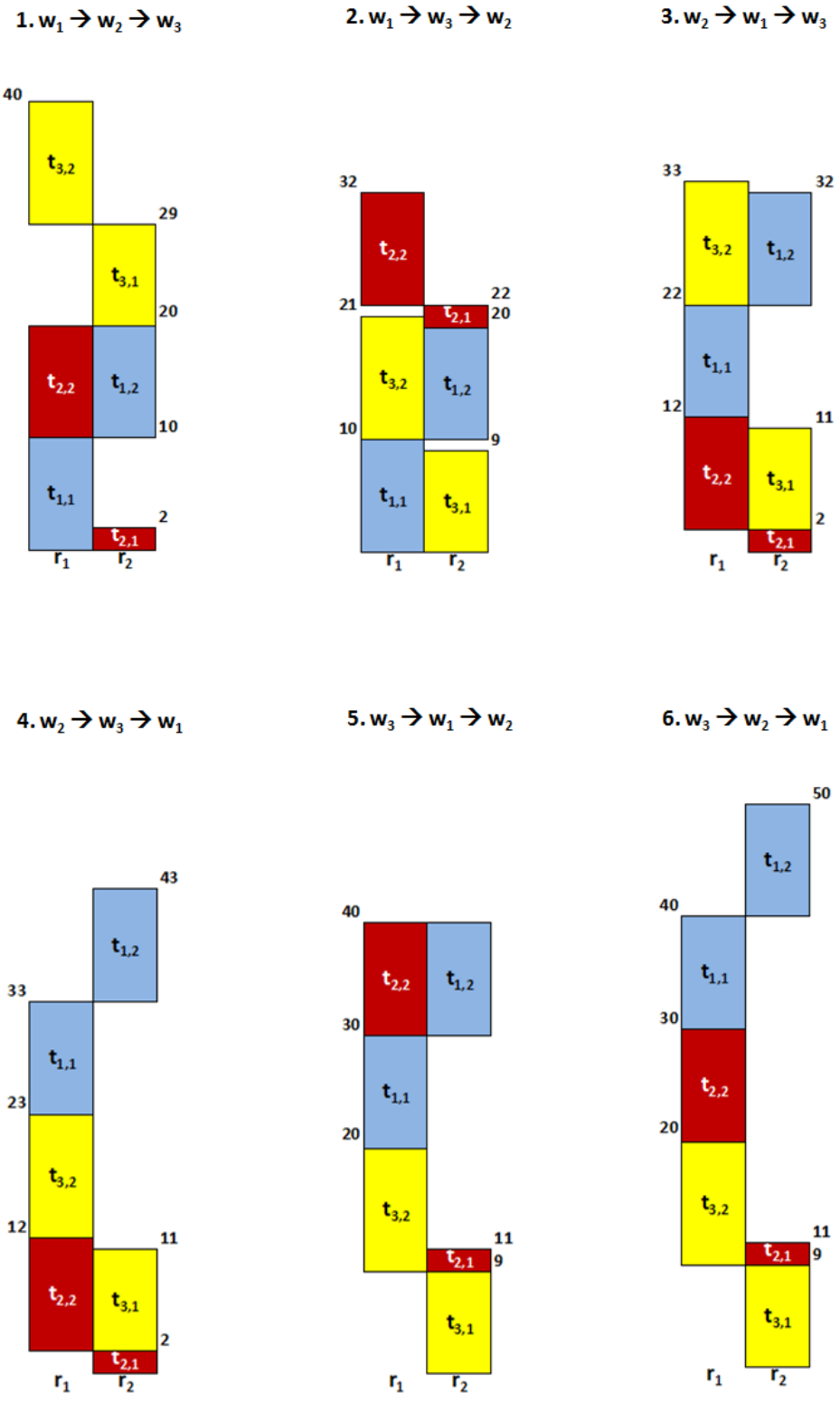


Figure N.2. The obtained schedules for the 6 possible job sequences when the greedy scheduling algorithm is executed from job to job.

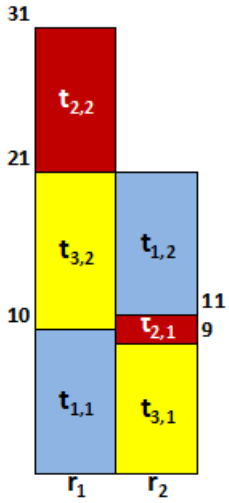


Figure N.3. A schedule of the problem that can be reached by greedy scheduling from operation to operation.

Although the presented problem is only one example, it is easy to see that in the case of scheduling problems with unrelated machines and precedence constraints, such a sequence of operations that results in an optimal solution by greedy scheduling can always be found. It is not true for the greedy scheduling from job to job.

Appendix O

Results of the improved GA of Chapter 3 on test instances from the scientific literature

The sequence of the tested instances in Tables O.2 and O.3 is not complete. Its reason is the size of the problems (and thus, the algorithm's runtime): smaller test problems were chosen instead of the big ones. In some cases, the algorithm was not run until its 200^{th} generation. The reasons for the different numbers of executions and the different generation numbers are also the long runtime of the algorithm on big inputs and the limited access of some computers that took part in the executions.

Table O.1. The difference between the optimum and the makespans obtained by the improved GA, on some test inputs of Brandimarte [3].

		Results of the GA				
name of the test instance	optimum	minimum gap	maximum gap	average gap	the average number of the generations	number of runs
mk01	40	2.5%	5%	4%	200	5
mk02	26	15.4%	23.1%	18.5%	200	5
mk03	204	0%	0%	0%	200	1
mk04	60	8.3%	13.3%	12%	200	5
mk05	173	2,3%	3.5%	2.9%	147.4	5
mk06	58	25.9%	25.9%	25.9%	200	1
mk07	144	9.7%	13.9%	11.1%	67.7	7
mk08	523	0.2%	0.2%	0.2%	200	1
mk09	307	21.2%	21.2%	21.2%	5	1
mk10	198	44.4%	44.4%	44.4%	5.5	2

Table O.2. The difference between the best known upper bounds and the makespans obtained by the improved GA, on some test inputs of Hurink et al. [1].

name of the test instance	the best known upper bound [166, 167]	Results of the GA minimum gap	maximum gap	average gap	the average number of the generations
Hurink - edata la01	609*	3.9%	7.2%	5.6%	200
Hurink - edata la02	655*	0.9%	4.9%	3.3%	200
Hurink - edata la03	550*	7.8%	10.4%	9.0%	200
Hurink - edata la04	568*	9.0%	13.6%	11.8%	200
Hurink - edata la05	503*	5.4%	7.4%	6.4%	200
Hurink - edata la06	833*	4.6%	6.1%	5.4%	200
Hurink - edata la07	762*	6.2%	8.4%	7.7%	200
Hurink - edata la08	845*	4.2%	6.0%	5.1%	200
Hurink - edata la09	878*	3.3%	5.3%	4.7%	200
Hurink - edata la11	1103	7.0%	9.0%	7.8%	198.4
Hurink - edata la12	960*	6.5%	8.4%	7.2%	12.5
Hurink - edata car1	6176*	4.3%	9.1%	7.1%	200
Hurink - edata car2	6327*	3.8%	6.2%	5.2%	200
Hurink - edata car3	6856*	5.3%	10.9%	8.1%	200
Hurink - edata car4	7789*	2.0%	3.9%	3.4%	200
Hurink - edata car5	7229*	5.0%	6.0%	5.4%	200
Hurink - edata car6	7990*	4.3%	9.2%	5.9%	200
Hurink - edata car7	6123*	0%	8.1%	5.0%	200
Hurink - edata car8	7689*	4.8%	7.9%	6.2%	200

*optimum

The results are obtained from 5 runs, except for the results of test instance la12, which are obtained from 4 runs.

Table O.3. The difference between the best known upper bounds and the makespans obtained by the improved GA, on some test inputs of Hurink et al. [1], and Barnes and Chambers [4].

name of the test instance	the best known upper bound [166, 167]	Results of the GA: minimum gap	maximum gap	average gap	the average number of the generations
Hurink - rdata la01	571	6.0%	7.4%	6.6%	200
Hurink - rdata la02	530	4.2%	7.2%	6.0%	200
Hurink - rdata la03	478	6.1%	8.8%	7.4%	200
Hurink - rdata la04	502*	5.4%	9.0%	7.4%	200
Hurink - rdata la05	457*	4.2%	7.2%	5.8%	200
Hurink - rdata la06	799*	2.1%	4.3%	3.2%	200
Hurink - rdata la07	750	3.5%	5.1%	4.3%	184.6
Hurink - rdata la10	804*	2.2%	3.7%	3.1%	200
Hurink - rdata la11	1071*	2.0%	3.7%	2.9%	163.2
Hurink - rdata la40	970	33.1%	35.1%	34.1%	8.5
Barnes - mt10c1	927*	11.8%	16.9%	14.2%	200
Barnes - mt10cc	908*	11.3%	15.0%	12.8%	200
Barnes - mt10x	918*	8.3%	14.2%	10.0%	200
Barnes - mt10xx	918*	9.5%	14.9%	11.5%	200
Barnes - mt10xxx	918*	12.2%	14.5%	13.1%	198
Barnes - mt10xy	905*	11.2%	14.3%	12.6%	98.5

*optimum

The results are obtained from 5 runs, except for the results of test instances la40 and mt10xy, which are obtained from 4 runs.

