CZESŁAW SMUTNICKI

OPTIMIZATION AND CONTROL IN JUST-IN-TIME MANUFACTURING SYSTEMS

Prace Naukowe Instytutu Cybernetyki Technicznej
Politechniki Wrocławskiej 97

Seria:
Monografie 25

Czesław Smutnicki

Optimization and control in just-in-time manufacturing systems

Oficyna Wydawnicza Politechniki Wrocławskiej · Wrocław 1997

Recenzenci
Eugeniusz TOCZYŁOWSKI
Konrad WALA

Weryfikacja językowa
Ewa SOBESTO

OFICYNA WYDAWNICZA POLITECHNIKI WROCŁAWSKIEJ
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław

ISSN 0324-9786

Drukarnia Oficyny Wydawniczej Politechniki Wrocławskiej. Zam. nr 336/97.

Contents

Prace Naukowe Instytutu Cybernetyki Technicznej
Nr 97 Politechniki Wrocławskiej Nr 97
Monografie Nr 25 1997

dyskretny proces produkcyjny,
wytwarzanie na żądanie, szeregowanie,
optymalizacja, algorytm

Czesław SMUTNICKI [1]

OPTIMIZATION AND CONTROL IN JUST-IN-TIME MANUFACTURING SYSTEMS

Just-in-Time (JIT) is a new, very efficient strategy of control of discrete production systems. Its analysis requires essentially new mathematical models and new efficient methods of solution of optimization problems. At the begining, review of research directions in the field of JIT systems has been presented and that of discrete optimisation methods with particular attention paid to those based on elements of Artificial Intelligence. In the essential part of the book, there are analysed some scheduling problems which can be applied to modelling JIT systems. There are considered, among others, the problem of scheduling of a production cell with various measures of on-time parts supply, the problem of minimising the work-in-process in the flow-line and the problems of intercell buffering in the flexible flow-line. For each considered problem there are provided mathematical models, some properties and solution algorithms. From among solution methods, the significant attention is paid to local search techniques and very efficient tabu search method which has been developed essentially. The numerical properties of the algorithms are examined analytically and experimentally.

[1]Institute of Engineering Cybernetics, Technical University of Wrocław, Wybrzeże Wyspiańskiego 27, 50-372 Wrocław

Chapter 1

Introduction

Just-in-time (JIT) is the strategy of manufacturing and inventory control whereby the required items are produced at the time and in the quantities needed. JIT stimulates a new flexible thinking in planning, controlling and performing activities, stock controlling, cash controlling and has significant influence on the reduction of manufacturing costs. In practice, its effects are much more significant by simultaneous improving the overall performance of the whole organisation. Generally, JIT can be seen as an approach that combines several objectives, commonly considered as conflicting, namely decrease in cost, increase in product quality, manufacturing flexibility and delivery performances.

One of the most spectacular implementations of JIT manufacturing, performed on the Q-Flex accelerometer product line at Soundstrand Data Control's Instrument Systems Division, provided already in the first year of application the following benefits: quality up by 50%, rework down by 66%, scrap costs cut by 60%, cycle to produce a unit reduced by 90%, money tied up in work-in-process down by 80%, unit hours decreased by 20%, overtime virtually eliminated, Scanzon [277]. All this with a 20% improvement in total capacity and no additional floor space required. Many other famous implementations have been provided, as an example Xerox plant, where the JIT strategy introduced in 1983 reduced the time of holding inventory 1.5 times with respect to MRP inventory control (dated from 1979), this in turn reduced 2.6 times inventory with respect to the older order point system, Flapper et al. [71]. Other well-known names appear frequently in the context of JIT successful implementations: Intel, Motorola, Texas Instruments, IBM, Westinghouse, General Electric.

The source of JIT philosophy one can find in Japanese manufacturing techniques from the end of 70's and at the beginning of 80's, see Toyota production systems, Schonenberger [278]. Since these works and the important addition concerning zero inventories, Hall [133], a considerable amount of results has been

reported in this approach. The greatest number of papers published is observed close to the end of 80's and at the beginning of 90's, hence the JIT philosophy is quite young. The significance of JIT strategy is clearly shown in the survey of the state of its developement and implementation in the USA in 1990, Gilbert [80], where 31 characteristic elements of the JIT technique were checked on a representative sample. 20–30% of manufacturing systems have been found with appropriate JIT implementation programs well developed, and next 20–40% – with programs just starting. All JIT characteristic elements constitute a high innovation in traditional manufacturing systems, thus are particulary recommended for obsolete manufacturing enviroments. In fact, JIT systems come near the attention of many practitioners since they offer essentially new benefits as well as profitable attributes in the aggressive, competitive manufacturing environment[1], particularly in flexible manufacturing systems (FMS) and computer–integrated manufacturing (CIM), where this strategy can easily be implemented. One can say that the future belongs to manufacturing systems controlled by JIT strategy.

On the other hand, most of the papers dealing with JIT systems are classified as case studies based on experience of the authors (some practical aspects). The number of mathematical models and algorithms reported on the JIT manufacturing systems is rather small. Quite often the proposed models either simplify the reality excessively, or become inadequate because of too general assumptions, unacceptable in practice. Moreover, formal mathematical models are still too complex (or of too high dimension) to solve them by *standard* tools of discrete optimisation, queuing theory, control theory, etc. Solving the stated theoretical problems of optimisation or control by using a *reasonable* computer resources (storage- and time- requirements) is usually very hard because of *dimension curse* and *NP–hardness* of the problems. Up to now, only simulation studies perform well; however in most of the complex cases, a huge amount of time is required. Therefore there is room for applying sophisticated, powerful tools of operations research (OR), management science (MS), control theory (CT), optimisation theory (OT), scheduling theory (ST) and queuing theory (QT) to represent JIT manufacturing systems.

The development of JIT strategy coincides well with some research directions arisen recently in the deterministic sequencing theory, as an example problems of due date requirements, nonregular penalty functions, flow lines, multi-purpose flow lines, no-store and zero-wait policy, etc. In this paper, there is presented an approach, having its origin in classic deterministic sequencing theory, which can be applied in modelling and solving some optimisation and control problems in JIT manufacturing systems. Modern discrete optimisation methods with ele-

[1]The immediate result of the market economy.

ments of artificial intelligence (AI) constitute a base for this approach. It should be emphasised, that not only *formulation of the problems* and *solution methods* are important in the considered research area. Equally valid are also the *rationality* and *efficiency* of proposed algorithms, which must be adjusted to act in the real control systems and/or decision support systems. The proposed approach is derived partially from the author's experiences gained during studies on the following subjects and research projects:

(a) Theory, models, algorithms and software for sequencing problems, within a Research Program "Numerical methods of optimisation and control" [2], [98], [99], [100], [101], [102], [103], [104], [105], [106], [107],

(b) Algorithms for complex optimisation problems in manufacturing systems, within a Research Program "Theory of control and optimisation of continuous dynamic systems and discrete processes" [3], [109], [110], [111], [112], [215], [216], [218], [220], [221], [292], [293], [295], [296], [297], [298], [299], [300], [113]

(c) Decision support systems, within an International Research Project "Resource Constrained Project Scheduling Problem: An International Exercise in DSS Development" held under the auspices of the International Institute of Applied System Analysis (IIASA), Laxenburg, Austria [4], [217], [222], [223], [227], [294], [304]

(d) Worst-case analysis as a method of performance evaluation for approximation algorithms [5]. [219], [224], [225], [226], [228], [229], [301], [303], [306]

(e) Artificial intelligence and approximation algorithms for scheduling in manufacturing systems [6], [230], [231], [232], [302], [305], [307], [308]

The author also wish to thank all the persons and organisations which supported this work. Special thanks are addressed to my family for the patience, the State Commitee – for funds for the research [7], and Professor Fred Glover from Colorado University at Boulder – for valuable suggestions referring to TS technique.

[2] Research supported by National Research Program R.I.21.01, ASO 4.2.3-4.2.5, 1982-85

[3] Research supported by National Research Program RP I.02, 1986-90

[4] System developed by the two-person team with the author obtained the highest rank among all participated teams, 1987-91.

[5] A part of this research was supported by the State Committee, Grant 307609101, 1991-92.

[6] A part of this research was supported by Contract IIASA-PAN, 1991-92, and by the State Committee, Grant 3P40301906 1994-95.

[7] A part of this research is supported by the State Committee, Grants 8T11A01712 (1997) and 8T11A02112 (1997-99).

Chapter 2

Optimisation and control in JIT systems

The manufacturing system is a kind of highly complex system presented as a *black box* having row materials (substrates) and resources (machines, personnel, energy, etc.) at the input, and final products at the output. The basic aim of this system is to provide required final products in required quantities and required time moments. These requirements are either set arbitrary (firm contracts) or derived from a forecasting of environment needs [1]. From some point of view we would like to obtain the programmed (assumed in advance) output values in time. To reach this goal one must determine the values of input variables in time, having means of *control variables*. There are also available some additional control variables which influence flows of materials and the distribution of resources inside the system. In practice, by controlling the system we can usually get only output the *close* to that assumed because of random system breakdowns or input disturbances. Moreover, there exist many ways of achieving this goal. Hence, there is needed an additional measure of the system activity. Beside the " distance" between the assumed and realy outputs, a supporting criterion of the system behaviour has frequently been introduced. It is based chiefly on *production cost* or *net present value* as the most common and universal ones. Many methods of these cost calculations have been proposed and developed.

Manufacturing systems can be analysed by taking advantage of either deterministic or stochastic models. If the system reliability is small (frequent machine breakdowns, man faults, poor product quality, etc. [2]) and the input disturbance high, the system tends towards stochastic one. (On the other hand, instead of

[1] Although the quality of forecasting can have an influence on the system behaviour, we will assume hereafter *known* output requirements obtained by forecasting made *outside* the system.

[2] All these random breakdowns are associated with the presence of man in the machine world.

controling such a poor system one can propose to apply beforehand a firm reconstruction program, that results in a significant improvement of the system performance.)

Quite often, in order to decrease the influence of uncertainty on the assumed goal, inputs as well as outputs are buffered. Unfortunately, there is a tendency in practice to increase the buffer stock with increasing uncertainty, that implies the higher production cost followed from the superfluous inventory. These inventory costs, linked with the costs of machine repairing and worker idle time, clearly show faults which will eliminate in the near future this system from the manufacturing environment because of too high production cost.

Modern manufacturing systems, by releasing human factor (CIM systems, FMS), admit a smaller number of random events and thus can be described with a high accuracy by deterministic models. Moreover, many production structures investigated theoretically well, as an example the production cell with (non)uniform parallel machines, flow line, multi-machine flow line, flexible job shop, etc., are immediately applicable in these systems. In this context, there is a room for applying models as well as algorithms from deterministic scheduling theory as a convenient tool for analysis and control. Note that better (more adequate) models allow us to use more tight "aggressive" control. In terms of the above, JIT philosophy resolves problems of the production control which provide assumed output with sufficiently high accuracy at the minimal production cost.

There are also other significant features which essentially distinguish JIT systems from the surrounding manufacturing environment. The main one refers to the structure of the control system. In manufacturing systems consisting of several production units (PU), there are at least three mutually disjoined control structures (see Fig. 2.1) : (a) single-level centralised control (CC), (b) hierarchical multi-level control (HC) with a selected special sub-class two-level (master-slave) control, where parameters for local controllers (LC) are set by superordinary control system (SC), (c) decentralised control based on co-operated local controllers (LC). Other types of the control can be obtained due to a combination of these three, for example by admitting in (b) the co-operation between successive local controllers, like in (c). In this figure, information flows are marked by thin lines, control flows by thick lines, whereas material/product flows by wide arrows. Although we have assumed, for the simplicity of presentation, that the production system has a flow line structure, nevertheless this classification remains valid for other types of production flow, as an example for rooted tree architecture (assembly systems). One can say that enumerated types of control are well known and have been performed for years in the manufacturing systems with a man as an "intelligent controller". However, followed by a high automation of the technology processes, there is a common tendency to replace human controller

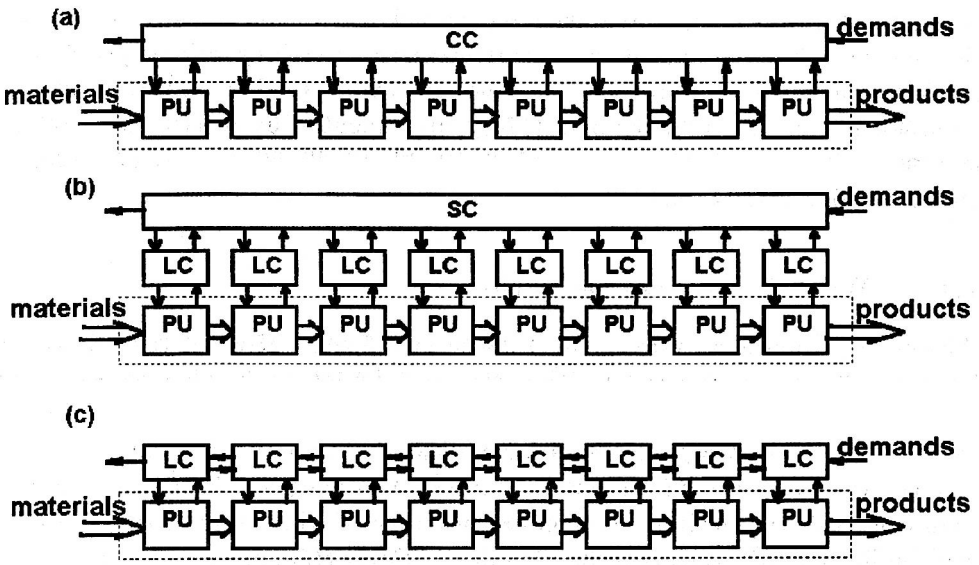

Figure 2.1: Control structures of manufacturing systems

by automatic (programmed) one, and to simplify the control algorithm. Whereas the control in traditional manufacturing systems has been kept within structures (a) or (b), JIT philosophy prefers control of the type (c) with the algorithm for LC as simply as possible and as fast as possible.

Other differences refer to the technique of finding the control. Classic manufacturing control techniques treat the data of the problem (processing times, transport times, set-up times, etc.) as fixed, whereas JIT treats them as at least *modifiable*. These modification can be made under certain conditions, with some limited scope, at some additional cost, etc. One can say that JIT is an "active" strategy with respect to the old one, which is considered as "passive".

There have been found more characteristic elements of JIT systems. Gilbert [80] provides a list containing 31 such elements among which one can find as e.g. pull strategy, set-up and lead times reduction, zero-buffer and no-wait policy, dedicated production lines, synchronised scheduling, zero deviation from schedule, regularity in end-product scheduling, Kanbans.

2.1 Classification of problems

During the past few years, a number of papers have dealt with concept, design, justification, control, modelling, implementation, management and operational

aspects of JIT manufacturing systems. Although more than thirty characteristic elements of JIT systems have been recognised and distinguished, [80], most of the papers can be reckoned in several research directions briefly presented in the subsequent subsections, see excellent surveys of Golhar and Stamm [93] [3], Gunasekaran et al. [122], Kubiak [174], supplemented by works since 1993, we mention only some of these papers [5, 6, 12, 42, 48, 49, 50, 60, 67, 121, 123, 129, 165, 185, 201, 214, 244, 340].

2.1.1 Work in process reduction

Wastes in production systems can take several forms. Most of them have pure inventory character associated with cycle stock, buffer stock, whereas others refer to time, material handling, motion, and product quality, Hall [133]. To reduce these wastes a lot of organisational propositions have been suggested. Nevertheless, the work in process (WIP) reduction problem has been discussed more often than others, particularly in the context of impact of the buffer storage on a production line output, Buzacott [33], Ignall and Silver [150] or impact of production parameters (set-up time/cost, delivery times, etc.) on the inventory cost, Funk [76]. The former models assume either stochastic processing times or random machine breakdowns, and obviously a limited buffer storage. Note that in JIT environments no buffer stock is allowed, and hence zero inventory (ZI) is highly desirable. The notion ZI is often used in the context of Japanese inventory system, JIT system, and Kanban system. ZI with a simultaneous set-up reduction offer a new promising direction of improving production efficiency. However, extreme reducing the set-up costs does not necessarily reduce inventory, Zangwill [341].

The reduction of inventory costs in JIT systems and differences between WIP- and JIT-type inventory control policies have been discussed among others by Funk [76] and Sipper and Shapira [290]. Other problems related to WIP in JIT associated with AGV systems have been considered by Occena and Yokota [233].

2.1.2 Kanban controlled systems

This type of control can be classified as mixed (c) and (b) control structure from Fig. 2.1. Kanban philosophy has been introduced to JIT manufacturing to keep a tight control over inventory and to reduce the inventory level to a minimum. Kanbans derive originally from Toyota, and now are identified as one of the characteristic elements of JIT systems.

[3] This survey is made for a huge number of 860 papers.

In the Kanban system, items are put into containers and different types of items are held in different containers. Once a container is full, a Kanban is attached to it. "Kanban" in the Japanese, refers to a card or tag. Some of the authors call it also a Kanban ticket or pass. Kanbans are destined for multiple use, their number is limited and used to control the system. The full container is valid only with attached Kanban. A Kanban usually carries the following data: (A) item specification (name, description, cardinality in the container, etc.), (B) container specification (type, capacity), (C) identification number, (D) technological order (preceding stage, succeeding stage).

Each stage of the Kanban system can be seen as a production process (fabrication, subassembly, delivery, purchase) with an inventory point located at the output of this stage. Produced items fill a container, which next is stored in the inventory point with a Kanban attached to it. Each stage acts using as inputs the items stored at the inventory point of the immediate predecessor stage. When a full container is taken from the inventory point of the current stage (to be used by the production process of the immediate successor stage) the Kanban originally attached to the container is detached and left at this inventory point. All detached Kanbans are collected and periodically sent back to the production process of the current stage to provide a new processing order usually fulfiled by the FIFO rule.

The total number of Kanbans circulating between production process and the inventory point of each stage is fixed over time, and used to control the maximum inventory build-up of this stage. It is clear that the production schedule (production order) of the final stage is automatically transmitted back to all the upstream stages. Since each detached Kanban automatically becomes a new order, no other inventory control is needed. Similarly, each machine breakdown automatically stops unwanted inventory of preceding stages. One can say that upstream are flexible and self-operated.

Kanbans play at least five roles in the manufacturing process. The container is a kind of buffer designed to attenuate random production events, e.g. machine breakdowns. The size of the container explicitly allots production lot size. The number of Kanban cards limits the number of production lots. The order of receiving Kanban cards provides the production order. Kanban cards are also used to ensure communication loop (with feedback) between successive production units and to stop unwanted production if it is necessary.

There are a few theoretical optimisation problems considered in Kanban systems. First, determination of the number of Kanbans and a container lot size at each stage required to satisfy the customer's demand, e.g. Philicom et al. [247], Monden [119], Chaudhury and Whinstom [41], Huang et al. [146], or other papers on this subject in the bibliography. These models assume deterministic or

stochastic demands, stochastic machine breakdowns, single or multiple item production, assembly type production, single or multiple machine stages. Formulated models are either too unrealistic, or too complicated for practical applications. As to now, only simulation studies prove to be satisfactory. Second, the choice between Kanban system and traditional lot-sized model of production that takes into account a trade-off between the planning horizon and a fluctuation of demands. Note that if the demand fluctuates markedly over the planning horizon the Kanban system, because of the fixed number of Kanbans, forces the facility to carry more inventory than one really needed. On the other hand, these trade-offs are significantly situation dependent.

2.1.3 Assembly line leveling

The primal source of the JIT philosophy follows from assembly systems consisting of a single or multiple cooperated lines with a tree-like structure [4]. Consequently, there exists in JIT field the research sub-direction associated with assembly lines, see the papers [120, 152, 172, 173, 199, 200, 202] and the survey of Kubiak [174].

These are the problems specific by means of considered models and scheduling objectives. To satisfy the customer demands for a variety of products without holding large inventories, one proposes to keep the quantity of each product produced per unit time as close as possible to the demand for that product per unit time. This type of problems is called the *leveling* of the schedule of mixed-model assembly line, Hall [133], Monden [207]. There are several objectives used in these models, as e.g. minimising variation of the rate at which different products are produced on the line (PRV), minimising variation of the rate at which the line supply different products on its output (ORV), smoothing the workload on each workstation on the line, etc. Special cases are distinguished, as e.g. when outputs require the same/various number of parts, when requirements for each different product are approximately equal or distinct, Kubiak [174]. Solution algorithms proposed emply either mixed-integer programing models or scheduling models with appropriately defined penalty function.

2.1.4 Push, pull and squeeze shop floor control

The push control policy is the oldest and commonly used one, also in many highly industrialised Western countries. Production commands are given per period in advance, based on forecast demands and predicated product flows in the inventory system. The most adequate characteristic of such a system is associated with a

[4] Some further implementations of JIT strategy refers to non-assembly systems with a more general structure, as e.g. job-shop Gravel and Price [119].

job (product) flow. The job (material) is released to the first stage of the system, in turn this stage pushes the work in process (WIP) to the following stage, and so forth until the product reaches the final stage. One can say that this type of control pushes product through the system in accordance with an external *master schedule*. It represents the type (a) of control in Fig. 2.1.

The pull control policy prevents the superfluous inventory. The production material goes through the system only on demand, each stage pulls semi-products from the preceding stages, so as to meet precisely customer requirements, see (c) in Fig. 2.1. JIT system is a pull system supplying product only on demand, Kim [161]. Nevertheless, the pull philosophy need not be used in all manufacturing systems. Although there are several techniques of achieving a pull type control system, the most of the known implementations refer to the use of Kanbans, quite often to balance an assembly line.

The squeeze control policy concentrates only on those machines in the production process which are bottleneck, i.e. where production is squeezed. Based on the observation that normal fluctuations in production always have a negative effect on co-operated streams, bottleneck machines are protected with an inventory buffer so as to minimise the impact of this fluctuation on the downstream–dependent processes. Next, bottleneck machines are scheduled in accordance to the final product schedule, whereas schedule of the remain machines is adjusted to those bottlenecks.

It has been shown by Sarker and Fitzsimmons [269] that pure push and pure pull strategies have different advantages and disadvantages. Therefore, a compromise, called hybrid push/pull control strategies, seems to be the most suitable.

2.1.5 MRP/JIT/OPT inventory control

MRP (Material Requirements Planning), MRPII (Manufacturing Resource Planning), JIT and OPT (Optimised Production Technology) are considered as mutually exclusive systems of the inventory control. They correspond to the control techniques *push*, *push*, *pull*, and *squeeze*, respectively, mentioned in the previous subsection, Ramsey et al. [258].

MRP, MRPII and OPT form a centralised system of controlling and planning the inventory. Demands for final products are translated into requirements for parts, materials, resources, etc. addressed to particular components of the system. The production is completely planned and is run in accordance with a *master production schedule* developed by balancing requirements and resources. This schedule is being performed with the use of monitoring and controlling of the production progress, called the *shop floor control*. The cost of the production is observed, compared with standard costs for all manufacturing activities and

provides production account. MRP is an ideal technique for planning and controlling production activities, but is less useful for reducing costs and lead-times as well as for improving quality.

Generally, application of each of these systems is specific, however there are some suggestions derived from the research and implementations. In a repetitive manufacturing environment with a small variety of products (e.g. assembly systems), Kanban is found as the most effective method of control. There, it drastically reduces inventory and simplifies control and planning. However, in a job-shop environment with a large variety of products, MRP is preferable. In the complex production environments, OPT is preferred to MRP and JIT.

2.1.6 Set-up reduction

One of the important aspect of the JIT systems and ZI concept is to reduce the set-up cost. In this way one can save the capacity of a facility, to reduce lot-sizes, to improve the overall utilisation and efficiency of production system, Porteues [310, 248, 249]. Japanese apply set-up reduction programmes by using quick clamping devices, jigs, fixtures, etc., to reduce set-up time. It can also be shown that in some cases larger batches may also influence a product quality. Besides the problems mentioned, one can also state the problem of finding optimal set-up costs under various conditions and the problem of finding optimal set-up reduction program assuming investment means and profits. Karmarkar [159] has proved that the set-up reduction program can significantly improve the overall utilisation and efficiency of a manufacturing system. These models can be used while evaluating different strategies for investment in the set-up reduction program for achieving a JIT manufacturing system.

2.1.7 JIT purchasing/suppliers systems

Basic concepts underlying JIT production and purchasing systems along with potential problems and benefits have been presented by Hahn [130]. Note that conventional inventory models assume cost per order fixed, but JIT models treat this cost as variable.

Various strategies, effective in JIT purchasing environment, were described, by Ansari and Heckel [9], as well as Schonenberger and Gilbert [280]. They presented among others, the delivery frequency model from the Hewlett-Packard. However the JIT purchasing system needs a more general analytical framework. O'Neal [235] described the behavioural and logistical aspects of the buyer–seller relations. Moreover, he presented an empirical study designed to measure the level of JIT system adoption in the automotive industry. Unfortunately, this

study has some limitations because of too small sample, single industry branch, etc., that limits its real significance. Transportation costs (vehicle routing) in JIT systems linked with the appropriate supplying problem have been considered by Golden and Assad [92]. Note, modern JIT purchasing practices are in contrast with most employed purchasing traditions. This problem is pinpointed in the paper of Schonenberger and Gilbert [280].

2.1.8 Cellular manufacturing systems

Cellular manufacturing systems are closely related to JIT systems, since it is difficult to find JIT system that does not employ this idea. It is one of the best means applied to the implementation at JIT philosophy and total quality control. Cellular manufacturing can be seen as an application of group technology (GT) philosophy, Sassani [270], Wemmerlöw and Hyer [332], where a portion of a firm's production system has been converted to cells or modules. Although several papers deal with cellular manufacturing, application of general GT philosophy in JIT have not received a comparable attention.

2.1.9 JIT implementations

This is the most descriptive and currently the most practical subject of JIT system applications. Generally, JIT manufacturing requires better forecasting, much tighter control of work-in-process and near perfect quality. Parnaby [242] proposes an approach to the implementation of JIT philosophy in selected industry branches. He formulates some basic principles based on simplification and improving reliability of the flow system, creation of cellular manufacture, team work and non–traditional organisational structures. Some aspects of implementations in other branches of industry have been discussed, among others, by Crawford [51], Miltenburg and Wijngaard [200], Richmond and Blackstone [263].

2.1.10 Design of JIT systems

The physical layout of the production facilities in JIT systems is suggested to be U-shaped, Monden [208], due to its flexibility in changing the number of workers on demand. Clearly those workers should be trained to be multi-functional. Among the layouts inappropriate for JIT systems one can find isolated islands, bird cages and linear layouts. In practice, such the re-organisation of production facilities might be possible in some systems.

There have been relatively few models dealing with the equipment selection problems in JIT manufacturing systems, most of them were reviewed by Kusiak and Heragu [179], Kusiak [178]. These models chiefly consider the problem of

selecting the number of machines required, taking account of the standard part processing, daily production of parts, budget, floor space, overtime constraints, influence of Kanbans, lot-sizes, set-up costs, etc.

A general decision framework for the problem of equipment requirements has been proposed there. The final machine configurations are determined by the following factors: (i) machine set-up times, (ii) machine processing capacity, (iii) scrappage rates, (iv) investments in buying machines. Therefore the selection of machines for JIT systems should provide at least: (a) minimum or no in-process inventory, (b) minimum set-up times, (c) minimum or no scrappages, (d) appropriate level of general purpose machinability, (e) integration with the material handling system.

2.2 Towards JIT system needs

The problem of the design of JIT system must incorporate particular features of JIT systems such as no buffers allowed, balanced loading, U-shaped system layout, Kanbans, pull-type production, simple shop floor, reduced set-up times, standardisation of process and products, multi-machine work-centre, and multi-product work-centre, zero deviation from schedule, synchronized scheduling, etc. In order to introduce a master production schedule, a suitable forecasting technique is necessary.

Many of attributes associated with JIT manufacturing appears independently; however, a system having only one or two attributes rarely exists. Aims, such as zero inventory, zero set-up times, zero defects, zero break-downs, etc., set by JIT philosophy are usually difficult to reach, Gunasekaran et al. [122]. Therefore, the models considered should be more complex to be adequate to real systems.