Appendix P

Clarke and Wright savings method

The Clarke and Wright savings method is an efficient, popular, and widespread method for creating the routes of a VRP problem. The proposed algorithm of my thesis can be applied for an arbitrary VRP solution. I chose the Clarke and Wright savings method for initial VRP solution creation because of its simplicity and wide usage in the VRP literature. Its main steps are as follows.

Step 1: Calculate the saving value $S(m_i, m_j) = L(d, m_i) + L(d, m_j) - L(m_i, m_j)$ for every pair (m_i, m_j) of customers.

Step 2: Order the saving values in descending order. This results in the "savings list". Start the processing of the saving list with its first element.

Step 3: For the saving value $S(m_i, m_j)$ under consideration, include link (m_i, m_j) in a route if no route constraints will be violated, and if:

- Either, neither m_i nor m_j has already been assigned to a route. In this case, a new route is created, including both m_i and m_j.
- Or, exactly one of the two customers (m_i or m_j) has already been included in an existing route, and that customer is not interior to that route - the concept "interior to a route" means that the customer is not adjacent to the depot -, in which case the link (m_i, m_j) is added to that same route.
- Or, both m_i and m_j have already been included in two different existing routes, and neither customer is interior to its route. In this case, the two routes are merged.

Step 4: If the end of the saving list has not been reached, process its next entry returning to Step 3; otherwise, stop. If any customer has not been assigned to a route, it must be assigned to its own route, which begins at the depot, visits the customer (only one), and returns to the depot.

Appendix Q

The flowchart of the algorithm of subsection 4.4.3 that determines the best route direction composition

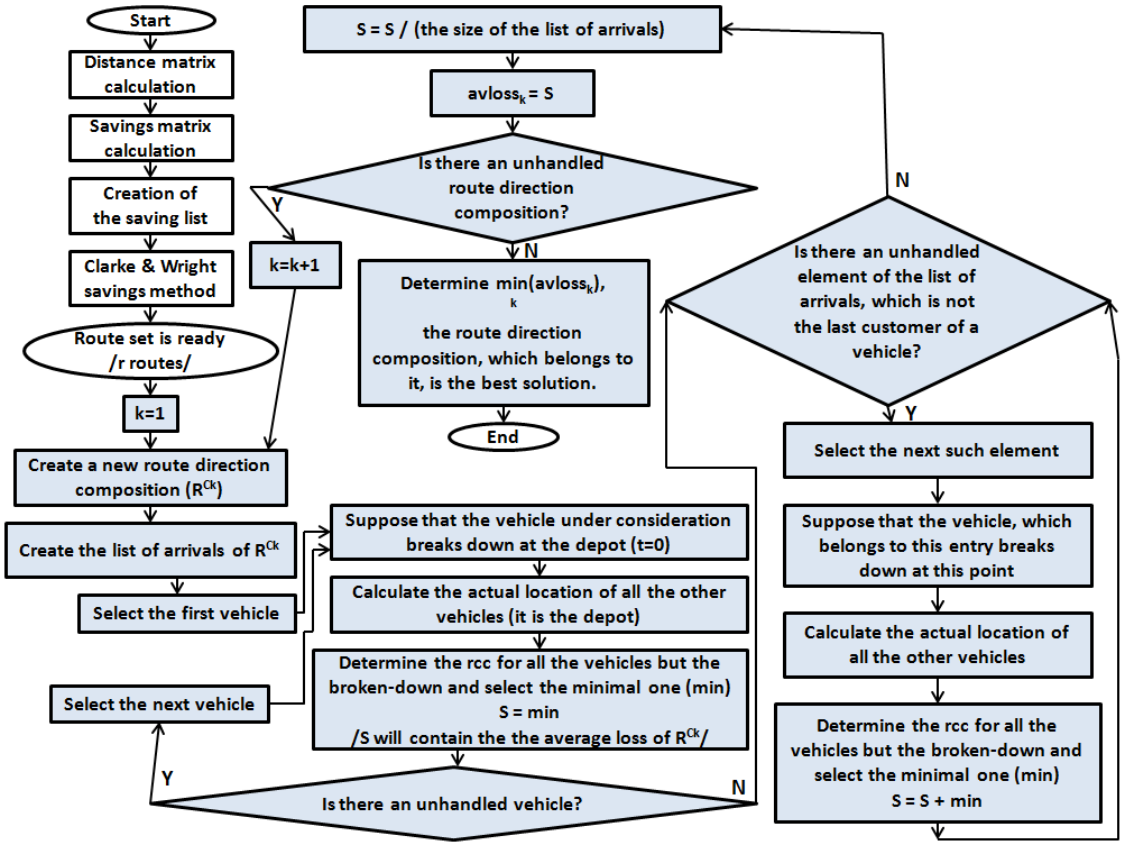


Figure Q.1. The flowchart of the algorithm that determines the best route direction composition for making a VRP solution to be fault-tolerant.

Appendix R

An example for the execution of the algorithm of subsection 4.4.4 that determines the helper vehicle and its modified route

Suppose that the route direction composition of Figure R.1 is started to be executed, and vehicle n_2 breaks down when it reaches its second customer ($m_{2,2}$), as Figure R.2 shows that. In the figures, the labels on the arrows show the lengths of the route segments, and the amount of the demand of each customer is highlighted in parentheses after the customer's ID. It can be seen that each vehicle has the same capacity (100).

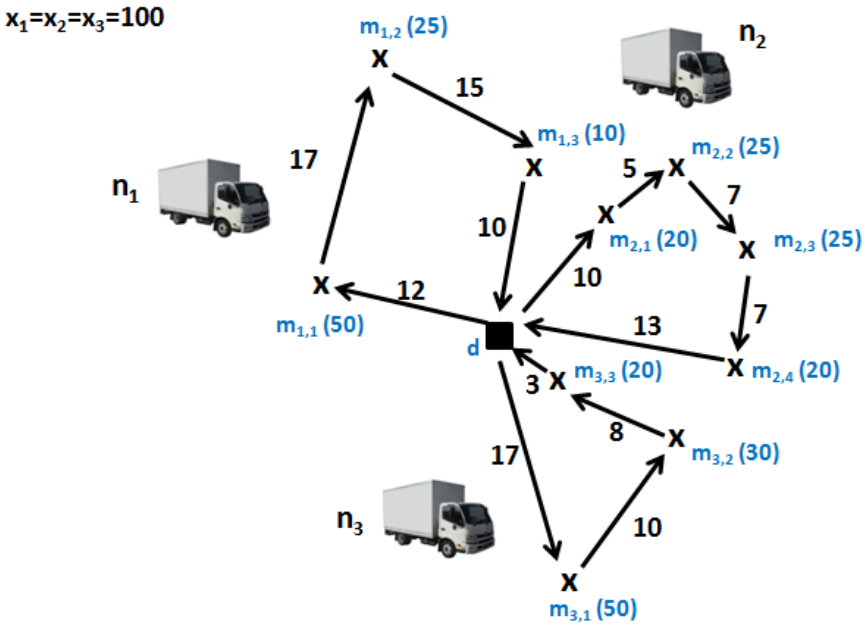


Figure R.1. The example route direction composition.