It appears that JIT performs well for the flow-shop type production, however other production environments e.g. job-shop or flow- and job-shop with parallel uniform and non-uniform machines should also be investigated. Considered structures should employ cellular decomposition of production facilities. The problem of allocation buffers in front of different stages may be an interesting area for further research.

The major JIT research should be concentrated on manufacturing, purchasing, Kanbans and WIP inventory reduction. There is also need for research on various optimisation problems in JIT systems. These models can be designed for the use of queering theory methods, scheduling and sequencing algorithms, transfer line models without buffer, control theory and other methods mentioned earlier.

2.3 Discrete optimisation methods

Most of the problems of optimal planning and control in JIT manufacturing systems are either pure *discrete* or *discrete-continuous* optimisation programs. For simplicity, we will write these problems as $\min_{x \in \mathcal{X}} K(x)$, where $\mathcal{X}$ is the set of feasible solutions. Although these problems have simple formulations, they are very troublesome if they are considered from the algorithmic point of view. From the nature, they have a huge number of various local extrema. Their NP-hardness essentially restricts the set of approaches which can be applied to solve the problem. In the literature, one can find exact algorithms, as well as a rich variety of approximation methods. A lot of these procedures have been designed for special cases, since particular properties of the problems usually allow us to improve the numerical characteristics of an algorithm. It is more and more frequent to meet a problem that has many exact and approximation algorithms developed. These algorithms differ from each other by the computational complexity, accuracy, speed of convergence, etc. Many of very recent approaches, not sensitive to local extrema, refer to Nature, Artificial Inteligence (AI) and Machine Learning (ML). Finally, just the *user* has to make a decision which – as the only one among these algorithms – is the most suitable for industrial applications in order to support manufacturing process. Basically this needs the decision whether we search for an optimal solution (at high cost of the search, almost exponential dependending on the problem size), or a sub-optimal solution of *acceptable* solution quality within a *reasonable* time. Note, due to the possibility of performing *parallel computations* some among the methods presented hereafter can be accelerated additionally.

2.3.1 Exact methods

The exact optimisation algorithms are based chiefly on the branch and bound scheme (B&B) (see e.g. Applegate and Cook [10], Brucker et al. [30], Carlier and Pinson [38]), mixed integer programming (MIP) and dynamic programming (PD). While considerable progress has been made in B&B approach, practitioners still find such algorithms unattractive or restrict their application to a very limited scope. The algorithms are time-consuming and the size of problems, which can be solved within a reasonable time limit, is too small. Moreover, their implementation requires a certain level of programmer's sophistication. Similarly, the size of MIP models is impractically large even for the problems of a small size. These large MIP problems cannot be solved by means of standard methods and software packages in a reasonable time. Although PD models are relatively simple, advanced PD algorithms remain very time– and storage–consuming even for the problem of small size. Therefore, one can say the cost of search for an

optimal solution is still too high comparing with the profits obtained from its application. On the other hand, application of these complex algorithms is fully justified in repetitive manufacturing where minimal cycle time is sought for a small number of products, see paper of McCormick et al. [195]. There, the better solution accuracy multiplied by a great number of cycles provides profits much higher than the cost of computations. Taking account of a broader context, practitioners frequently are not interested in the optimal solution because the data of the problem have some finite accuracy, sometimes they do not reflect all really existed constraints, and very often are disturbed just after the optimal solution has been found. One can regard the stated problems as superfluously rigid. In this context, it is observed a tendency towards avoiding too complex exact methods and replacing them by approximation algorithms, which are a quite good alternative.

2.3.2 Approximation methods

Approximation methods provide a good solution in a quick time. These two parameters have adverse trends, i.e. better solutions require a longer algorithm time. Traditionally, approximation algorithms are classified as either *constructive* or *improvement* method. The former algorithms allow us construct step-by-step a solution or generate a solution. They are very fast, easily implementable, however the performance of generated solution is rather poor. The latter algorithms improve a given solution. They need a slightly longer time to run, they are also easily implementable, and the performance of provided solutions is very good or even excellent. Most of these algorithms are developed and recommended for narrow classes of problems. The ideal approximation algorithm should ensure a compromise between the running time and the solution quality, and allow the user to form this compromise in a flexible way.

Constructive algorithms

These algorithms are based chiefly on the following approaches: (a) static or dynamic priority rules, (b) relaxation to simplest problems, (c) approximations by other problems. Priority rules is one of the popular and commonly applied technique, e.g. Baker [15], Hunsucker and Sah [149]. A constructive algorithm generates either *single* solution or a *fixed a priori subset* of solutions among which the best one is selected. A hybrid combination of these two techniques generates step-by-step a single solution, whereas at each step a partial solution is selected among fixed a priori subset of partial solutions. Algorithms of this class can be applied alone or in a group of competitive algorithms. They are also used as

generators of an initial solution for local search algorithms. Due to combination of several known heuristics by means of certain method of their parametrisation, we can synthesise new algorithms for new problems, Wala and Gądek-Madeja [325]. New interesting proposition is also offered by application of the so–called space of heuristics, Storer et al. [312].

Curtailed B&B

By limiting computer resources (processor time, storage) employed in the classic B&B algorithm one can obtain an approximation algorithm (CB) with some compromise between solution time and solution quality, see for example Kadłuczka and Wala [157]. This refers to the following constraints introduced a priori to B&B method: (a) the depth of the tree is limited, (b) the number of immediate descendants of a node in the tree is limited (either a priori fixed or various, depending on the level of a node), (c) the total number of nodes is limited, (d) the total running time is limited, (d) fathomed are only nodes having the lower bound value smaller than some threshold value (it is usually set as reduction by ϵ the current upper bound value), (e) fathomed are only nodes having a measure of usefulness below a threshold value. Although the reduction of the running time is significant compared it with B&B, CB algorithms continually reveal an explosion of calculations for problems of a greater size.

Local search

Recently a significant attention has been paid to the approximation methods based on various local search approach (LS), since they link to the same extent a high running speed, simplicity of implementation and high accuracy of solutions. Many recent papers have recommended this approach as the most promising for very hard optimisation problems. More advanced LS methods are able to escape from the local minimum, to search the global one. Usually, the implementation of approximation algorithms is simple, however, some of them are quite nontrivial.

Descending search/Hill climbing

The descending search method (DS) is the oldest historically and one of the simplest techniques used to improve a given solution $x^0 \in \mathcal{X}$ [5]. The elementary step of the method performs, for a given solution x^k, a search through a subset of solutions called the *neighbourhood* $\mathcal{N}(x^k) \subseteq \mathcal{X}$ of x^k to find a solution $x^{k+1} \in \mathcal{N}(x^k)$ with the lowest criterion value. If only in the neighbourhood $\mathcal{N}(x^k)$ there exists a solution with the criterion value less than $K(x^k)$, the search repeats from

[5] This method is close to *hill climbing* technique for maximisation problems.

a new starting solution, from the best possible, and the process is repeated as long as the goal function value decreases. The trajectory of the search $x^0, x^1, \ldots$ roll monotonously down towards a local optimum. Clearly, this method ends the activity there. By starting the algorithm several times from various (randomly selected) initial solutions, one can eliminate in a very limited scope its susceptibility to local minima. A mutation of this method, called *descending search with drift* (DSD), accepts on the search trajectory also unvisited solutions with the same criterion value, i.e. it can be $K(x^k) = K(x^{k+1})$.

Random search

The random search method (RS) generates a trajectory of the search $x^0, x^1, \ldots$ which goes up-and-down randomly through the set $\mathcal{X}$. The new solution x^{k+1} in this trajectory is selected at random in the neighbourhood $\mathcal{N}(x^k)$. Clearly, this algorithm returns the best solution from the search trajectory. Although this method is not sensitive to local optima, its convergence to a good solution is generally poor. Some mutations of this method modify the distribution of probability while selecting the neighbour to guide the search into an assumed in advance region of the solution space.

Guided local search

Since random search does not provide algorithms with sufficiently good experimental analysis, there have been proposed algorithms that *guide the search* (GLS) using specific deterministic scenario proposed separately for each particular problem, see e.g. Adams et al. [3] Balas and Vazacopolous [20]. The search is guided to some most *promising* regions of the solution space, whereby these regions were selected a priori taking into account some special properties of the problem. In the mentioned paper, these selected regions were searched using a curtailed B&B method. Thus, the method GLS owns all characteristic elements of CB, as an example, reveals the explosion of calculations if the size of the problem solved exceeds certain threshold value. Despite quite sophisticated implementation of this method, the results obtained for selected applications are quite good. Nevertheless, this approach cannot be extended in an easy way to other problems.

Genetic search

Genetic search (GS) refers to Nature. It uses a subset of distributed solutions $\mathcal{P} \subset \mathcal{X}$ called *population* to lead the search in many areas of the solution space simultaneously. Therefore GS not only avoids local minima but also reaches to many regions of the solution space.

Each solution x called *individual* is coded by a set of its attributes expressed in genetic material (genes, chromosomes). Many techniques of such a coding for various problems have been proposed and developed. The population is controlled, in principle, by three processes: *reproduction*, *crossing-over* and *mutation*. In reproduction phase, individuals are multiplied according to their *adaptation measure* [6]. This guarantees that individuals better adopted will have more descendants in the next generation. Individuals chosen by *selection* from expanded population provide the *matting pool*. Fro this pool we match couples of *parents* which next provide the *off-spring generation*. Each member of this new generation is a new solution x' obtained from the two parent solutions x and y using the *cross-over operators*. Many such operators have been proposed and examined, see for example the reviews of Wala and Chmiel [327, 329] or Vaessens et al. [322]. The mutation phase is an insurance against the loss of valid attributes of the good solutions and a slow mechanism of introducing innovative attributes. It introduces sporadic, random (with a low probability) changes in genetic material.

It is clear that this method have many points of freedom. It needs at least: the technique of coding the solution attributes in the chromosome(s), the form of the adaptation function, the scheme of the matting pool selection, scheme of matching parents, cross-over operators and scheme of mutations. Although many of these elements have already been proposed and examined in case studies, their suitable composition is considered as a key for the success of the algorithm, Wala and Chmiel [328].

The major topic of recent research in genetic algorithms is focused on the ability to control the population dynamics. More and more complex mechanisms of the selection and cross-over operators are proposed. However, some failures have been still observed in these algorithms, they are chiefly expressed by the premature convergence to a local extremum or too poor convergence to a good solution. While GS behaves well for small problems, it suffers from strong suboptimality for lager ones. This is proved that population dynamics is responsible for the premature convergence. In order to improve the fitness of the population, the best individuals are prefered in the reproduction if each generation, which results in the continuous decrease of genotype diversity. This in order decreases the possibility of finding better solutions by reproduction scheme, stops further progress, and one must wait until mutation (after a huge number of generations) introduces a significant change. Therefore, the main goal of a GS algorithm is to adjust the selection as sharply as possible without destroying a genotype diversity.

The algorithm's convergence can be controlled by matting strategies, structuring populations and patterns of social behaviour. Goldberg [90] introduced

[6] The basic adaptation measure is the goal function value $K(x)$ for the individual x.

a new starting solution, from the best possible, and the process is repeated as long as the goal function value decreases. The trajectory of the search $x^0, x^1, \ldots$ roll monotonously down towards a local optimum. Clearly, this method ends the activity there. By starting the algorithm several times from various (randomly selected) initial solutions, one can eliminate in a very limited scope its susceptibility to local minima. A mutation of this method, called *descending search with drift* (DSD), accepts on the search trajectory also unvisited solutions with the same criterion value, i.e. it can be $K(x^k) = K(x^{k+1})$.

Random search

The random search method (RS) generates a trajectory of the search $x^0, x^1, \ldots$ which goes up-and-down randomly through the set $\mathcal{X}$. The new solution x^{k+1} in this trajectory is selected at random in the neighbourhood $\mathcal{N}(x^k)$. Clearly, this algorithm returns the best solution from the search trajectory. Although this method is not sensitive to local optima, its convergence to a good solution is generally poor. Some mutations of this method modify the distribution of probability while selecting the neighbour to guide the search into an assumed in advance region of the solution space.

Guided local search

Since random search does not provide algorithms with sufficiently good experimental analysis, there have been proposed algorithms that *guide the search* (GLS) using specific deterministic scenario proposed separately for each particular problem, see e.g. Adams et al. [3] Balas and Vazacopolous [20]. The search is guided to some most *promising* regions of the solution space, whereby these regions were selected a priori taking into account some special properties of the problem. In the mentioned paper, these selected regions were searched using a curtailed B&B method. Thus, the method GLS owns all characteristic elements of CB, as an example, reveals the explosion of calculations if the size of the problem solved exceeds certain threshold value. Despite quite sophisticated implementation of this method, the results obtained for selected applications are quite good. Nevertheless, this approach cannot be extended in an easy way to other problems.

Genetic search

Genetic search (GS) refers to Nature. It uses a subset of distributed solutions $\mathcal{P} \subset \mathcal{X}$ called *population* to lead the search in many areas of the solution space simultaneously. Therefore GS not only avoids local minima but also reaches to many regions of the solution space.

Each solution x called *individual* is coded by a set of its attributes expressed in genetic material (genes, chromosomes). Many techniques of such a coding for various problems have been proposed and developed. The population is controlled, in principle, by three processes: *reproduction*, *crossing-over* and *mutation*. In reproduction phase, individuals are multiplied according to their *adaptation measure* [6]. This guarantees that individuals better adopted will have more descendants in the next generation. Individuals chosen by *selection* from expanded population provide the *matting pool*. From this pool we match couples of *parents* which next provide the *off-spring generation*. Each member of this new generation is a new solution x' obtained from the two parent solutions x and y using the *cross-over operators*. Many such operators have been proposed and examined, see for example the reviews of Wala and Chmiel [327, 329] or Vaessens et al. [322]. The mutation phase is an insurance against the loss of valid attributes of the good solutions and a slow mechanism of introducing innovative attributes. It introduces sporadic, random (with a low probability) changes in genetic material.

It is clear that this method have many points of freedom. It needs at least: the technique of coding the solution attributes in the chromosome(s), the form of the adaptation function, the scheme of the matting pool selection, scheme of matching parents, cross-over operators and scheme of mutations. Although many of these elements have already been proposed and examined in case studies, their suitable composition is considered as a key for the success of the algorithm, Wala and Chmiel [328].

The major topic of recent research in genetic algorithms is focused on the ability to control the population dynamics. More and more complex mechanisms of the selection and cross-over operators are proposed. However, some failures have been still observed in these algorithms, they are chiefly expressed by the premature convergence to a local extremum or too poor convergence to a good solution. While GS behaves well for small problems, it suffers from strong suboptimality for lager ones. This is proved that population dynamics is responsible for the premature convergence. In order to improve the fitness of the population, the best individuals are prefered in the reproduction if each generation, which results in the continuous decrease of genotype diversity. This in order decreases the possibility of finding better solutions by reproduction scheme, stops further progress, and one must wait until mutation (after a huge number of generations) introduces a significant change. Therefore, the main goal of a GS algorithm is to adjust the selection as sharply as possible without destroying a genotype diversity.

The algorithm's convergence can be controlled by matting strategies, structuring populations and patterns of social behaviour. Goldberg [90] introduced

[6] The basic adaptation measure is the goal function value $K(x)$ for the individual x.

a *sharing function* in fitting individuals (in mate selection) to prevent too close genotype similarity. More direct approaches of *incest prevention* uses the *hamming distance* as a measure of similarity between genotypes, see survey of Davidor [54]. Another approach controls mate selection by a dynamic threshold, Eshelman and Schafer [68]. Further results are derived from structuring the entire population. This can be reached either by splitting up population into *islands* (migration model) or by introducing *overlapping neighbourhoods* which cover the population in its entirety (diffusion model), see papers of Mühlenbein and Gorges-Schleuter [210], Davidor [54], or Gorges-Schleuter [94]. Both models restrict immediate data flows between individuals to local area of the population and thus introduce nature-like features of *ecological niches* to preserve a genotype diversity. The approach based on social behaviour assigns a pattern of attitude to each individual, e.g. pleased, satisfied or disappointed, see Mattfield et al. [197]. The assigned pattern depends, among other, on the goal function value found for this individual. The evaluation of the actual behaviour may change the attitude in the future, so individuals can react differently in the same environmental situation. Each behavioural pattern results in a readiness to go into cross-over, mutation, or *sleeping* (to avoid replacement by offspring).

Up to now, GS algorithms have been applied to several types of problems with various degrees of success. Because of too many control parameters each good implementation of GS requires a huge number of computational experiments. Nevertheless, good results obtained for some applications made allow us to consider GS as one of promising tools for solving hard combinatorial optimisation problems.

Simulated annealing

This local search method (SA) proposed originally by Kirkpatrick et al. [166] uses an analogy to a thermodynamic *cooling* process to avoid local minima and escape from them. States of a solid are viewed as being analogous to solutions, whereas the energy of the solid – analogous to an objective function. During the physical annealing process the temperature is reduced slowly in order to come, at each temperature, to the equilibrium of energy when entropy is maximum. Then, the search trajectory is guided through the set $\mathcal{X}$ in a "statistically suitable" direction.

SA generates a trajectory of the search $x^0, x^1, \ldots$ which goes up–and–down through the set $\mathcal{X}$. The new solution x^{k+1} in this trajectory is selected among those from the neighbourhood $\mathcal{N}(x^k)$ using specific scenario. First, a *perturbed solution* x', i.e. $x' \in \mathcal{N}(x^k)$, is chosen in an ordered way or randomly, assuming usually uniform distribution of probability. This solution can provide either

$K(x') \leq K(x^k)$ or $K(x') > K(x^k)$. In the former case, x' is accepted immediately as the new solution for the next iteration, i.e. $x^{k+1} = x'$. In the latter case x' is accepted as the new solution with probability $\min\{1, e^{-\Delta/T_i}\}$, where $\Delta = K(x') - K(x^k)$ and T_i is a parameter called *temperature* at iteration i [7]. If x' has not been accepted (neither by the former nor by the latter condition), we finally set $x^{k+1} = x^k$. The temperature is changed along iterations using *cooling scheme*. A number of iterations, say m, is performed at a fixed temperature. Although the cooling should be carried out very slowly, most of the authors consider the change of the temperature at every iteration ($m = 1$). Two schemes of the temperature modification are commonly used: geometric $T_{i+1} = \lambda_i T_i$, and logarithmic $T_{i+1} = T_i/(1 + \lambda_i T_i)$, $i = 0, \ldots, N-1$, where N is the total number of iterations, λ_i is a parameter, and T_0 is an initial temperature. Clearly, it has to be $T_N < T_0$ and T_N is close to zero.

The SA method has many control parameters that have to be chosen experimentally. The parameter λ_i is often set as a constant. Assuming known (estimated) values of T_0, T_N and N, one can propose the value of parameter λ_i, as an example $\lambda_i = (T_0 - T_N)/(NT_0T_N)$ for the logarithmic scheme, Osman and Potts [237]. Aarts and van Laarhooven [1] make also several suggestions about the choice of x^0, T_0, λ_i, m and the stopping criterion. They propose to select x^0 at random that helps in randomising the search and removing solution dependence on x^0. The initial temperature is set to be 10 times the maximum value Δ between any two succesive perturbed solutions when both are accepted. In practice, starting from x^0, some number k of succesive trial solutions $x^0, x^1, \ldots, x^{k-1}$ is generated, accepted and the value $\Delta_{\max} = \max_{1 \leq i \leq k} K(x^{i+1}) - K(x^i)$ is inspected. Then, we set $T_0 = -\Delta_{max}/ln(p)$, where $p \approx 0.9$. The following method of finding λ_i has been proposed in [1] for the logarithmic cooling scheme $\lambda_i = ln(1+\delta)/(3\sigma_i)$, where δ is parameter of closeness to the equilibrium (0.1 - 10.0) and σ_i is the standard deviation of $K(x)$ for all x generated at the temperature T_i. The value m is dependent on the nature of random perturbation applied and is set equal the number (or its fraction) of different solutions that can be reached from the given one by introducing a single perturbation. The algorithm can be terminated by limiting the running time, number of iterations, when the temperature is close to zero, or the derivate of the smoothed average value of $K(x)$ at T_i is less that the given parameter ϵ, see also Das et al. [53].

The cooling scheme strongly influences the performance of this procedure. If the cooling is too rapid, SA tends to behave like DSD method and usually becomes trapped in a local optimum of poor quality. If the cooling is too slow, the running time becomes unacceptablely long. In practice the proper selection

[7] In practice x' is accepted if for a chosen random value $R[0, 1]$ inequality $R < e^{-\Delta/T_i}$ holds.

of all control parametrs requires a huge number of computer tests.

Several various modifications of this basic scheme have been considered to improve numerical characteristics of this algorithm. First, by choosing at each iteration the *sample* $\mathcal{M}^k \subset \mathcal{N}(x^k)$ of perturbed solutions of cardinality m or at most m. All these solutions are ordered according to the criteria value. If there exists, among $\mathcal{M}^k$, the solution x' such that $K(x') \leq K(x^k)$, this solution is chosen as the current one for the next iteration. If not, the first solution from $\mathcal{M}^k$ is accepted with the probability $\min\{1, e^{-\Delta/T_i}\}$. Otherwise, the current solution remains, i.e. $x^{k+1} = x^k$. Second, by modifying the probability of acceptance of the perturbed solution, i.e. $e^{-\Delta/T_i}/(1 + e^{-\Delta/T_i})$, to give uniform probability of acceptation downhill ($\Delta \leq 0$) and uphill ($\Delta > 0$) solutions, Glauber [81]. The third modification changes the basic scheme of cooling by introducing periodical rise of the temperature to its initial value T_0.

It has been proved that some assumptions allow SA to converge with probability one to a global optimum, Aarts and van Laarhoven, [1]. Detailed analyses have also been carried out for the asymptotic convergence and the rate of convergence. Since some among these assumptions cannot be implemented in computer programs, the results obtained have more theoretical than practical meaning. Up to now, SA has been applied to several types of problems with various degrees of success. Nevertheless, good results obtained for some applications made allow us to consider SA as one of powerful tools for solving hard combinatorial optimisation problems.

Simulated jumping

This novel approach (SJ) uses also thermodynamic analogy to control the search and escape from local minima, Amin [8]. The so–called *jumping* process bases on rapid cooling and heating of a solid applied alternatively, that results in permanent shaking of the system state. The process of heating and cooling is continued until a desired low–energy state of the system is reached. The shaking of the system not only allows us to get access to many regions of the solution space but it also prevents the algorithm from getting stuck in local minimum and increases the probability of reaching the global minimum.

In practice, SJ introduces an additional heating scheme to SA. Heating is applied when the perturbed solution x' has not been accepted, i.e. when we have to set $x^{k+1} = x^k$. It can take, for example, the form $T_{i+1} = T_i + T_0 * r/i$, where r is the random number between [0.0,0.15]. Clearly, SJ has much more elements which have to be chosen experimentally in order to provide algorithm with a good numerical characteristics. The reported applications of SJ to TSP and QAP problems have shown that its performance is very encouraging, but the

speed of convergence significantly depends on the input parameters.

Geometric search

This approach (GES) follows from the research of Belov and Stolin [22], Sevast'yanov [283] and Fiala [69] concearning some scheduling problems. It provides many approximation schemes with good performance, but unfortunately with too high computational complexity, see also works of Shmoys et al. [285]. Basically it refers to some geometric interpretation of a solution and geometry properties. The most common scheme of these algorithms can be summarised by the following steps: (1) find an obvious solution x not necessarily feasible, i.e. it need to be $x \in \mathcal{X}$, (2) perturb this solution by introducing random (chosen in a reasonable way) values of components of x, to obtain x' not necessarily feasible, (3) transform x' into a feasible solution $x'' \in \mathcal{X}$ using special "spreading" approach. Although GES approach provides algorithms with poor experimental performance, it simultaneously allows one to evaluate analytically their performance. It is a pity that competitive performance is being obtained due to the high computational complexity, see conclusions by Nowicki and Smutnicki [226].

Tabu search

Tabu search (TS) is a method proposed by Glover [84, 85]. The basic version of TS starts from an *initial solution* $x^0 \in \mathcal{X}$ [8]. The elementary step of the method performs, for a given solution x^k, a search through the *neighbourhood* $\mathcal{N}(x^k)$ of x^k. The neighbourhood $\mathcal{N}(x^k)$ is defined by *moves* (*transitions*) performed from x^k [9]. The aim of this elementary search is to find in $\mathcal{N}(x^k)$ a solution x^{k+1} with the lowest cost measured in terms of $K(x)$ or other evaluation functions. Then the search repeats from the best found, as a new starting solution, and the process is continued, yielding the search trajectory $x^0, x^1, \ldots$ A solution x from the search trajectory with the lowest value $K(x)$ is the algorithm's outcome. In order to avoid cycling, becoming trapped to a local optimum, and more general to conduct the search in "good regions" of the solution space, a memory of the search history is introduced. Among many classes of the memory introduced for tabu search [84], the most frequently used is the *short term memory*, called the *tabu list*. This list recorded, for a chosen span of time, solutions from the search trajectory, solutions, selected *attributes* of these solutions, or moves leading to these solutions, all of them treated as a form of prohibition of the future search. A move having prohibited attributes is forbidden (i.e. the solution should be

[8] Some advanced variants of TS can start from an unfeasible solution.

[9] A move transforms a solution into another solution.

skipped in the search), however this move can be performed only if an *aspiration function* evaluates it as sufficiently profitable. The search stops when a given number of iterations without improving $K(x)$ has been reached, the algorithm has performed a given number of iterations, time has run out, and/or a solution with satisfying quality has been found. The *connectivity property* that ensures possibility of finding an optimal solution can be proved for selected problems and neighbourhoods. It states that for any starting solution x^0, there exist a search trajectory $x^0, x^1, \ldots, x^r$, such that $x^{k+1} \in \mathcal{N}(x^k)$ and x^r is the optimal solution. Note that this property does not provide the assurance of performing desired trajectory and discovering the required solution. The connectivity property is also analysed for SA and SJ methods.

Adaptive memory search

The adaptive memory search (AMS) notion is used to define advanced TS schemes which goes beyond subset of components and tools enumerated for TS. It has been verified that a search is successful if an appropriate balance between *intensification* and *diversification* is ensured. Thus, AMS emobdies a broader framework by means of more sophisticated memory techniques as, for example, *long–term memory*, *frequency and recency based memory*, *hashing tabu*, focuses on exploiting a collection of such strategic components as, for example, *strategic oscillation*, *candidate list strategy*, *path relinking*, and adaptive behaviour with learning as, for example, *reactive tabu*, see the papers of Glover and Laguna [86], Battiti and Tecchioli [21], Woodruff and Zemel [338]. More detailed discussion of AMS techniques one can find in paper of Glover [88]. Although simplified TS approaches are sometimes surprisingly successful, AMS methods are proved to be often considerably more effective than TS.

Ant search

This very recent method of optimisation (AS) refers to the "inteligent" behaviour of a colony of cooperative agents represented by the ant community, Dorigo et al. [62]. Although this method is recommended for stochastic combinatorial optimization, there are known its applications to classic discrete problems such as travelling salesman, quadratic assignment or job shop. The search activities are distributed over agents with very simple basic capabilities which, to some extent, mimic the behaviour of real ants. This method is inspired by the role of *pheromone trails* discovered in searching by ants the routes path from their colony to feeding sources and back. While an isolated ant moves essentially at random, an ant meeting the pheromone trail follows it with probability proportional to the

pheromone intensity. Each ant marks the trail by its own pheromone, thus multiplying the trail intensity, however simultaneously the pheromone evaporates from the trail to prevent explosion of the process. Assuming available tracks (based upon problem stated), certain model of an ant behaviour [10], primal distribution of ants over limited area and primal intensity of pheromone trails, we can start the simulation of the search-process with discrete time events. If only the *stagnation* of the search has not been observed, algorithm tends towards good solutions.

The results obtained are encouraging, however up to now they are not competitive to SA or TS method taking into account both the running time and quality of generated solutions.

Scatter search

Scatter search (SS) is designed to operate on a set of points called *reference points*, that constitute good solutions obtained from previous calculations, Glover [87]. This method systematically generates linear combination of the reference points to create new points, each of which is mapped on an associated solution. Solutions that result from the linear combination of the chosen reference points are allowed in turn to serve as inputs to accompanying heuristic process. The heuristic procedures then transform these inputs into improved outcomes, thereby bringing the full circle of approach. These outcomes provide a new set of reference points, and the process starts again. By this approach, linear combinations produced at each stage are dispersed across a region of the solution space whose form is biased by the distribution of reference points.