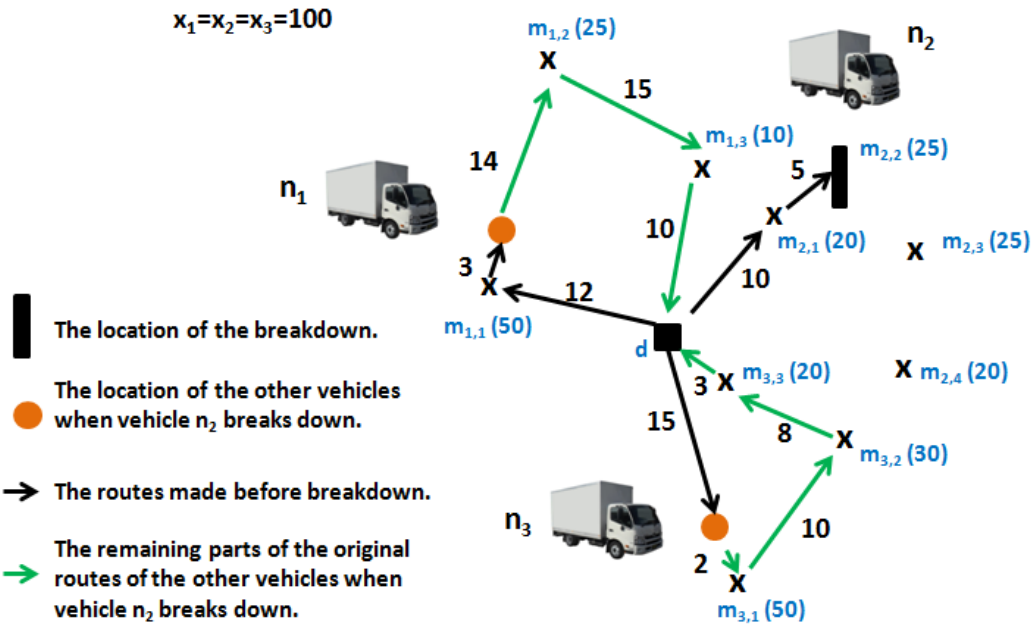


Figure R.2. The example route direction composition at the moment of the breakdown of vehicle n_2.

Executing Step C of the algorithm of subsection 4.4.4, the following calculations are made:
At the moment of the breakdown, all vehicles drove 15 units of distance, starting from the depot. As a result, the locations of vehicles n_1 and n_3 are highlighted by orange circles in Figure R.2. The set of the remaining customers of vehicle n_2 is $M_B^r = \{m_{2,3}, m_{2,4}\}$. Moreover, the remaining customers of vehicle n_1 are $m_{1,2}$ and $m_{1,3}$, and the remaining customers of vehicle n_3 are $m_{3,1}$, $m_{3,2}$, and $m_{3,3}$. Based on (4.20), The actual free capacity of vehicle n_1 is $100 - 25 - 10 = 65$, and the actual free capacity of vehicle n_3 is $100 - 50 - 30 - 20 = 0$.
Based on Step D of the algorithm of subsection 4.4.4, for

- vehicle n_1: $RF = \{q, m_{1,2}, m_{1,3}, m_{2,3}, m_{2,4}, d\}$, $A = 35$, $m_f = m_{2,3}$, $m_g = q$, and
- vehicle n_3: $RF = \{q, m_{3,1}, m_{3,2}, m_{2,4}, m_{2,3}, m_{3,3}, d\}$, $A = 100$, $m_f = m_{2,3}$, $m_g = m_{3,1}$.

If the helper vehicle carries such an amount of goods that is enough only for its own customers, the broken-down vehicle must be visited to get its remaining goods. By inserting the location of the broken-down vehicle (p) into the route of the helper vehicle - following the instructions of Step D -, the obtained route is:

- $RF = \{q, m_{1,2}, m_{1,3}, p, m_{2,3}, m_{2,4}, d\}$, if the helper vehicle is n_1, see Figure R.3.
- $RF = \{q, m_{3,1}, m_{3,2}, m_{2,4}, p, m_{2,3}, m_{3,3}, d\}$, if the helper vehicle is n_3, see Figure R.4.

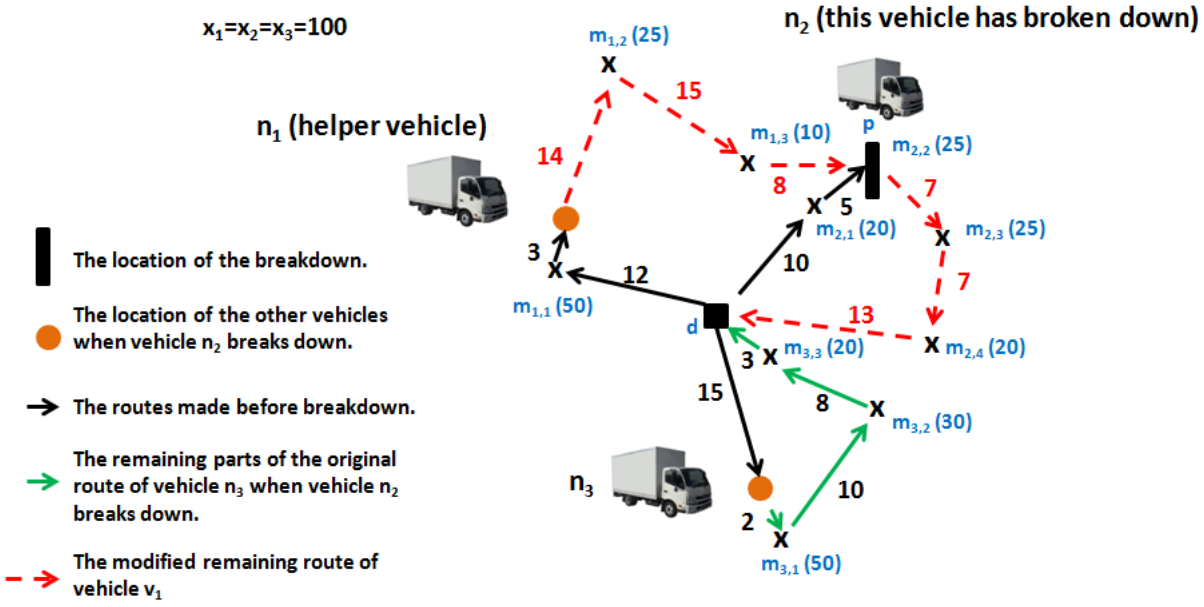


Figure R.3. The proposed execution if the helper vehicle is n_1.

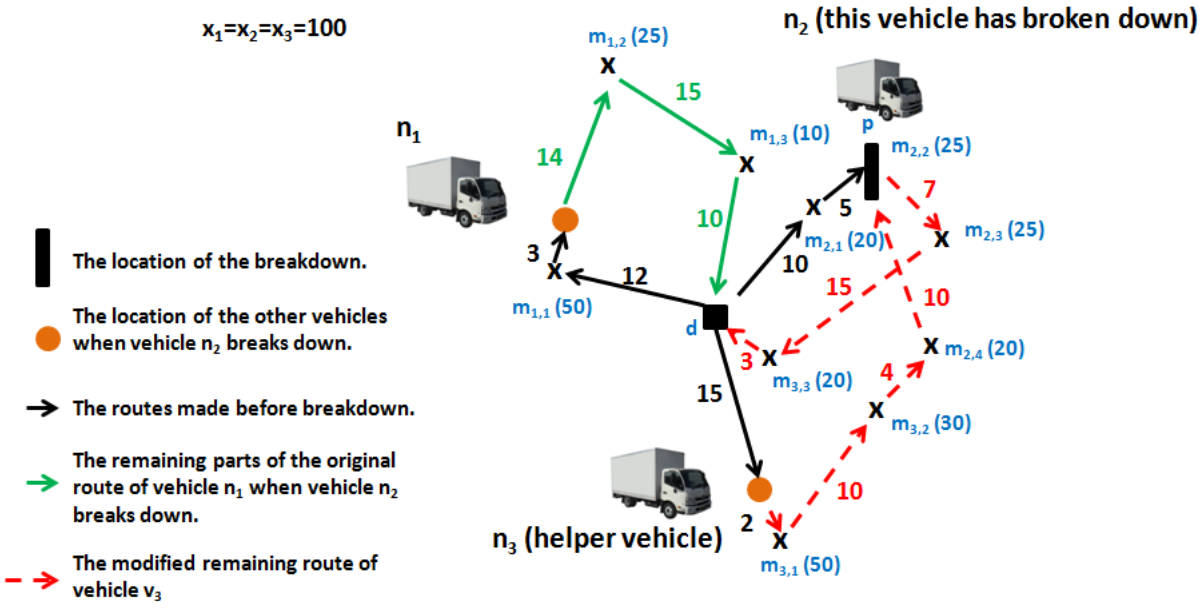


Figure R.4. The proposed execution if the helper vehicle is n_3.

Based on (4.17), *rcc* is

- $12+3+14+15+8+7+7+13-(12+17+15+10)=25$ if the helper vehicle is n_1.
- $15+2+10+4+10+7+15+3-(17+10+8+3)=28$ if the helper vehicle is n_3.

Finally, vehicle n_1 is chosen as the helper vehicle, and the modified routes after the breakdown are the routes of Figure R.3.