Similarities between this approach and basic GS technique are immediately evident. SS and GS are "population based" procedures, which start with some collection of solutions and evolve progressively these elements to yield new ones. They also assume that new solutions are generated as a *combination* of existed ones. Besides these similarities, there is certain contrast between both methods. GS produces offspring generation by random selection of parents and by setting random points of chromosome crossing–over, hence realised more democratic policy allowing solutions of all types to be combined. In the same time, SS focus on choosing good solutions as a basis for generating combinations without losing the ability to produce diverse solutions. Also, different mechanisms are used by these approaches to obtain the combined solutions.

[10] Ant has some short term memory and can see. Thus, moves can be performed as an elementary step towards some preferable directions.

Path search

This method (PS) performs the local search using specific scenario, Werner [334]. A number of search trajectories called *search paths* are generated from a given starting solution. A trajectory $x^0, \ldots, x^k$ contains solutions such that $x^{k+1} = \min_{x \in \mathcal{N}(x^k)} F(x)$, where $\mathcal{N}(x^k)$ is a neighbourhood, and $F(x)$ some *evaluation function*, e.g. $F(x) = K(x)$. The trajectory is abandoned if the number (given a priori) of recently generated solutions on this trajectory does not improve the solution which currently is the best. Just after a *non-perspective* search trajectory has been abandoned, a new search trajectory is started from a selected (unvisited) solution among search trajectories already generated. This approach has some similarities to simulated annealing as well as to the tabu search technique.

Chunking

This is not not an antonomous search method but an auxiliary tool in suporting the power of other search methods. *Chunking* (C) means the grouping of basic information units to create the higher–level units, and is considered as a critical aspect of human intelligence, Woodruff [339]. People faced with a hard problem to solve often proceed by linking integrated features they perceive as related and transfer them into solution process. They organize problem data and solving methods in hierachical fashion, which allows one to decompose the problem into sub-problems to solve it. When the problem cannot be decomposed or the decomposition method is unknown, then the common strategy is to group solution attributes to reduce the search space that can lead to discovering and exploiting higher–level relationships. Chunks being problem specific can be either pre-defined by the human problem solver or discovered as a by-product of the search. Chunking is related to Machine Learning and can be applied in many local search methods, i.e. GS, SA, SJ, TS, AMS, etc.

Constraint satisfaction

The optimisation problem has been replaced by a constraint satisfaction problem (CSP), see e.g. application in paper of Cheng and Smith [47]. Generally speaking this method operates on *decision variables* V [11], *domains* $\mathcal{D}$ for these variables, and *constraints* $\mathcal{C}$ on two or more variables from V. The basic step of this method is to construct in an *ordered* way solutions through depth–first extension of a current partial solution. Each such an extension defines a new CSP having modified $\mathcal{D}$ and $\mathcal{C}$. The new form of $\mathcal{D}$ is obtained by *propagating* the constraints $\mathcal{C}$. Steps

[11] For example $(x, K(x))$ in the considered case.

are repeated when the *inconsistency* in CSP is detected. Then, the search is backtracked from this state, and the process is continued. Specific CSP algorithms vary in the type of level of consistency reinforcement that is employed, in the mechanisms incorporated from recovering from inconsistent search states (e.g. *backjumping*, *dependency-directed backtracking*, *dynamic backtracking*), and in the heuristics utilised for ordering solutions while processing depth–first extension. This method has some similarities to *B&B* or curtailed *B&B* method, thus reveals equally positive and negative effects of these two approaches.

Neural networks

Although neural networks (NN) is a domain developing dynamically, applications of this technique to scheduling problems are still at an initial stage. Based on the deterministic, analogue neural network model of Hopfield and Tank [145] with a symmetric interconnections and quadratic energy function, there is developed in [72, 73] a solution approach to classic job-shop problem. Next, Zhou [343] proposed another significantly simplest network with a single neurone per operation, linear number of interconnections and linear energy function. It has been shown that the latter method is superior both in the terms of solution quality and network complexity. Nevertheless, besides poor simulation studies on a few selected instances up to 400 operations, there are no complete comparative studies on standard benchmarks, which would be helpful in a final performance evaluation of this approach. On the other hand, some recent studies criticise serious shortcomings of Hopfield's nets used for combinatorial optimisation, as an example of inability of finding global minimum and poor scaling properties. Other shortcomings detected, for example, for neural model of TSP cause inability of finding even a feasible solution of the optimisation problem.

2.3.3 Performance of algorithms

Approximation algorithms are usually ranked according to the running time (or computational complexity) and the distance from a generated solution to the optimal solution (an algorithm performance). Several measures of the algorithm performance have been introduced, Błażewicz [25]. These measures can be investigated either experimentally or analytically (worst-case analysis, probabilistic analysis). Experimental analysis being most popular and easy to perform is however a subjective method of evaluation of an algorithm performance since the results depend on a chosen sample of instances. Alternatively, the worst-case and/or probabilistic analyses yield an objective, instance independent evaluation of an algorithm performance. One can say that these analyses provide another,

supplementary, sometimes more suitable, evaluation of the algorithm behaviour. These analyses together with the computational complexity analysis and experimental analysis of the algorithm performance constitute the complete numerical characteristics of an algorithm.

Worst-case analysis

The set of data specifies an instance Z of the problem. Denote by $\mathcal{X}(Z)$ the set of all solutions of this instance, by $K(x;Z)$ the value of a criterion K for the solution x and instance Z. The solution $x^* \in \mathcal{X}(Z)$ such that $K(x^*;Z) = \min_{x\in\mathcal{X}(Z)} K(x;Z)$ is called the optimal solution for this instance. Let $x^A \in \mathcal{X}(Z)$ denote a solution generated by an algorithm A. The *worst-case performance ratio* of the algorithm A with the criterion K is defined as $\eta^A(K) = \min\{y : K(x^A;Z)/K(x^*;Z) \leq y$ for each instance $Z\}$. The *asymptotic worst-case performance ratio* is $\eta^A_\infty(K) = \min\{y : K(x^A;Z)/K(x^*;Z) \leq y$ for each instance Z such that $K(x^*;Z) \geq L$, where L is a number $\}$. Clearly, it has to be $\eta^A(K) \geq \eta^A_\infty(K)$. The guarantee of solutions quality can be ensured also using the so–called approximation schemes. Algorithm A is an *approximation scheme* if for the given Z and $\epsilon > 0$ provides solution x^A such that $K(x^A;Z) \leq 1+\epsilon$. A is *polynomial approximation scheme* if for any fixed ϵ A has a polynomial computational complexity. If additionally this complexity is a polynom of $1/\epsilon$, A is called a *fully polynomial approximation scheme.* In the sequel, the argument Z will be omitted in $K(x^*;Z)$ and $K(x^A;Z)$ if it is not necessary.

Probabilistic analysis

In this analysis it is assumed that Z is a realisation of n independent random variables with known (usually uniform) distribution of probability. We denote this fact using symbol Z_n. Hence, both $K(x^*;Z_n)$ and $K(x^A;Z_n)$ are random variables. Let $M^A(K;Z_n)$ be the value of the performance measure M of an algorithm A for the criterion K found in the instance Z_n, e.g. $M^A(K;Z_n) = K(x^A;Z_n) - K(x^*;Z_n)$. Clearly, $M^A(K;Z_n)$ is also a random variable. The probabilistic analyses provide primarily information about behaviour of random variable $M^A(K;Z_n)$, e.g. its distribution of probability, the mean value, variance, etc. However, the most interesting are characteristics associated with the type of convergence of $M^A(K;Z_n)$ to a constant m with increasing n. They are distinguished as follows: (a) *almost sure convergence* to a constant m, if $Prob\{\lim_{n\to\infty} M^A(K;Z_n) = m\} = 1$, (b) *convergence in probability*, when $\lim_{n\to\infty} Prob\{|M^A(K;Z_n) - m| > \epsilon\} = 0$ holds for any $\epsilon > 0$, (c) *convergence in expectation*, if $\lim_{n\to\infty} |Mean(M^A(K;Z_n)) - m|\} = 0$. Convergence (a) implies (b), but (b) implies (a) only if for any $\epsilon > 0$ the following condition

holds $\sum_{n=1}^{\infty} Prob\{|M^A(K;Z_n) - m| > \epsilon\} < \infty$. Similarly, (c) implies (b), but (b) implies (c) only under the above additional condition holds. The best is the algorithm having the measure almost sure convergent to zero. Beside the type of convergence there can be also analysed the *speed of convergence.*

Although probabilistic analysis provides interesting characteristics of an algorithm behaviour, it is usually highly complicated. Up to now there are only few basic results, moreover with too simplified assumption about distribution of data.

Experimental analysis

This is the most popular method of analysis despite its imperfections. It consists inxperimental evaluation of algorithm behaviour (performance measures, running time) on the base of limited sample of instances from Z. The sample can be either fixed (common benchmarks instances) or selected at random. Because of NP-hardness of the considered problems, the performance measures used refers more often to *reference solutions* instead of optimal ones. As the reference value one can get: (a) a lower bound of the optimal goal function value, (b) the optimal goal function value, even when it needs to run an algorithm with an exponential computational complexity, (c) a sub-optimal solution obtained by limited run of an exact method, e.g. curtailed $B\&B$, (d) the sub-optimal solution obtained by application of other approximation algorithms, (e) random solution.

Chapter 3

Delivery performance using R/Q models

In this chapter we discuss these JIT environment requirements which can be modelled using conventional notions of scheduling problems such as ready times, due dates, delivery times [1], set-up times, etc. and next solved by means of conventional tools of ST. The proposed approach will be exemplified by a problem, having immediate application in JIT systems as well as in some conventional manufacturing systems.

3.1 Cell work-loading/scheduling

Consider a production cell having m machines, and let $\mathcal{M} = \{1, \ldots, m\}$ be the set of these machines. The set of n parts $\mathcal{N} = \{1, 2, \ldots, n\}$ has to be processed by this cell. Each part $j \in \mathcal{N}$ requires a single machine for processing, posseses the processing time $p_{ij} \geq 0$ on machine $i \in \mathcal{M}$ and has the delivery interval $[a_j, b_j]$ within which this part has to be completed in the ideal case. Each part can be performed on any machine from $\mathcal{M}$. Once started, a part cannot be interrupted. Each machine can execute at most one part at a time, each part can be processed on at most one machine at a time. A *feasible schedule* is defined by a couple of vectors (S, T), $S = (S_1, \ldots, S_n)$, $T = (t_1, \ldots, t_n)$, such that the above constraints are satisfied, where part j is started on machine $t_j \in \mathcal{M}$ at time $S_j \geq 0$ and is completed at time $C_j = S_j + p_{t_j j}$.

The deviation from on time part supply can be measured in several ways [2], see also the survey of Baker and Scudder [18]. To show links of the problem stated

[1] Due to these notions, we call the approach in a little informal way *with the use of R/Q models.*

[2] This subject is discussed in detail in Section 4.1.

with R/Q models from ST, we use in this section a measure based on maximum deviation from delivery interval, namely

$$\max_{j\in\mathcal{N}}[a_j - C_j, C_j - b_j]^+ \tag{3.1}$$

where $[x_1, x_2, \ldots]^+ = \max\{0, x_1, x_2, \ldots\}$. Clearly, we wish to find the schedule that minimises (3.1). Note, if $a_j = b_j$ then the measure (3.1) can simply be written as $\max_{j\in\mathcal{N}} |C_j - b_j|$. An alternative (equivalent) representation of (3.1) is to reach an ideal delivery date $d_j = (a_j + b_j)/2$ with an admissible deviation around this date $\Delta_j = (b_j - a_j)/2$.

To determine *machines workload*, set $\mathcal{N}$ has to be partitioned into m subsets $\mathcal{N}_i \subset \mathcal{N}$, each of which is associated with suitable machine $i \in \mathcal{M}$, and let $n_i = |\mathcal{N}_i|$, $i \in \mathcal{M}$. The *processing order* of parts from $\mathcal{N}_i$ is prescribed by a permutation $\pi_i = (\pi_i(1), \ldots, \pi_i(n_i)) \in \mathcal{P}(\mathcal{N}_i)$, where $\pi_i(j)$ denotes the element of $\mathcal{N}_i$ which is in position j in π_i, and $\mathcal{P}(\mathcal{N}_i)$ is the set of all permutations on the set $\mathcal{N}_i$. The *overalll processing order* is defined by m-tuple $\pi = (\pi_1, \ldots, \pi_m)$. All such processing orders create the set

$$\Pi = \{\pi : (\pi_i \in \mathcal{P}(\mathcal{N}_i),\ i \in \mathcal{M}),\ ((\mathcal{N}_i,\ i \in \mathcal{M}) \text{ is a partition of } \mathcal{N})\}. \tag{3.2}$$

Each π generates a feasible schedule (S, T) in the following manner: set $t_{\pi_i(j)} = i,\ j = 1, \ldots, n_i,\ i \in \mathcal{M}$, and find starting times $S = (S_1, \ldots, S_n)$ by solving the optimisation problem

$$L(\pi) = \min_{S\in\mathcal{S}(\pi)} \max_{j\in\mathcal{N}} [a_j - C_j, C_j - b_j]^+ \tag{3.3}$$

with the set $\mathcal{S}(\pi)$ introduced by constraints

$$C_{\pi_i(j)} \le S_{\pi_i(j+1)},\ \ j = 1, \ldots, n_i - 1,\ \ i \in \mathcal{M}. \tag{3.4}$$

Now, we can rephrase originally stated problem as that of finding $\pi \in \Pi$ that minimises (3.3) under constraints (3.4). The problem (3.3)–(3.4) can be decomposed into m independent single-machine sub-problems

$$L(\pi) = \max_{i\in\mathcal{M}} L(\pi_i) \tag{3.5}$$

where

$$L(\pi_i) = \min_{S\in\mathcal{S}(\pi_i)} \max_{j\in\mathcal{N}_i} [a_j - C_j, C_j - b_j]^+ \tag{3.6}$$

and $\mathcal{S}(\pi_i)$ is given by (3.4) for fixed i. Moreover, we can make use of the following property.

Property 3.1. Problem (3.6) is equivalent to the problem of scheduling jobs from the set $\mathcal{N}_i$ on a single machine, with job *heads* (ready times) $r_j = a_j - p_{ij}$, *tails* (delivery times) $q_j = K - b_j$, where $K = \max_{j\in\mathcal{N}} b_j$, and the makespan criterion $\max_{j\in\mathcal{N}_i}(C_j + q_j)$. □

Proof. Without losing generality we can set $m = 1$. Thus, $i = 1$ (for simplicity in notation we skip index i in all appropriate notions) and $\mathcal{N}_i = \mathcal{N}$. The proof will be made for natural $\pi = (1,\ldots,n)$. We also refer to the well-known formulae giving the makespan value for the single-machine problem with heads and tails and the makespan criterion, see e.g. Grabowski et al. [108], which takes in our case the following form

$$C_{\max}(\pi) = \max_{1\le k\le l\le n}(r_k + \sum_{s=k}^{l} p_s + q_l). \tag{3.7}$$

Since for every pair $1 \le k \le l \le n$ and every $S \in \mathcal{S}(\pi)$, we have

$$\max_{j\in\mathcal{N}}[a_j - C_j, C_j - b_j]^+ \ge \max\{[a_k - C_k]^+, [C_l - b_l]^+\}$$

$$\ge \frac{1}{2}[a_k - C_k + C_l - b_l]^+ \ge \frac{1}{2}[r_k + \sum_{s=k}^{l} p_s + q_l - K]^+, \tag{3.8}$$

then (3.8) provides a lower bound on $L(\pi)$

$$L(\pi) \ge \frac{1}{2}[C_{\max}(\pi) - K]^+ \stackrel{\text{def}}{=} LB(\pi). \tag{3.9}$$

Applying the recursive formulae given below, one can obtain a schedule reaching this bound

$$S_j = \max\{S_{j-1} + p_{j-1}, a_j - p_j - LB(\pi)\},\ j = 1,\ldots,n, \tag{3.10}$$

where $S_0 + p_0 = -\infty$. Indeed, for the values S_j found by (3.10) we immediately have

$$a_j - C_j = a_j - \max\{S_{j-1} + p_{j-1}, a_j - p_j - LB(\pi)\} \le LB(\pi), \tag{3.11}$$

and this holds for any $j \in \mathcal{N}$. Next, we will show that also $C_j - b_j \le LB(\pi)$ for any $j \in \mathcal{N}$. For each fixed j, let k be the job such that $k \le j$, $C_{k-1} < S_k$, and $C_{s-1} = S_s$ for $s = k+1,\ldots,j$. From (3.10) it has to be $S_k = a_k - p_k - LB(\pi)$. Since $C_j = S_k + \sum_{s=k}^{j} p_s$, then employing (3.9), we have

$$C_j - b_j = S_k + \sum_{s=k}^{j} p_s - b_j = a_k - p_k - LB(\pi) + \sum_{s=k}^{j} p_s - b_j$$

$$= r_k + \sum_{s=k}^{j} p_s + q_j - K - LB(\pi) \leq C_{max}(\pi) - K - LB(\pi) \leq LB(\pi) \quad (3.12)$$

From (3.11) and (3.12) we obtain

$$L(\pi) = \max_{j \in \mathcal{N}} [a_j - C_j, C_j - b_j]^+ \leq LB(\pi) \quad (3.13)$$

that yields the property required for the schedule found by (3.10) and completes the proof. ∎

The problem stated in Property 3.1 is denoted as $1|r_j, q_j|C_{\max}$ using Graham et al. notation [117]. Property 3.1 also allows us to reformulate the original problem (3.3)–(3.4) in terms of scheduling problems.

Property 3.2. The problem (3.3)–(3.4) is equivalent to a problem of scheduling jobs from the set $\mathcal{N}$ on m parallel non-uniform, unrelated [3] machines, with machine-dependent job heads $r_{ij} = a_j - p_{ij}$, machine-independent tails $q_j = K - b_j$ and the makespan criterion. □

Proof of this fact is obvious, therefore will be skipped. This new problem can be denoted as $R|r_{ij}, q_j|C_{\max}$ by using denotation from [117]. Finally, we can reformulate the problem originally stated as that of finding $\pi \in \Pi$ which minimises the makespan $C_{\max}(\pi)$ in the reformulated problem, see Property 3.2. Hereafter, we will analyse problem stated only in the form provided by this Property. If all machines in the cell are identical, i.e. $p_{ij} = p_j$, this problem becomes equivalent to the known from the literature problem of scheduling jobs on parallel identical machines with heads, tails, and the makespan criterion $(P|r_j, q_j|C_{\max})$.

It is also convenient in the analysis to use a simple graph representation of a fixed job processing order π, see Fig. 3.1. It takes the form $\mathcal{G}(\pi) = (\mathcal{N} \cup \{o, *\}, \mathcal{A} \cup \mathcal{A}(\pi))$, with the set of nodes $\mathcal{N}$ extended by two auxiliary nodes o (start) and $*$ (end), and the set of arcs $\mathcal{A} = \bigcup_{j \in \mathcal{N}} \{(o, j), (j, *)\}$ and $\mathcal{A}(\pi) = \bigcup_{i \in \mathcal{M}} \bigcup_{j=1}^{n_i - 1} \{(\pi_i(j), \pi_i(j+1))\}$. Each arc $(o, \pi_i(j)) \in \mathcal{A}$ has weight $r_{i\pi_i(j)}$, whereas each arc $((\pi_i(j), *) \in \mathcal{A}$ has weight $q_{\pi_i(j)}$. Each node $\pi_i(j) \in \mathcal{N}$ has weight $p_{i\pi_i(j)}$. Nodes o and $*$ have weight zero. The makespan $C_{\max}(\pi)$ equals the length of the longest path in this graph. It is easy to verify that, by using formula (3.7), this makespan can be expressed

$$C_{\max}(\pi) = \max_{1 \leq i \leq m} \max_{1 \leq k \leq l \leq n_i} (r_{i\pi_i(k)} + \sum_{j=k}^{l} p_{i\pi_i(j)} + q_{\pi_i(l)}). \quad (3.14)$$

[3] These are machines where p_{ij} is defined individually for jobs and machines.

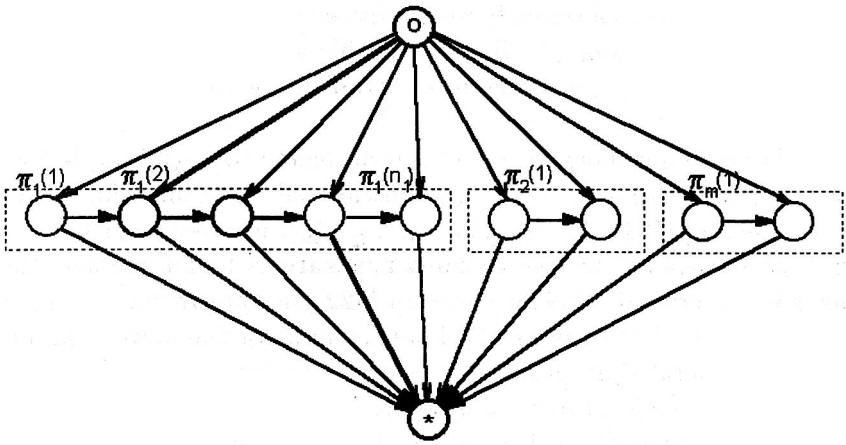

Figure 3.1: The graph structure for the fixed job processing order.

Table 3.1: The computational complexity C and worst-case performance ratio η^A of an algorithm A (developed in B) for the single-machine problem with ready times, delivery times to minimise the makespan.

A	B	C	η^A
S	[281]	$O(n \log n)$	2
P	[250]	$O(n^2 \log n)$	3/2
HS	[132]	$O(n^2 \log n)$	4/3
NS	[229]	$O(n \log n)$	3/2

This formula on $C_{\max}(\pi)$ is convenient to develop some further properties. Despite complexity of (3.14), due to the problem decomposition and graph representation, the makespan for the given π can be simply found in a time $O(n)$.

3.2 Solution methods

Only few special cases of the problem (3.3)–(3.4) were considered in the literature. The first one refers to single-machine problem with $a_j = b_j$, $j \in \mathcal{N}$, Garey et al. [78], called also the problem with *maximum discrepancy* cost. This problem can be solved in a time $O(n)$. Unfortunately, the key condition necessary to extend further on this approach does not hold for the problem (3.6).

The second case refers to the single-machine problem with min-max cost, Sidney [286], which however needs a strongly restrictive assumption "if $(a_i - p_i \leq a_j - p_j)$ then $(b_i \leq b_j)$, for any i and j". Hence, suitable solution methods are sought, by Properties 3.1–3.2, among methods for problems with heads, tails and the makespan criterion.

Already the single-machine version of this problem $(1|r_j, q_j|C_{\max})$ is NP-hard, Lenstra et al. [192], although there exist polynomial algorithms for some of its special cases, Lenstra [322]. The problem $1|r_j, q_j|C_{\max}$ has received a considerable attention in past twenty years due to many applications found, among others, in scheduling jobs on critical machine, Lenstra [322], in approximation algorithms for job-shop problem, Adams et al. [3], lower bounds for the flow-shop, job-shop and other more general shop problems, see e.g. Bratley et al. [29], Carlier and Pinson [38], Brucker [30], Grabowski et al. [108]. Results of the computer tests show that instances of medium size (up to 1,000 parts) can be solved optimal in a reasonable time by an algorithm of Carlier, [36], which is currently the best one among the known exact algorithms for this problem, see already cited papers [36, 108] and these of Baker and Su [19], MacMahon and Florian [196]. There are proposed, as alternative solution methods, several approximation algorithms, Schrage [281], Potts [250], Hall and Shmoys [132], Nowicki and Smutnicki [229], see Table 3.1. Algorithm S is also known as an extended Jackson rule [154]. Its implementation in $O(n \log n)$ time is given by Carlier [36], whereas the worst case evaluation derives from Kise et al. [167]. There are also proposed, by Hall and Shmoys [132], two approximation schemes for this problem which however, because of great computational complexity, have theoretical rather than practical meaning.

The preemptive version of the single-machine problem is polynomially solvable, Lawler [186], in a time $O(n \log n)$, Carlier [36]. It provides the best lower bound for the single-machine non-preemptive case, Nowicki and Smutnicki [216].

The multi-machine preemptive case with parallel identical machines (denoted as $P|r_j, q_j, pmtn|C_{\max}$) can be approximated with the absolute error ϵ in a pseudopolynomial time $O(n^3(\log n + log(1/\epsilon) + \log p_{max}))$ by using maximum flow problem in a dedicated network, see for details Labetoulle et al. [181], Goldberg and Tarjan [91], Błażewicz et al. [26], Brucker [31]. (There are also solved appropriate versions of the preemptive problem with uniform and unrelated machines.) It provides the best lower bound for the multiple uniform machine non-preemptive case among those from Carlier [37], Vandervelde [324], Hogeveen [143], Carlier and Pinson [39]. However, such a high computational complexity of this method limits its too intensive use in an enumerative process. Thus, there is proposed very recently by Carlier and Pinson [39] an $O(n \log n + nm \log m)$ algorithm for finding the so-called Jackson's pseudo-preemptive schedule that provides a tight

(competitive) lower bound for the multi-machine preemptive problem with uniform machines. These bounds can be used in evaluation of algorithm's quality as well as in B&B schemes.

Problems with zero heads and tails constitute a special category of the related problems. Generally, they cannot be applied in our case (excluding some very special cases), however they can provide valuable insights into construction of heuristic solution methods. The basic result refers to the use of the List Schedule (LS) algorithm with an arbitrary selected list, designed for problem with parallel identical machines and the makespan criterion. This algorithm has the computational complexity $O(n)$ and the worst-case performance ratio $2 - 1/m$. By using in this algorithm the list prepared by means of LPT (Longest Processing Time) priority rule, we obtain a method with the worst-case performance ratio $4/3 - 1/3m$ and the computational complexity $O(n \log n)$, Graham [116]. To solve more general problems with *related* parallel machines we can apply either generalised $LS + LPT$ algorithm or MF (Multifit) method. $LS + LPT$ assigns a job from the LPT list to the free machine with the highest speed, and provides solution with the worst-case performance ratio $2 - 2/(m+1)$, Gonzales et al. [95]. Algorithm MF is based on the First Fit Decreasing (FFT) algorithm of the bin-packing problem, and requires an initial evaluation of the "expected" makespan, Kunde and Steppad [176]. MF assigns jobs from the list ordered accordingly to SPT (Shortest Processing Time) rule using FFD method to the first free machine as long as the "expected" makespan is exceeded. Starting with lower and upper evaluation of the "expected" makespan and using binary separation method, we can tend towards quite good solution. Some methods of choosing primal evaluations of the makespan value have been recommended. The worst-case performance ratio is bounded by the value 1.4 (1.2 for the case of identical machines). The problem with unrelated machines and the makespan criterion has been considered rarely. The most significant obstacle, detected in the research, is associated with an unknown a priori (or hardly predictable) influence of a decision made by a scheduling algorithm on the final goal function value, see e.g. a discussion of this subject for a more general problem by Nowicki and Smutnicki [227]. For the problem with a single cell having unrelated machines, zero heads and tails, and the makespan criterion, there is commonly proposed the LS algorithm with a rule of preparing the initial list (as an example LPT) and the rule ECT (Earliest Completion Time) for scheduling. ECT assigns a job at a time moment t to that machine which provides the minimal completion time of this job. The algorithm LS with arbitrary list and ECT has the worst-case performance ratio m. Another more sophisticated approach LP/ECT based on ECT and Linear Programming (LP), see Potts [251], provides an algorithm with the worst-case performance ratio $2 + [\log(m-1)]$.

3.2.1 Approximation algorithms

In this section, we discuss some approximation algorithms formulated for the problem stated. The wide spectrum of the possible solution methods as well as its numerical characteristics have been presented. Although the provided variety does not exhaust all possible solution approaches, it pointed out these important features of the problem which are crucial for the algorithm construction.

List schedules

Certain basic method can be recommended for the case of uniform machines. The proposed list schedule ($LS + MWR$) is based on the Most Work Remaining priority dispatching rule, the same as in algorithm S. To build it, we schedule, at each moment t where at least one machine and at least one job is available, the available job with maximal tail on available machine. Then we update t and iterate until all jobs are scheduled. Using *heap* structure [4] to represent a static priority queue, this schedule can be found in a time $O(n \log n)$.