Appendix S

The detailed results of the proposed algorithm for the simple example of subsection 4.5.1

Vehicles that broke down and those that possibly can offer assistance are illustrated for each route direction composition at the points of route change cost calculation. These points are determined based on *Step* 7 of subsection 4.4.3. At each point, the minimum route change costs are highlighted. The minimum *rcc* values are then averaged, and the highlighted number shows the result of the algorithm: a route direction composition (number V.) with an averaged route change cost value of 53.82 units.
Without applying the proposed algorithm, vehicles start in an arbitrary direction on their route. The effectiveness of the method of this thesis can be well analyzed using the provided simple example. Table S.1 and Table S.2 represent all the possible route direction compositions of the simple example. Route change cost values show the extra cost that has to be paid in case of a vehicle breakdown. As the last line of the tables (averaged minimal route change cost) shows, the proposed method determines the route direction composition with the minimum average extra cost in case of a vehicle breakdown. It is important to emphasize "average", as it is not always the selected route direction composition that is the optimal one. Supposing vehicle breakdowns at different locations (see the rows of the tables - the vehicle breakdown is supposed to happen at the noted point in the list of arrivals), the different route direction compositions differ in the cost of the recourse action. E.g., supposing that the vehicle breakdown happens at the 5th element of the list of arrivals, the route change cost varies between 22.59 and 61.25, even if the helper vehicle with the minimal extra cost is selected. It can also be seen that it is not the proposed solution (route direction composition variation number *V.*) that provides the best response if a vehicle breakdown occurs at the 5th element of the list of arrivals, but route direction composition variation number *IV.* However, in other cases, the proposed method performs better, and on average, it outperforms all the other route direction compositions. Table 4.6 of subsection 4.5.1 summarizes the efficiency of the directed route set selected by the proposed method compared to the other possible solutions.

Table S.1. Averaged route change cost values for the route direction compositions of the simple example of subsection 4.5.1 - Part 1.

	route direction composition variation number															
	I.			II.			III.			IV.						
point of the list of arrivals	broken-down vehicle	helper vehicle	route change cost	broken-down vehicle	helper vehicle	route change cost	broken-down vehicle	helper vehicle	route change cost	broken-down vehicle	helper vehicle	route change cost	broken-down vehicle	helper vehicle	route change cost	
0a in the depot	n_1	n_2	78.34	n_1	n_2	78.34	n_1	n_2	78.34	n_1	n_2	78.34	n_2	n_1	78.34	
		n_3	**77.03**		n_3	**77.03**		n_3	**77.03**		n_3	**77.03**	n_3	n_1	**77.03**	
0b in the depot	n_2	n_1	76.34	n_2	n_1	76.34	n_2	n_1	76.34	n_2	n_1	76.34	n_1	n_2	76.34	
		n_3	**68,75**		n_3	**68,75**		n_3	**68,75**		n_3	**68,75**	n_3	n_2	**68,75**	
0c in the depot	n_3	n_1	71.03	n_3	n_1	71.03	n_3	n_1	71.03	n_3	n_1	71.03	n_1	n_3	71.03	
		n_2	**64.75**		n_2	**64.75**		n_2	**64.75**		n_2	**64.75**	n_2	n_3	**64.75**	
1.	n_3	n_1	78.37	n_1	n_2	76.76	n_2	n_1	93.00	n_2	n_1	87.21	n_1	n_2	87.21	
		n_2	**51.97**		n_3	**64.60**		n_3	**64.19**		n_3	**64.19**	n_3	n_1	**64.19**	
2.	n_2	n_1	93.00	n_3	n_1	68.88	n_3	n_1	78.37	n_1	n_2	**62.73**	n_2	n_3	**62.73**	
		n_3	**88.30**		n_2	**51.97**		n_2	**61.61**		n_3	64.60	n_3	n_2	64.60	
3.	n_1	n_2	88.06	n_2	n_1	**85.40**	n_2	n_1	83.00	n_3	n_1	68.88	n_1	n_2	68.88	
		n_3	**64.66**		n_3	88.29		n_3	**59.82**		n_2	**61.61**	n_2	n_3	**61.61**	
4.	n_2	n_1	65.04	n_1	n_2	69.95	n_1	n_2	79.94	n_2	n_1	75.40	n_1	n_3	75.40	
		n_3	**61.25**		n_3	**22.59**		n_3	**64.66**		n_3	**59.82**	n_3	n_2	**59.82**	
5.	n_1	n_2	65.44	n_2	n_1	65.93	n_2	n_1	**28.04**	n_1	n_2	53.91	n_2	n_3	53.91	
		n_3	**22.95**		n_3	**61.25**		n_3	42.56		n_3	**22.59**	n_3	n_1	**22.59**	
6.	n_3	-	-	n_3	-	-	n_1	n_2	48.67	n_2	n_1	**34.57**	n_1	n_3	**34.57**	
		-	-		-	-		n_3	**22.95**		n_3	42.56	n_3	n_2	42.56	
7.	n_2	n_1	21.84	n_2	n_1	22.72	n_3	-	-	n_3	-	-	-	-	-	
		n_3	**18.13**		n_3	**18.13**		-	-		-	-	-	-	-	
8.	n_1	-	-	n_1	-	-	n_2	-	-	n_2	-	-	-	-	-	
		-	-		-	-		-	-		-	-	-	-	-	
9.	n_2	-	-	n_2	-	-	n_1	-	-	n_1	-	-	-	-	-	
		-	-		-	-		-	-		-	-	-	-	-	
averaged minimal route change cost			57.53			57.16			56.87						57.34	

Table S.2. Averaged route change cost values for the route direction compositions of the simple example of subsection 4.5.1 - Part 2.

route direction composition variation number																
	V.				VI.				VII.				VIII.			
point of the list of arrivals	broken-down vehicle	helper vehicle	route change cost		broken-down vehicle	helper vehicle	route change cost		broken-down vehicle	helper vehicle	route change cost		broken-down vehicle	helper vehicle	route change cost	
0a in the depot	n_1	n_2	78.34		n_1	n_2	78.34		n_1	n_2	78.34		n_1	n_2	78.34	
		n_3	**77.03**			n_3	**77.03**			n_3	**77.03**			n_3	**77.03**	
0b in the depot	n_2	n_1	76.34		n_2	n_1	76.34		n_2	n_1	76.34		n_2	n_1	76.34	
		n_3	**68,75**			n_3	**68,75**			n_3	**68,75**			n_3	**68,75**	
0c in the depot	n_3	n_1	71.03		n_3	n_1	71.03		n_3	n_1	71.03		n_3	n_1	71.03	
		n_2	**64.75**			n_2	**64.75**			n_2	**64.75**			n_2	**64.75**	
1.	n_2	n_1	93.00		n_1	n_2	76.76		n_2	n_1	93.00		n_2	n_1	87.21	
		n_3	**80.03**			n_3	**76.36**			n_3	**70.62**			n_3	**70.62**	
2.	n_1	n_2	88.06		n_2	n_1	85.40		n_2	n_1	83.00		n_1	n_2	**62.73**	
		n_3	**76.36**			n_3	**80.03**			n_3	**51.71**			n_3	76.36	
3.	n_3	n_1	36.36		n_3	n_1	31.22		n_1	n_2	79.94		n_2	n_1	75.40	
		n_2	**24.54**			n_2	**24.54**			n_3	**76.36**			n_3	**51.71**	
4.	n_2	n_1	65.04		n_1	n_2	69.95		n_3	n_1	36.36		n_3	n_1	**31.22**	
		n_3	**53.02**			n_3	**38.99**			n_2	**34.43**			n_2	34.43	
5.	n_1	n_2	65.44		n_2	n_1	65.93		n_2	n_1	**28.04**		n_1	n_2	53.91	
		n_3	**30.08**			n_3	**53.02**			n_3	39.70			n_3	**38.99**	
6.	n_3	-	-		n_3	-	-		n_1	n_2	48.67		n_2	n_1	**34.57**	
		-	-			-	-			n_3	**30.08**			n_3	39.70	
7.	n_2	n_1	21.84		n_2	n_1	22.72		n_3	-	-		n_3	-	-	
		n_3	**9.80**			n_3	**9.80**			-	-			-	-	
8.	n_1	-	-		n_1	-	-		n_2	-	-		n_2	-	-	
		-	-			-	-			-	-			-	-	
9.	n_2	-	-		n_2	-	-		n_1	-	-		n_1	-	-	
		-	-			-	-			-	-			-	-	
averaged minimal route change cost			**53.82**				54.81				55.75				55.60	

Appendix T

The histograms of averaged minimal route change costs of the different Solomon instance groups' route direction compositions