Although these procedures were designed for the case of uniform parallel machines only, one can adopt it also for our needs. To this aim, first we construct an auxiliary problem with identical parallel machines, machine-independent processing times $p_{ij} = (1/m) \sum_{i \in \mathcal{M}} p_{ij}(i)$ and machine-independent heads $r_j = (1/m) \sum_{i \in \mathcal{M}} r_{ij}$. Then, this auxiliary problem solved by the algorithm ($LS + MWR$), provides only an approximate job processing for the problem originally stated. The "incorrect" makespan must be recalculated for original job processing times. Clearly, the efficiency of this approximation increases if the variance of processing times decreases.

The application of algorithms LS, $LS + LPT$, MF, and others followed from problems with zero heads and tails is generally restricted because of too many points of "freedom": machine-dependent heads, processing times and tails. Even if we assume machine-independent heads and tails, the insights necessary for forming the list in algorithm LS are usually worse than those for S, P, HS, etc. The job priority needs to take into account a combination of head, tail and the processing time; however, this combination and its influence of the future scheduling result are hardly predictable. In practice, the initial list is formed according to nondecreasing values of approximate heads, or the (weighted) sum of the head, processing time and tail, whereas scheduling at each examined time moment is being performed with the help of ECT heuristic. More sophisticated methods with a prediction of the near future are also discussed by Nowicki and Smutnicki [227]. Computational experiments have not shown superiority of these

[4] The same as in the *heapsort* algorithm.

algorithms over other constructive methods.

Basic TS algorithm

Based on conclusions obtained from some single-machine studies, Smutnicki [302], and researches of more general problems, Nowicki and Smutnicki [230, 231], there has been considered only the neighbourhood based on the so-called *insertion moves*. In our case, this type of moves is associated with π and can be defined by a triple $v = (j, i, y)$, such that job (part) j is removed from $\mathcal{N}_{t_j}$ and then inserted in $\mathcal{N}_i$ in position y in the permutation π_i (y becomes new position of j in π_i extended in such a way). Clearly, it has to be $1 \leq y \leq n_i + 1$ if $i \neq t_j$, $1 \leq y \leq n_i$ if $i = t_j$.

The neighbourhood of π consists of orders $(\pi)_v$ generated by moves v from a given set. Using natural, simple approach one can propose the *insertion neighbourhood* $\{(\pi)_v : v \in \mathcal{V}(\pi)\}$ generated by the move set

$$\mathcal{V}(\pi) = \bigcup_{j \in \mathcal{N}} \bigcup_{i \in \mathcal{M}} \mathcal{V}_{ji}(\pi) \tag{3.15}$$

where $\mathcal{V}_{ji}(\pi) = \bigcup_{y=1}^{n_i+1}\{(j,i,y)\}$ if $i \neq t_j$ and $\mathcal{V}_{ji}(\pi) = \bigcup_{y=1}^{n_i}\{(j,i,y)\}$ if $i = t_j$. To avoid solution repetitions, set $\mathcal{V}(\pi)$ is on-line examined to remove redundant moves [5]. The set $\mathcal{V}_{ji}(\pi)$ contains moves such that job j is deleted from the permutation π_{t_j} and inserted in all possible positions in the permutation π_i. One can verify that the number of moves in $\mathcal{V}(\pi)$ is $(n-1)^2$. This is one of the biggest insertion neighbourhoods proposed. It possesses the *connectivity property*, see Section 2.3 for its significance. The whole neighbourhood based on $\mathcal{V}(\pi)$ can be searched exactly (using makespans) in a time $O(n^3)$.

The proposed algorithm uses a short-term memory of the search history represented by the cyclic list called the *tabu list*. For the problem considered, we store on this list the attributes of the visited overall processing orders defined by some pairs of adjacent jobs at a cell. The selection of the pairs depends on the move performed. The list is initiated by empty elements. The move performed from a processing order π introduces to the list one or two new elements in the following manner. Let (j, i, y) be a performed move, j' and j'' be the immediate predecessor and successor of job j on machine t_j, respectively. Then, if $i \neq t_j$ we introduce to the tabu list the pair (j', j) if j' exists, and additionally the pair (j, j'') if j'' exists. If $i = t_j$, we introduce (j, j'') if this move is performed on the position y higher than the current position of j in π_{t_j}, or (j', j) otherwise. The

[5] Moves are redundant if they provide the same solutions as an example any move $(\pi_i(y), i, y)$, since it provides π, and one of two moves $(\pi_i(y), i, y-1)$ and $(\pi_i(y-1), i, y)$, since both provide the same processing order.

move $v = (j, i, y)$, $i \neq t_j$, has tabu status if at least one pair $(j, \pi_i(k))$, $y \leq k \leq n_i$, or $(\pi_i(k), j)$, $1 \leq k < y$, is on the list. For $i = t_j$ this move has tabu status if at least one pair $(\pi_i(k), j)$, $x_j < k \leq y$, if $x_j < y$, or at least one pair $(j, \pi_i(k))$, $y \leq k < x_j$, if $x_j > y$, is on the list.

To select the move leading to the next solution on the search trajectory, all moves are divided into three categories: unforbidden, forbidden but promising (these forbidden that lead to the makespan less than $C_{\max}(\pi^{TS})$, where π^{TS} is currently the best processing order found during the search), forbidden and nonpromising (the rest forbidden). Finally, we decide to perform a move selected from those unforbidden and forbidden but promising which yields the lowest makespan.

Advanced TS algorithm

The exact search of the neighbourhood $\{v : v \in \mathcal{V}(\pi)\}$ is expensively time-consuming and makes the search slowly convergent to a good solution. This fault can be eliminated in several ways: (1) by evaluating neighbours with a measure other than the makespan, assuming that the value of this measure can be found in an inexpensive time shorter than $O(n^3)$, (2) by reducing the neighbourhood size with the precise search, (3) by reducing the search complexity with the precise search. We propose an approach that join together (2) and (3) skipping contemporary (1) because of poor convergence to good solutions observed in computer experiments. This approach links together philosophies of the so-called *reduced neighborhoods* and *search accelerator* [231, 232].

First, consider the method of creating the reduced neighborhood. Let (u, w, z) be the sequence of indices (i, k, l), $1 \leq i \leq n$, $1 \leq k \leq l \leq n_i$ that maximises the right hand side of formula (3.14), i.e.

$$C_{\max}(\pi) = r_{u\pi_u(w)} + \sum_{s=w}^{z} p_{u\pi_u(s)} + q_{\pi_u(z)}. \tag{3.16}$$

This sequence determines the unique path (critical path) in the graph $\mathcal{G}(\pi)$, see Fig. 3.1. We call the sequence $B = (\pi_u(w), \ldots, \pi_u(z))$ the *block*, see for example Grabowski et al. [99]. The block corresponds to a subsequence of elements processed on some machine without inserted idle time. In the graph notions, block is a sequence of nodes (without nodes o and $*$) located on the critical path in $\mathcal{G}(\pi)$. We use $\mathcal{B}$ to denote the set of elements from the sequence B. Although, there can exist several sequences (u, w, z) and thus several blocks, we restrict our attention to only one of them arbitrary selected. Based on the introduced notion we can reduce move set $\mathcal{V}(\pi)$ by eliminating a priori some *useless moves*, i.e. moves v such that $C_{\max}((\pi)_v) \geq C_{\max}(\pi)$.

Property 3.3. If $|\mathcal{B}| = 1$ then π is optimal. All moves from $\mathcal{V}(\pi)$ are useless. □

Property 3.4. If $j \notin \mathcal{B}$, then for any $i \in \mathcal{M}$ all moves from $\mathcal{V}_{ji}(\pi)$ are useless. □

Property 3.5. If $j \in \mathcal{B} \setminus \{\pi_u(w), \pi_u(z)\}$, then each move $v = (j, u, y)$, $w < y < z$, is useless. □

Property 3.6. If $j = \pi_u(w)$, then each move $v = (j, u, y)$, $1 \leq y < w$, is useless. □

Property 3.7. If $j = \pi_u(z)$, then each move $v = (j, u, y)$, $z < y \leq n_u$, is useless. □

Property 3.4 provides the highest reduction of the move set. Proofs of properties can be done by analyses of the paths in the graph $\mathcal{G}(\pi)$. Employing Properties 3.3–3.7 we propose the following *reduced set of moves*

$$\mathcal{W}(\pi) = \bigcup_{j \in \mathcal{B}} \bigcup_{i \in \mathcal{M}} \mathcal{W}_{ji}(\pi), \tag{3.17}$$

where sets $\mathcal{W}_{ji}(\pi)$ are defined only for $j \in \mathcal{B}$ in the following manner. If $i \neq u$ we set $\mathcal{W}_{ji}(\pi) = \mathcal{V}_{ji}(\pi)$. If $i = u$ and $j \in \mathcal{B} \setminus \{\pi_u(w), \pi_u(z)\}$ we set $\mathcal{W}_{ji}(\pi) = \mathcal{V}_{ji} \setminus \{v = (j, i, y) : w < y < z\}$. If $i = u$ and $j = \pi_u(w)$ than $\mathcal{W}_{ji}(\pi) = \mathcal{V}_{ji} \setminus \{v = (j, i, y) : 1 \leq y < w\}$. If $i = u$ and $j = \pi_u(z)$ than $\mathcal{W}_{ji}(\pi) = \mathcal{V}_{ji} \setminus \{v = (j, i, y) : z < y \leq n_u\}$ [6].

The set $\mathcal{W}(\pi)$ contains $O((z - w + 1)(n - z + w - 1))$ moves. This evaluation strongly depends on the block cardinality, and varies from $O(n)$ to $O(n^2)$. Assuming almost uniform distribution of jobs on machines we get evaluation of the number of moves of the order $O(n^2/m)$, which is approximately m times smaller than that of $\mathcal{V}(\pi)$. The neighbourhood $\{v : v \in \mathcal{W}(\pi)\}$ also possesses the connectivity property, see next subsection.

We can also make use of the below properties, however applying them we can lose the connectivity property. Although this property is considered to be significant for the success of the algorithm, nevertheless there are known TS algorithms without this property that have excellent numerical properties, see algorithm of Nowicki and Smutnicki [230] and its rank in papers of Veassens et al. [322] and Błażewicz et al. [27].

[6] Set $\mathcal{W}(\pi)$ is also scanned to eliminate redundant moves.

Property 3.8. If $j \in \mathcal{B} \setminus \{\pi_u(w), \pi_u(z)\}$, and $r_{u\pi_u(w)} \leq r_{uj}$ then move $v = (j, u, y)$, $1 \leq y \leq w$, is useless. □

Property 3.9. If $j \in \mathcal{B} \setminus \{\pi_u(w), \pi_u(z)\}$, and $q_{\pi_u(z)} \geq q_j$ then move $v = (j, u, y)$, $z \leq y \leq n_u$, is useless. □

Beside the reduction of the neighbourhood size, we provide some further theoretical properties to accelerate the search process. The proposed method reduces the computational complexity of the single neighbourhood search, and can be applied to $\mathcal{W}(\pi)$ as well as to $\mathcal{V}(\pi)$ and the makespan criterion. Let π be a given processing order. For each $s \in \mathcal{N}$ we calculate *virtual heads* R_s and *virtual tails* Q_s, by using the following recursive formulae

$$R_{\pi_i(s)} = \max\{R_{\pi_i(s-1)} + p_{i\pi_i(s-1)}, r_{i\pi_i(s)}\} \tag{3.18}$$

for $s = 1, \ldots, n_i$, $i = 1, \ldots, m$, where $\pi_i(0) = 0$ and $p_{i0} = 0$ for any i and $R_0 = 0$,

$$Q_{\pi_i(s)} = \max\{Q_{\pi_i(s+1)} + p_{i\pi_i(s+1)}, q_{\pi_i(s)}\} \tag{3.19}$$

for $s = n_i, \ldots, 1$, $i = 1, \ldots, m$, where $\pi_i(n_i + 1) = 0$ for any i and $Q_0 = 0$. These values can be found in a time $O(n)$. Component makespans can be found in a time $O(1)$ immediately from the equation

$$C_{\max}(\pi_i) = R_{\pi_i(s)} + p_{i\pi_i(s)} + Q_{\pi_i(s)} \tag{3.20}$$

which holds for any $s = 1, \ldots, n_i$.

Let $v = (j, i, y)$ be a move performed. Observe, this move modifies only single permutation π_{t_j} if $i = t_j$, and exactly two permutations π_{t_j} and π_i if $i \neq t_j$. Since $C_{\max}(\pi) = \max_{i \in \mathcal{M}} C_{\max}(\pi_i)$, then to find the makespan after the move performed, it is enough to provide at most two proper values of component makespans associated with π_{t_j} and π_i. To be more precise, we introduce the denotation $\beta = (\pi)_v$. Concentrate first on permutation β_{t_j} where t_j follows from π. It has been obtained from π_{t_j} by removing single element j. To use this method, we need to re-calculate virtual heads and tails for jobs in β_{t_j}. To distinguish, call them R'_s and Q'_s. Clearly, $C_{\max}(\beta_{t_j})$ can be found in the similar way as (3.20), however using R'_s instead of R_s. All these calculations can be made in a time $O(n_i - 1)$. Now, consider β_i that is obtained from π_i by insertion of the job j. By (3.20) for $s = y$ we have

$$C_{\max}(\beta_i) = R_{\beta_i(y)} + p_{i\beta_i(y)} + Q_{\beta_i(y)} = \max\{R_{\beta_i(y-1)} + p_{i\beta_i(y-1)}, r_{i\beta(y)}\}$$

$$+ p_{i\beta_i(y)} + \max\{Q_{\beta_i(y+1)} + p_{i\beta_i(y+1)}, q_{\beta_i(y)}\}. \tag{3.21}$$

Since $\beta_i(y) = j$, due to relations between β_i and π_i, we can write simpler form of (3.21) and thus more sophisticated form of the makespan for the neighbourhood $(\pi)_v$. To this end consider separately two disjoint cases. If $i \neq t_j$ then

$$C_{\max}((\pi)_v) = \max\{C, C_{\max}(\beta_{t_j}), \max\{R_{\pi_i(y-1)}, r_{ij}\} + p_{ij} + \max\{q_j, Q_{\pi_i(y)}\}\} \tag{3.22}$$

where

$$C = \max_{i \in \mathcal{M} \setminus \{t_j\}} C_{\max}(\pi_i) \tag{3.23}$$

If $i = t_j$ then

$$C_{\max}((\pi)_v) = \max\{C, \max\{R'_{\pi_i(y-1)}, r_{ij}\} + p_{ij} + \max\{q_j, Q'_{\pi_i(y)}\}\}. \tag{3.24}$$

Since C can be found while calculating $C_{\max}(\pi_i)$, then assuming known R_j, Q_j, R'_j, Q'_j, we can find single $C_{\max}((\pi)_v)$ in a time $O(1)$. Values R_j, Q_j do not depend on the performed moves so can be found in advance. Values R'_j, Q'_j depend only on job j but not on two remain components i and y of the move (j, i, y). The calculation of R'_j, Q'_j needs a time $O(n_{t_j})$ per job. Therefore, makespans for all $O(n^2/m)$ moves from $\mathcal{W}(\pi)$ can be found in a time $O(n^2/m)$, instead of $O(n^3/m)$ originally required. Note, the computational complexity of the single neighbourhood search has decreased e times. The analogous computational complexity for the move set $\mathcal{V}(\pi)$ is $O(n^2)$ instead of $O(n^3)$.