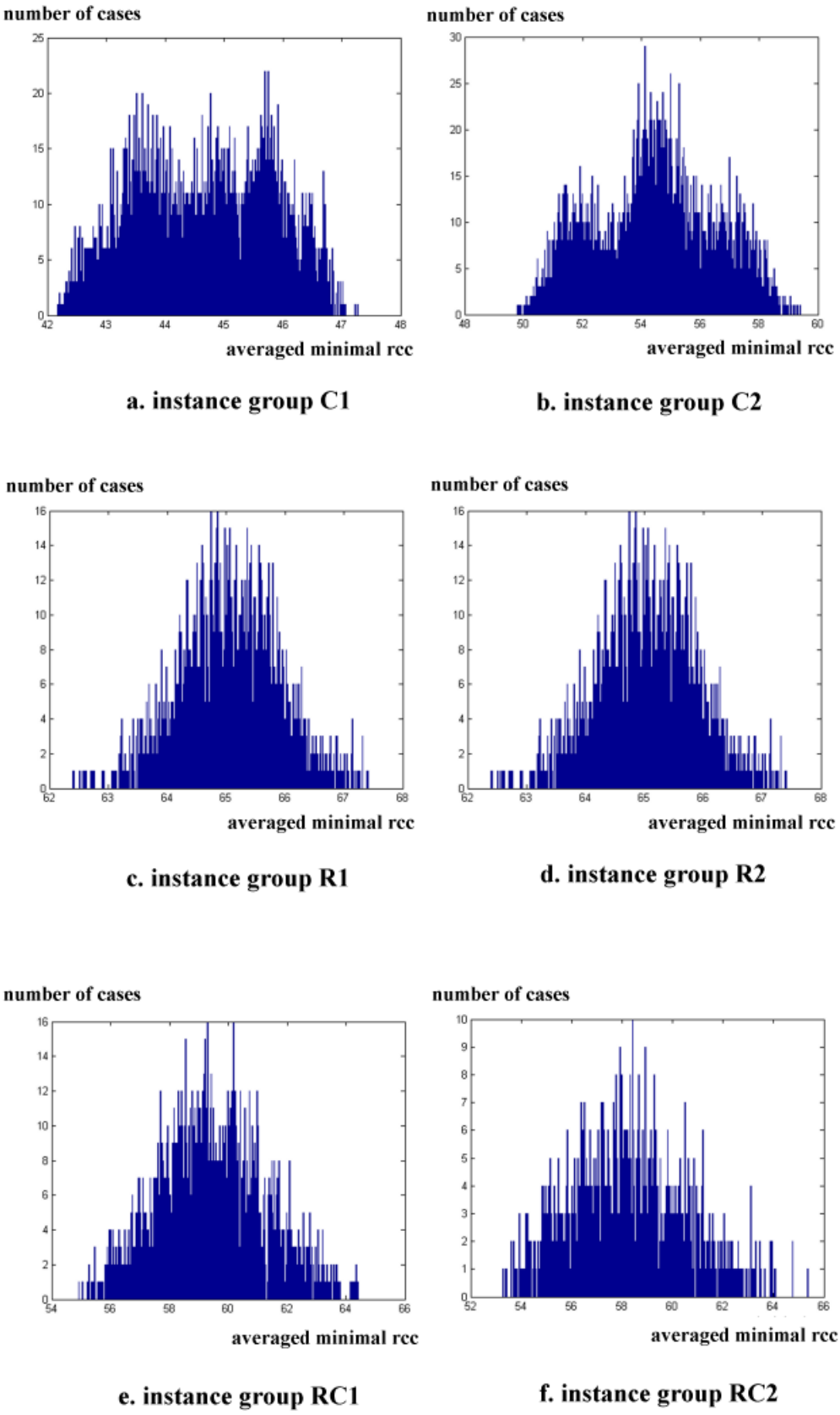


Figure T.1. Histograms of averaged minimal route change costs (*rcc*, see (4.19)) of the different Solomon instance groups' route direction compositions.

Bibliography

[1] J. Hurink, B. Jurisch, and M. Thole. Tabu search for the job-shop scheduling problem with multi-purpose machines. *Operations-Research-Spektrum*, 15:205–215, 1994.

[2] P. Pongcharoen, C. Hicks, P.M. Braiden, and D.J. Stewardson. Determining optimum genetic algorithm parameters for scheduling the manufacturing and assembly of complex products. *International Journal of Production Economics*, 78(3):311–322, 2002.

[3] P. Brandimarte. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 41:157–183, 1993.

[4] J.W. Barnes and J.B. Chambers. Flexible job shop scheduling by tabu search. Research Report ORP96-09, Graduate Program in Operations Research and Industrial Engineering, The University of Texas at Austin, 1996.

[5] L.F. Gelders and L.N. Van Wassenhove. Production planning: a review. *European Journal of Operational Research (Fourth EURO III Special Issue)*, 7(2):101–110, 1981.

[6] B. Chen, C.N. Potts, and G.J. Woeginger. *A review of machine scheduling: Complexity, algorithms and approximability, Handbook of Combinatorial Optimization (Vol 3)*. Kluwer Academic Publishers, 1998.

[7] P. Brucker. *Scheduling Algorithms*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 3rd edition, 2001.

[8] C.T. Maravelias and C. Sung. Integration of production planning and scheduling: Overview, challenges and opportunities. *Computers & Chemical Engineering*, 33(12):1919–1930, 2009.

[9] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable Secure Computing*, 1(1):11–33, 2004.

[10] W. Hohl, E. Michel, and A. Pataricza. Hardware support for error detection in multiprocessor systems — a case study. *Microprocessors and Microsystems*, 17(4):201–206, 1993.

[11] J. Arlat, K. Kanoun, and J. Laprie. Dependability modeling and evaluation of software fault-tolerant systems. *IEEE Transactions on Computers*, 39(4):504–513, 1990.

[12] D. Poola, K. Ramamohanarao, and R. Buyya. Chapter 15. A taxonomy and survey of fault-tolerant workflow management systems in cloud and distributed computing environments. In *Software architecture for big data and the cloud*, pages 285–320, 2017.

[13] K.C. Sevcik. Application scheduling and processor allocation in multiprogrammed parallel processing systems. *Performance Evaluation*, 19(2):107–140, 1994.

[14] J. Błażewicz, P. Dell'Olmo, and M. Drozdowski. Scheduling of client-server applications. *International Transactions in Operational Research*, 6(4):345–363, 1999.

[15] M. Abouelela and M. El-Darieby. Scheduling big data applications within advance reservation framework in optical grids. *Applied Soft Computing*, 38:1049–1059, 2016.

[16] T. Anttalainen and V. Jaaskelainen. *Introduction to Communication Networks*. Artech House, Boston, London, 2015.

[17] T. Dulai. A játékelmélet lehetséges szerepe a távközlésben. *Híradástechnika*, 2:33–36, 2007.

[18] T. Dulai. Non-cooperative games for self-adaptive telecommunication protocols. *Periodica Polytechnica*, 49(3-4):223–237, 2005.

[19] T. Dulai, Sz. Jaskó, D. Muhi, and K. Tarnay. Self-adaptive test system. In *Lecture Notes in Computer Science, Self-Adaptive Software (IWSAS 2003, Arlington (VA), USA, June 2003)*, 2003.

[20] B. Dezfouli, M. Radi, K. Whitehouse, S. Abd Razak, and T. Hwee-Pink. Dicsa: Distributed and concurrent link scheduling algorithm for data gathering in wireless sensor networks. *Ad Hoc Networks*, 25:54–71, 2015.

[21] N. Safaei and A.K.S. Jardine. Aircraft routing with generalized maintenance constraints. *Omega*, 80(C):111–122, 2018.

[22] A.S. Tanenbaum and D.J. Wetherall. *Computer Networks*. Prentice Hall Press, USA, 5th edition, 2010.

[23] T.E. Amah, M. Kamat, W. Moreira, K. Abu Bakar, S. Mandala, and M.A. Batista. Towards next-generation routing protocols for pocket switched networks. *Journal of Network and Computer Applications*, 70:51–88, 2016.

[24] V. Vandeginste and K. Piessens. Pipeline design for a least-cost router application for CO2 transport in the CO2 sequestration cycle. *International Journal of Greenhouse Gas Control*, 2(4):571–581, 2008.

[25] R.J. Nayaka and R.C. Biradar. Data aggregation and routing scheme for smart city public utility services using WSN. In *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pages 1–8, 2017.

[26] J. Heil, K. Hoffmann, and U. Buscher. Railway crew scheduling: Models, methods and applications. *European Journal of Operational Research*, 283(2):405–425, 2020.

[27] G. Pundoor and Z.-L. Chen. Scheduling a production-distribution system to optimize the tradeoff between delivery tardiness and distribution cost. *Naval Research Logistics*, 52:571–589, 2005.

[28] Y. Park. An integrated approach for production and distribution planning in supply chain management. *International Journal of Production Research*, 43:1205–1224, 2005.

[29] W. Zhong, Gy. Dósa, and Z. Tan. On the machine scheduling problem with job delivery coordination. *European Journal of Operational Research*, 182:1057–1072, 2007.

[30] M. Hajiaghaei-Keshteli, M. Aminnayeri, and S.M.T. Fatemi Ghomi. Integrated scheduling of production and rail transportation. *Comput. Ind. Eng.*, 74:240–256, 2014.

[31] F. Zesch and B. Hellingrath. Integrated production-distribution planning an optimization model for mixed-model assembly lines. In *Proceedings of the 22nd Annual NOFOMA Conference*, pages 1–17, 2010.

[32] F. Marandi and S.H. Zegordi. Integrated production and distribution scheduling for perishable products. *Scientia Iranica*, 24(4):2105–2118, 2017.

[33] L. Liu and S. Liu. Integrated production and distribution problem of perishable products with a minimum total order weighted delivery time. *Mathematics*, 8(2):146, 2020.

[34] Z.-L. Chen. Integrated production and distribution operations. *In: D. Simchi-Levi and S.D. Wu and Z.-J. Shen (eds.), Handbook of Quantitative Supply Chain Analysis: Modeling in the E-Business Era*, pages 711–745, 2004.

[35] Z.-L. Chen. Integrated production and outbound distribution scheduling: Review and extensions. *Operations Research*, 58:130–148, 2010.

[36] L.-L. Fu, M.A. Aloulou, and C. Artigues. Integrated production and outbound distribution scheduling problems with job release dates and deadline. *Journal of Scheduling*, 21:443–460, 2018.

[37] J. Behnamian and S.M.T. Fatemi Ghomi. A survey of multi-factory scheduling. *Journal of Intelligent Manufacturing*, 27(1):231–249, 2016.

[38] V. Abdollahzadeh, I. Nakhaikamalabadi, S. Hajimolana, and S. Zegordi. A multifactory integrated production and distribution scheduling problem with parallel machines and immediate shipments solved by improved whale optimization algorithm. *Complexity*, 2018:1–21, 2018.

[39] S. Kesen and T. Bektaş. Integrated production scheduling and distribution planning with time windows: Optimization models and algorithms. *International Series in Operations Research and Management Science*, pages 231–252, 2019.

[40] Y.-C. Chang and C.-Y. Lee. Machine scheduling with job delivery coordination. *European Journal of Operational Research*, 158:470–487, 2004.

[41] M. Mazdeh, M. Sarhadi, and K. Hindi. A branch-and-bound algorithm for single-machine scheduling with batch delivery and job release times. *Computers & Operations Research*, 35:1099–1111, 2008.

[42] A. Memari, R. Ahmad, A.A. Rahim, and A. Hassan. Optimizing a just-in-time logistics network problem under fuzzy supply and demand: two parameter-tuned metaheuristics algorithms. *Neural Computing and Applications*, pages 1–13, 2017.

[43] L. Dayou, Y. Pu, and Y. Ji. Development of a multiobjective GA for advanced planning and scheduling problem. *The International Journal of Advanced Manufacturing Technology*, 42:974–992, 2008.

[44] C.-J. Liao, Y.-L. Tsai, and C.-W. Chao. An ant colony optimization algorithm for setup coordination in a two-stage production system. *Applied Soft Computing*, 11(8):4521–4529, 2011.

[45] F. Li, L. Zhou, G. Xu, H. Lu, K. Wang, and S.-B. Tsai. An empirical study on solving an integrated production and distribution problem with a hybrid strategy. *PLoS ONE*, 13:e0206806, 2018.

[46] Z. Ferenczi. *Operációkutatás*. Novadat, Győr, 2006.

[47] P. Laborie, J. Rogerie, P. Shaw, and P. Vilím. IBM ILOG CP optimizer for scheduling: 20+ years of scheduling with constraints at IBM/ILOG. *Constraints*, 23:210–250, 2018.

[48] S.I. Gass and A.A. Assad. *An Annotated Timeline of Operations Research: An Informal History (International Series in Operations Research & Management Science)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[49] A. Nagar, J. Haddock, and S. Heragu. Multiple and bicriteria scheduling: A literature survey. *European Journal of Operational Research*, 81(1):88–104, 1995.

[50] P. Wojakowski and D. Warżołek. The classification of scheduling problems under production uncertainty. *Research in Logistics and Production*, 4(3):245–256, 2014.

[51] T. Kis. Job-shop scheduling with processing alternatives. *European Journal of Operational Research*, 151:307–332, 2003.

[52] J.R. Jackson. Scheduling a production line to minimize maximum tardiness. Research report 43, University of California, Los Angeles, USA, 1955.

[53] W.E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.

[54] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[55] M.L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 3rd edition, 2008.

[56] R. Ruiz and J.A.V. Rodríguez. The hybrid flow shop scheduling problem. *European Journal of Operational Research*, 205:1–18, 2010.

[57] V. Yaurima, A. Tchernykh, F. Villalobos-Rodríguez, and R. Salomon-Torres. Hybrid flow shop with unrelated machines, setup time, and work in progress buffers for bi-objective optimization of tortilla manufacturing. *Algorithms*, 11:68, 2018.

[58] F.T.S. Chan, T.C. Wong, and L.Y. Chan. Lot streaming for product assembly in job shop environment. *Robotics and Computer-Integrated Manufacturing*, 24(3):321–331, 2008.

[59] P. Brucker and R. Schlie. Job-shop scheduling with multi-purpose machines. *Computing*, 45(4):369–375, 1991.

[60] J. Xie, L. Gao, K. Peng, X. Li, and H. Li. Review on flexible job shop scheduling. *IET Collaborative Intelligent Manufacturing*, 1(3):67–77, 2019.

[61] P. Brucker and J. Neyer. Tabu-search for the multi-mode job-shop problem. *Operations-Research-Spektrum*, 20:21–28, 1998.

[62] P.-H. Chiang and C.-C. Torng. A production planning and optimisation of multi-mode job shop scheduling problem for an avionics manufacturing plant. *International Journal of Manufacturing Technology and Management*, 30:179–195, 2016.

[63] S. Dauzère-Pérès, W. Roux, and J.B. Lasserre. Multi-resource shop scheduling with resource flexibility. *European Journal of Operational Research*, 107(2):289–305, 1998.

[64] S. Muthuraman and V.P. Venkatesan. A comprehensive study on hybrid metaheuristic approaches used for solving combinatorial optimization problems. *2017 World Congress on Computing and Communication Technologies (WCCCT)*, pages 185–190, 2017.

[65] A.M. Shaheen, S.R. Spea, S.M. Farrag, and M.A. Abido. A review of meta-heuristic algorithms for reactive power planning problem. *Ain Shams Engineering Journal*, 9(2):215–231, 2018.

[66] A. Kumar and S. Bawa. A comparative review of meta-heuristic approaches to optimize the SLA violation costs for dynamic execution of cloud services. *Soft computing*, 24:3909–3922, 2020.

[67] I. Wegener and R. Pruim. *Complexity Theory: Exploring the Limits of Efficient Algorithms*. Springer-Verlag, Berlin, Heidelberg, 2005.

[68] M.R. Garey, D.S. Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129, 1976.

[69] Y. Pochet and L.A. Wolsey. *Production Planning by Mixed Integer Programming*. Springer Publishing Company, Incorporated, New York, USA, 2006.

[70] T. Sawik. *Production Planning and Scheduling in Flexible Assembly Systems*. Springer-Verlag Berlin, Heidelberg, New York, 1999.

[71] F. Seeanner. *Multi-Stage Simultaneous Lot-Sizing and Scheduling*. Springer Fachmedien, Wiesbaden, 2013.

[72] S. Voß and D.L. Woodruff. *Introduction to Computational Optimization Models for Production Planning in a Supply Chain*. Springer-Verlag Berlin, Heidelberg, 2006.

[73] C. Almeder, D. Klabjan, R. Traxler, and B. Almada-Lobo. Lead time considerations for the multi-level capacitated lot-sizing problem. *European Journal of Operational Research*, 241(3):727–738, 2015.

[74] J.R. Correa and A.S. Schulz. Single-Machine Scheduling with Precedence Constraints. *Mathematics of Operations Research*, 30(4):1005–1021, 2005.

[75] C. Gicquel and M. Minoux. Multi-product valid inequalities for the discrete lot-sizing and scheduling problem. *Computers & Operations Research*, 54:12–20, 2015.

[76] C. Gicquel, M. Minoux, and Y. Dallery. On the discrete lot-sizing and scheduling problem with sequence-dependent changeover times. *Operations Research Letters*, 37:32–36, 2009.

[77] K. Sörensen and F.W. Glover. Metaheuristics. *In: S.I. Gass and M.C. Fu (eds.), Encyclopedia of Operations Research and Management Science*, pages 960–970, 2013.

[78] F. Glover. Tabu search-Part I. *ORSA Journal on computing*, 1(3):190–206, 1989.

[79] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[80] M. Dorigo and T. Stützle. Ant colony optimization: Overview and recent advances. *Handbook of Metaheuristics*, 146:227–263, 2010.

[81] J.H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* Ann Arbor: University of Michigan Press, 1975.

[82] E. Alba and J.F. Chicano. Software project management with GAs. *Information Sciences*, 177(11):2380–2401, 2007.

[83] R. Jans and Z. Degraeve. Meta-heuristics for dynamic lot sizing: A review and comparison of solution approaches. *European Journal of Operational Research*, 177:1855–1875, 2007.

[84] S. Terrazas-Moreno, I.E. Grossmann, and J.M. Wassick. A Mixed-Integer Linear Programming model for optimizing the scheduling and assignment of tank farm operations. *Industrial & Engineering Chemistry Research*, 51:6441–6454, 2012.

[85] C.K. Chang, H.-Y. Jiang, Y. Di, D. Zhu, and Y. Ge. Time-line based model for software project scheduling with genetic algorithms. *Information and Software Technology*, 50(11):1142–1154, 2008.

[86] C.K. Chang, M.J. Christensen, and T. Zhang. Genetic algorithms for project management. *Annals of Software Engineering*, 11(1):107–139, 2001.

[87] A. Sadegheih. Scheduling problem using genetic algorithm, simulated annealing and the effects of parameter values on GA performance. *Applied Mathematical Modelling*, 30:147–154, 2006.

[88] G. Zhang, L. Gao, and Y. Shi. An effective genetic algorithm for the flexible job-shop scheduling problem. *Expert Syst. Appl.*, 38(4):3563–3573, 2011.

[89] X. Li and L. Gao. An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem. *International Journal of Production Economics*, 174:93–110, 2016.

[90] T. Dulai, Gy. Dósa, and Á. Werner-Stark. Multi-project optimization with multi-functional resources by a genetic scheduling algorithm. *Acta Polytechnica Hungarica* ***IF: 0.909***, 15:101–119, 2018.

[91] J.R. Montoya-Torres, J.L. Franco, S.N. Isaza, H.F. Jiménez, and N. Herazo-Padilla. A literature review on the vehicle routing problem with multiple depots. *Computers & Industrial Engineering*, 79:115–129, 2015.

[92] H.N. Psaraftis, M. Wen, and C.A. Kontovas. Dynamic vehicle routing problems: Three decades and counting. *Networks*, 67(1):3–31, 2016.

[93] R. Lahyani, M. Khemakhem, and F. Semet. Rich vehicle routing problems: From a taxonomy to a definition. *European Journal of Operational Research*, 241(1):1–14, 2015.

[94] B. Eksioglu, A.V. Vural, and A. Reisman. The vehicle routing problem: A taxonomic review. *Computers & Industrial Engineering*, 57:1472–1483, 2009.

[95] X. Wang and E.A. Wasil. On the road to better routes: Five decades of published research on the vehicle routing problem. *Networks*, 77(1):66–87, 2021.

[96] D.L. Applegate, R.E. Bixby, V. Chvatal, and W.J. Cook. *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton University Press, Princeton, NJ, USA, 2007.

[97] G.B. Dantzig, D.R. Fulkerson, and S.M. Johnson. Solution of a large-scale traveling salesman problem. *Operations Research*, 2:393–410, 1954.

[98] A. Király and J. Abonyi. A novel approach to solve multiple traveling salesmen problem by genetic algorithm. *Studies in Computational Intelligence*, 313:141–151, 2010.

[99] G.B. Dantzig and R.H. Ramser. The truck dispatching problem. *Management Science*, 6:80–91, 1959.

[100] J. Homberger and H. Gehring. Two evolutionary metaheuristics for the vehicle routing problem with time windows. *INFOR: Information Systems and Operational Research*, 37:297–318, 1999.

[101] K.C. Tan, L.H. Lee, and K. Ou. Artificial intelligence heuristics in solving vehicle routing problems with time window constraints. *Engineering Applications of Artificial Intelligence*, 14(6):825–837, 2001.

[102] J. Tang, Z. Pan, R. Fung, and H.C.W. Lau. Vehicle routing problem with fuzzy time windows. *Fuzzy Sets and Systems*, 160:683–695, 2009.

[103] Z. Dan, L. Cai, and L. Zheng. Improved multi-agent system for the vehicle routing problem with time windows. *Tsinghua Science & Technology*, 14:407–412, 2009.

[104] W. Ho, G.T.S. Ho, P. Ji, and H.C.W. Lau. A hybrid genetic algorithm for the multi-depot vehicle routing problem. *Engineering Applications of Artificial Intelligence*, 21:548–557, 2008.

[105] B. Crevier, J.-F. Cordeau, and G. Laporte. The multi-depot vehicle routing problem with inter-depot routes. *European Journal of Operational Research*, 176(2):756–773, 2007.

[106] E. Angelelli and M.G. Speranza. The periodic vehicle routing problem with intermediate facilities. *European Journal of Operational Research*, 137:233–247, 2002.

[107] G. Mosheiov. Vehicle routing with pick-up and delivery: Tour-partitioning heuristics. *Computers & Industrial Engineering*, 34:669–684, 1998.

[108] M. Gendreau, F. Guertin, J.-Y. Potvin, and R. Séguin. Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. *Transportation Research Part C: Emerging Technologies*, 14:157–174, 2006.

[109] I. Borgulya. An algorithm for the capacitated vehicle routing problem with route balancing. *Central European Journal of Operations Research*, 16:331–343, 2008.

[110] P. Kilby, P. Prosser, and P. Shaw. Dynamic VRPs: A study of scenarios. Technical Report APES-06-1998, University of Strathclyde, U.K., 1998.

[111] A. Larsen. *The Dynamic Vehicle Routing Problem, PhD Dissertation at the Department of Mathematical Modelling, The Technical University of Denmark.* PhD thesis, 2000.

[112] V. Pillac, M. Gendreau, C. Guéret, and A. Medaglia. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225:1–11, 2013.

[113] M. Gendreau, G. Laporte, and R. Séguin. Stochastic vehicle routing. *European Journal of Operational Research*, 88(1):3–12, 1996.

[114] Z. Shen, F. Ordonez, and M.M. Dessouky. The stochastic vehicle routing problem for minimum unmet demand. *Springer Series on Optimization and Its Applications*, 30:349–371, 2009.

[115] L. Bianchi, M. Birattari, M. Chiarandini, M. Manfrin, M. Mastrolilli, L. Paquete, O. Rossi-Doria, and T. Schiavinotto. Hybrid metaheuristics for the vehicle routing problem with stochastic demands. *Journal of Mathematical Modelling and Algorithms*, 5:91–110, 2006.

[116] J. Goodson, J. Ohlmann, and B. Thomas. Rollout policies for dynamic solutions to the multivehicle routing problem with stochastic demand and duration limits. *Operations Research*, 61:138–154, 2013.

[117] N. Secomandi and F. Margot. Reoptimization approaches for the vehicle-routing problem with stochastic demands. *Operations Research*, 57:214–230, 2009.

[118] Army science and technology master plan. Technical Report 199903, Department of the Army, Washington, DC., 1998.

[119] Hassan Noura, Didier Theilliol, Jean-Christophe Ponsart, and Abbas Chamseddine. *Fault-Tolerant Control Systems: Design and Practical Applications.* 2009.

[120] C. Cortés, A. Núñez, and D. Saez. Hybrid adaptive predictive control for a dynamic pickup and delivery problem including traffic congestion. *International Journal of Adaptive Control and Signal Processing*, 22:103–123, 2008.

[121] C. Novoa, R. Berger, J. Linderoth, and R. Storer. A set-partitioning-based model for the stochastic vehicle routing problem. Technical Report 06T-008, Lehigh University, USA, 2006.

[122] A. Novaes, E. Bez, and P. Burin. Fault detection in dynamic vehicle routing operations. In *Dynamics in logistcs: Third international conference, LDIC 2012*, pages 13–34, 2013.

[123] R. Sprenger and L. Mönch. A decision support system for cooperative transportation planning: Design, implementation, and performance assessment. *Expert Systems with Applications*, 41:5125–5138, 2014.

[124] P. Auer, Gy. Dósa, T. Dulai, A. Fügenschuh, P. Näser, R. Ortner, and Á. Werner-Stark. A new heuristic and an exact approach for a production planning problem. *Central European Journal of Operations Research* ***IF: 2.000***, pages 1–35, 2020, online published.

[125] T. Dulai, Á. Werner-Stark, and K. Hangos. Algorithm for directing cooperative vehicles of a vehicle routing problem for improving fault-tolerance. *Optimization and Engineering* ***IF: 1.352***, 19(2):239–270, 2017.

[126] M. Ridha. The role of heuristic methods as a decision-making tool in aggregate production planning. *International Journal of Business Administration*, 6:68–76, 2015.

[127] A. Newell and H. Simon. *Human problem solving*. Prentice-Hall, 1972.

[128] J.F. Dillenburg. Techniques for improving the efficiency of heuristic search, PhD Thesis, University of Illinois at Chicago, 1993.

[129] J. Puchinger and G.R. Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. *In: J. Mira and J.R. Álvarez (eds.), Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, pages 41–53, 2005.

[130] Fault Tolerant Systems Research Group. Process modeling. Lecture notes, Budapest University of Technology and Economics, 2020.

[131] R. Conradi, C. Liu, and L. Jaccheri. Process modeling paradigms: An evaluation. In *Proceedings of the 7th International Software Process Workshop, Communication and Coordination in the Software Process*, pages 51–53, 1991.

[132] J. Herrmann, J.M. Proth, and N. Sauer. Heuristics for unrelated machine scheduling with precedence constraints. *European Journal of Operational Research*, 102(3):528–537, 1997.

[133] P.L. Rocha, M.G. Ravetti, G.R. Mateus, and P.M. Pardalos. Exact algorithms for a scheduling problem with unrelated parallel machines and sequence and machine-dependent setup times. *Computers and Operations Research*, 35:1250–1264, 2008.

[134] V.S.A. Kumar, M.V. Marathe, S. Parthasarathy, and A. Srinivasan. Scheduling on Unrelated Machines Under Tree-Like Precedence Constraints. *In: C. Chekuri et al. (eds.), APPROX and RANDOM 2005, LNCS*, 3624:146–157, 2005.

[135] C. Liu and S. Yang. A heuristic serial schedule algorithm for unrelated parallel machine scheduling with precedence constraints. *Journal of Software*, 6(6):1146–1153, 2011.

[136] M.A. Hassan, I. Kacem, S. Martin, and I.M. Osman. Unrelated Parallel Machine Scheduling Problem with Precedence Constraints: Polyhedral Analysis and Branch-and-Cut. *In: R. Cerulli, S. Fujishige, A. Mahjoub (eds.), Combinatorial Optimization. ISCO 2016. Lecture Notes in Computer Science*, 9849:308–319, 2016.

[137] T. Dulai, Gy. Dósa, Á. Werner-Stark, Gy. Ábrahám, and Zs. Nagy. A hybridization technique for improving a scheduling heuristic in mass production. *In: Algorithms for Intelligent Systems, Proceedings of International Conference on Communication and Computational Technologies, ICCCT 2021*, Accepted, 2021.

[138] L.D. Chambers. *Practical Handbook of Genetic Algorithms.* CRC Press, Inc., USA, 1995.

[139] J.C. Tay and D. Wibowo. An effective chromosome representation for evolving flexible job shop schedules. *In: K. Deb (ed.), Genetic and Evolutionary Computation – GECCO 2004. Lecture Notes in Computer Science*, 3103:210–221, 2004.

[140] L. Gao, C.Y. Peng, C. Zhou, and P.G. Li. Solving flexible job-shop scheduling problem using general particle swarm optimization. In *Proceedings of the* 36*th CIE Conference on Computers & Industrial Engineering*, pages 3018–3027, 2006.

[141] I. Kacem, S. Hammadi, and P. Borne. Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. *IEEE Transactions on Systems Man and Cybernetics Part C (Applications and Reviews)*, 32:1–13, 2002.

[142] F. Pezzella, G. Morganti, and G. Ciaschetti. A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research*, 35(10):3202–3212, 2008.

[143] P. Kora and P. Yadlapalli. Crossover operators in genetic algorithms: A review. *International Journal of Computer Applications*, 162:34–36, 2017.

[144] S. Rahnamayan, H. Tizhoosh, and M.M.A. Salama. A novel population initialization method for accelerating evolutionary algorithms. *Computers & Mathematics with Applications*, 53:1605–1614, 2007.

[145] L. Al-Kanj, W.B. Powell, and B. Bouzaïene-Ayari. The information-collecting vehicle routing problem: Stochastic optimization for emergency storm response. *CoRR*, abs/1605.05711, 2016.

[146] L. Tansini, M. Urquhart, and O. Viera. Comparing assignment algorithms for the Multi-Depot VRP. Technical Report 01-08, University of the Republic Uruguay (UDELAR), 2001.

[147] A. Rybickova, A. Karásková, and D. Mocková. Use of genetic algorithms for spare parts distribution system. *Engineering Optimization IV - Proceedings of the 4th International Conference on Engineering Optimization, ENGOPT 2014*, pages 639–644, 2014.

[148] A. dos Santos, A. Barradas, S. Labidi, and N. Costa. A comparison between optimization heuristics of the best path problem applied to S-route. In *Proceedings of GEOProcessing 2013 - The Fifth International Conference on Advanced Geographic Information Systems, Applications, and Services, Athens, Greece*, pages 172–177, 2013.

[149] J. Renaud, G. Laporte, and F. Boctor. A tabu search heuristic for the multi-depot vehicle routing problem. *Computers & Operations Research*, 23:229–235, 1996.

[150] T.-H. Wu, C. Low, and J.-W. Bai. Heuristic solutions to multi-depot location-routing problems. *Computers & Operations Research*, 29:1393–1415, 2002.

[151] E. Alba and B. Dorronsoro. Solving the vehicle routing problem by using cellular genetic algorithms. *Evolutionary Computation in Combinatorial Optimization (EvoCOP), Lecture Notes in Computer Science*, 3004:11–20, 2004.

[152] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[153] T.M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1st edition, 1997.

[154] H. Nazif and L.S. Lee. Optimized crossover genetic algorithm for vehicle routing problem with time windows. *American Journal of Applied Sciences*, 7:95–101, 2010.

[155] V. Ciesielski and P. Scerri. Compound optimisation. Solving transport and routing problems with a multi-chromosome genetic algorithm. In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, pages 365–370, 1998.

[156] J. Berger, M. Sassi, and M. Salois. A hybrid genetic algorithm for the vehicle routing problem with time windows and itinerary constraints. In *Proceedings of the Genetic and Evolutionary Computation Conference, Orlando, USA*, pages 44–51, 1999.

[157] O. Bräysy. Genetic algorithms for the vehicle routing problem with time windows. *Arpakannus, Special Issue on Bioinformatics and Genetic Algorithms*, pages 33–38, 2001.

[158] Y.-B. Park. A hybrid genetic algorithm for the vehicle scheduling problem with due times and time deadlines. *International Journal of Production Economics*, 73(2):175–188, 2001.

[159] J.-Y. Potvin and S. Bengio. The vehicle routing problem with time windows - Part II: Genetic Search. *INFORMS Journal on Computing*, 8:165–172, 2002.

[160] S. Salhi and R.J. Petch. A GA based heuristic for the vehicle routing problem with multiple trips. *Journal of Mathematical Modelling and Algorithms*, 6:591–613, 2007.

[161] S.R. Thangiah. Vehicle routing with time windows using genetic algorithms. *In: L. Chambers (ed.), Practical Handbook of Genetic Algorithms: New Frontiers*, pages 253–277, 1995.

[162] S.N. Kumar and R. Panneerselvam. A survey on the vehicle routing problem and its variants. *Intelligent Information Management*, 4:66–74, 2012.

[163] M.M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35:254–266, 1987.

[164] T. Dulai, Á. Werner-Stark, and K.M. Hangos. Immediate event-aware model and algorithm of a general scheduler. *Hungarian Journal of Industry and Chemistry*, 41(1):27–34, 2013.

[165] T. Dulai and Á. Werner-Stark. A database-oriented workflow scheduler with historical data and resource substitution possibilities. In *Proceedings of the International Conference on Operations Research and Enterprise Systems (ICORES, Lisbon, Portugal, January 2015)*, pages 325–330, 2015.

[166] D. Behnke and M.J. Geiger. Test instances for the flexible job shop scheduling problem with work centers. Research Report RR-12-01-01, Helmut-Schmidt-Universität, 2012.

[167] G.A. Kasapidis, D.C. Paraskevopoulos, P.P. Repoussis, and C.D. Tarantilis. Flexible job shop scheduling problems with complex precedence graphs. 2020, preprint.