SA algorithm

We consider the SA algorithm based on the idea of Osman and Potts, [237]. Three versions of algorithm SA have been examined with neighbourhoods based on the move sets $\mathcal{V}(\pi)$, $\mathcal{W}(\pi)$, and $\mathcal{W}(\pi)$ extended by the use of Properties 3.8–3.9 and the random selection of the neighbour. We refer to these variants as algorithms $SA + V$, $SA + W$, and $SA + (W+)$, respectively. Some tests need to be performed to select the best initial and final temperatures, limit of iterations and the best cooling scheme (logarithmic is recommended), see Section 2.3 for details. There are a few propositions to accelerate the speed of this method. Observe, the performed move changes the component mekespans only on two machines: the original and target one. This means that to find the makespan after the move we need to recalculate only these two, which require a time $O(n/m)$ on average. Also observe, zero deviation from the delivery interval is the lower bound of the criterion value. Clearly, SA and TS should be stopped whenever such a solution will be found. Next, since at the low temperature the current processing order becomes the new processing order very often, some neighbourhoods are considered many times. If the sequence of such repetitions

is long (greater than the cardinality of the move set), some moves can be chosen several times. To prevent superfluous (repetitive) calculations, the makespans found for neighbours can be stored and used in the future. The *hashing table* data structure is particularly suitable for such applications [7] and improve the search speed significantly, particularly at the final part of the search trajectory. The suggested *hashing function* is $h(j,i,y) = (j(n-1) + \sum_{s=1}^{i-1} n_s + y \bmod H)$ where H is the size of hashing vector.

Other reduced neighbourhoods

The application of the so-called reduced neigbourhoods in local search methods is intuitively clear and justified. Assuming the neighbourhood too big we get, beside a fast convergence to a good solution, an unacceptably high computational cost of the search, that finally implies a slow convergence in time. In turn, admitting it too small we get the advantegeous accelaration of the search, but simultaneously it has been observed a poor convergence (in iterations) to a good solution. The latter fault is usually associated with inability of escaping from local extrema, or simply with the lack of the connectivity property. If other mechanisms of search continuation are not provided, this local search method will be far from our expectations. Therefore, the proper selection of the move set is considered as the key element for the algorithm and is commonly regarded as the art. In this context, the following question can be asked: how to find a proper balance between these adverse trends of the neighborhood design and where is a final borderline of this reduction, beyond which the connectivity is lost [8]?

Undoubtedly, the fundamental role of the neighbourhood is to set a proper guidance into promising regions of the solution space. The connectivity property ensures continuation of the search process to avoid getting stuck in a local extrema too early. In this section, we consider an extremely reduced neighborhood for the problem (3.3)–(3.4) (i.e. the problem with some particular properties) in the context of the connectivity property. At the end of this section, we generalize the obtained result in such a way as to apply it to the case where no particular problem properties have been found.

Consider a strongly reduced neighborhood based on the move set

$$\mathcal{U}(\pi) = \bigcup_{j \in \mathcal{B}} (\mathcal{U}_{j,u-1}(\pi) \cup \mathcal{U}_{j,u+1}(\pi)) \tag{3.25}$$

where each set $\mathcal{U}_{ji}(\pi) \subseteq \mathcal{V}_{ji}(\pi)$ contains at most one move and is defined only for

[7] The hashing table is used with great advantage in some applications of tabu search method, see Woodruff and Zemel [338].

[8] Some of these aspects have been discussed in [230, 231, 232].

$j \in \mathcal{B}$ in the following manner. First, we assume $\mathcal{U}_{ji}(\pi)$ to be empty if $i \notin \mathcal{M}$. Next, we consider only the case $w < z$ since for $w = z$ we have found optimal solution, see Property 3.3. If $i \neq u$ the set $\mathcal{U}_{ji}(\pi)$ contains only single move (j, i, y) selected arbitrarily (e.g. randomly) from the set $\mathcal{V}_{ji}(\pi)$. The remain cases refer to $i = u$. For $j \in \mathcal{B} \setminus \{\pi_u(w), \pi_u(z)\}$ we set $\mathcal{U}_{ji}(\pi) = \{(j, w), (j, z)\}$. For $j = \pi_u(w)$ we set $\mathcal{U}_{ji}(\pi) = \{(\pi_u(w), \pi_u(z))\}$ only if $z - w > 1$. Otherwise this set is empty as well as the set $\mathcal{U}_{ji}(\pi)$ for $j = \pi_u(z)$. The set $\mathcal{U}(\pi)$ contains $O(z - w)$ moves. Its real cardinality depends on the cardinality of $\mathcal{B}$ and varies from $O(1)$ to $O(n)$. Assuming almost uniform distribution of parts on machines we have on average $O(n/m)$ moves in $\mathcal{U}(\pi)$, which is siginificantly less than these for $\mathcal{W}(\pi)$ and $\mathcal{V}(\pi)$. Clearly, $\mathcal{U}(\pi) \subseteq \mathcal{W}(\pi) \subseteq \mathcal{V}(\pi)$, and $\mathcal{U}(\pi)$ is one of the smallest move sets.

Property 3.10. The neighbourhood based on the move set $\mathcal{U}(\pi)$ satisfies the connectivity property. □

Proof. We need to operate on machine workloads for different processing orders. To prevent faults, notions n_i, N_i, t_j will be indexed by the name of processing order. We start from the definition of the set of job pairs *consistent* with a processing order $\alpha \in \Pi$

$$\mathcal{Y}(\alpha) = \bigcup_{i \in \mathcal{M}} \{(\alpha_i(k), \alpha_i(l)) : 1 \leq k < l \leq n_i^\alpha\} \tag{3.26}$$

Let us define the distance between processing orders α, β as

$$\rho(\alpha, \beta) = M\rho_1(\alpha, \beta) + \rho_2(\alpha, \beta), \tag{3.27}$$

where

$$\rho_1(\alpha, \beta) = \sum_{k \in \mathcal{N}} \left| t_k^\alpha - t_k^\beta \right|, \tag{3.28}$$

alow us to measure the distance between machine workloads, whereas

$$\rho_2(\alpha, \beta) = |\mathcal{Y}(\alpha) \setminus \mathcal{Y}(\beta)|, \tag{3.29}$$

enables the measure meant of the distance between permutations, and M is a sufficiently big integer. This definition differs from that of Nowicki and Smutnicki [231] since it uses another, more tight measure for $\rho_1(\alpha, \beta)$. Note, if $\rho(\alpha, \beta) = 0$ then $\alpha = \beta$.

Consider a processing order $\pi \in \Pi$ which is not optimal, i.e. $C_{\max}(\pi) > C_{\max}(\pi^*)$ where $\pi^* \in \Pi$ is an optimal processing order. First we will show that for each π there exist a move $v \in \mathcal{U}(\pi)$ such that $\rho((\pi)_v, \pi^*) < \rho(\pi, \pi^*)$.

Let us define sets $\mathcal{E} = \{(\pi_u(k), \pi_u(z)) : w \leq k < z\}$ and $\mathcal{F} = \{(\pi_u(w), \pi_u(k)) : w < k \leq z\}$. Consider separately the following exhaustive cases.

Case 1. "$\mathcal{E} \subset \mathcal{Y}(\pi^*)$ and $\mathcal{F} \subset Y(\pi^*)$". There exists in $G(\pi^*)$ a path containing all nodes from the critical path in $G(\pi)$. Therefore $\tau \geq C_{\max}(\pi)$ where τ is the length of this path. Taking account of this statement and assuming that $C_{\max}(\pi) > C_{\max}(\pi^*)$, we get the contradiction $C_{\max}(\pi) > C_{\max}(\pi^*) \geq \tau \geq C_{\max}(\pi)$.

Case 2. "$\mathcal{E} \setminus \mathcal{Y}(\pi^*) \neq \emptyset$". Then there exists j, $j \in \mathcal{B} \setminus \{\pi_u(z)\}$ such that

$$(j, \pi_u(z)) \notin Y(\pi^*) \tag{3.30}$$

and

$$(\pi_u(k), \pi_u(z)) \in Y(\pi^*), \quad x_j < k < z, \tag{3.31}$$

where x_j is the position of job j in the permutation, i.e. $\pi_{t_j}(x_j) = j$. Let $i^* \in \mathcal{M}$ be the machine such that $j \in \mathcal{N}_{i^*}^{\pi^*}$. It follows from (3.31) that there exists the machine $i' \in \mathcal{M}$ such that $\pi_u(k) \in N_{i'}^{\pi^*}$ for $x_j < k \leq z$. Consider two subcases.

Subcase $i^* = i'$. Since jobs j and $\pi_u(z)$ belong to the same set $N_{i^*}^{\pi^*}$, by (3.30) we have

$$(\pi_u(z), j) \in Y(\pi^*). \tag{3.32}$$

Taking into account (3.32) and (3.31) we obtain

$$(\pi_u(k)), j) \in Y(\pi^*), x_j < k < z. \tag{3.33}$$

From (3.33) and the definition of $Y(\pi^*)$ we obtain

$$(j, \pi_u(k)) \notin Y(\pi^*), x_j < k < z. \tag{3.34}$$

Now, we will show that the required move exists.

If $j \neq \pi_u(w)$ or ($j = \pi_u(w)$ and $z - w = 1$) then we chose the move $v = (j, u, z) \in \mathcal{U}(\pi)$. Clearly, $\mathcal{N}_i^{(\pi)_v} = \mathcal{N}_i^{\pi}$, $i \in \mathcal{M}$, and therefore $\rho_1((\pi)_v, \pi^*) = \rho_1(\pi, \pi^*)$. The definition of $Y(\alpha)$ for $\alpha \in \{(\pi)_v, \pi\}$ implies

$$Y((\pi)_v) = [Y(\pi) \setminus \bigcup_{x_j < k \leq z} \{(j, \pi_u(k))\}] \cup \bigcup_{x_j < k \leq z} \{(\pi_u(k)), j)\}.$$

From the above and from (3.32)–(3.33), we get

$$Y((\pi)_v) \setminus Y(\pi^*) = [Y(\pi) \setminus \bigcup_{x_j < k \leq z} \{(j, \pi_u(k))\}] \setminus Y(\pi^*).$$

Taking account of the above equation and from (3.34), (3.30), we get

$$\rho_2((\pi)_v, \pi^*) = |Y((\pi)_v) \setminus Y(\pi^*)| = \rho_2(\pi, \pi^*) - (z - t_j) < \rho_2(\pi, \pi^*).$$

Finally $\rho((\pi)_v, \pi^*) < \rho(\pi, \pi^*)$.

Otherwise, if $j = \pi_u(w)$ and $z - w > 1$ then we chose the move $v = (\pi_u(w+1), u, w) \in \mathcal{U}(\pi)$. Observe, this move provides the same processing order than the move $(\pi_u(w), u, w+1)$ due to redundancy. Therefore, we can make the analysis exactly in the same way as for the move $(\pi_u(w), u, w+1)$ (assuming that one can be performed). By analogy to considerations presented above, we have

$$Y((\pi)_v) = [Y(\pi) \setminus \{(j, \pi_u(x_j + 1))\}] \cup \{(\pi_u(x_j + 1)), j)\},$$

which provides $\rho_2((\pi)_v, \pi^*) = \rho_2(\pi, \pi^*) - 1$ and finally $\rho((\pi)_v, \pi^*) < \rho(\pi, \pi^*)$.

Subcase $i^* \neq i'$. We show that there exists move $v \in \mathcal{U}(\pi)$ such that

$$\sum_{k \in \mathcal{N}} \left| t_k^{(\pi)_v} - t_k^{\pi^*} \right| = \sum_{k \in \mathcal{N}} \left| t_k^{\pi} - t_k^{\pi^*} \right| - 1. \tag{3.35}$$

In order to prove the above, if $u > i^*$ we chose the move $v = (j, u-1, y) \in \mathcal{U}_{j,u-1}(\pi)$. Hence, $t_j^{(\pi)_v} = t_j^{\pi} - 1$, and $t_k^{(\pi)_v} = t_k^{\pi}$ for $k \neq j$. Therefore, we get immediately (3.35). Otherwise, if $u < i^*$ we chose the move $v = (z, u+1, y) \in \mathcal{U}_{j,u+1}(\pi)$. By analogy to the previous case we also get (3.35). Next, using (3.35) we have $\rho_1((\pi)_v, \pi^*) \leq \rho_1(\pi, \pi^*) - 1$. Therefore, $\rho(\pi_v, \pi^*) = M\rho_1(\pi_v, \pi^*) + \rho_2(\pi_v, \pi^*) \leq M(\rho_1(\pi, \pi^*) - 1) + \rho_2(\pi_v, \pi^*) = \rho(\pi, \pi^*) - [M - \rho_2(\pi_v, \pi^*) + \rho_2(\pi, \pi^*)]$ and assuming M sufficiently large we obtain $\rho(\pi_v, \pi^*) < \rho(\pi, \pi^*)$.

Case 3. "$\mathcal{E} \subset \mathcal{Y}(\pi^*)$, and $\mathcal{F} \setminus \mathcal{Y}(\pi^*) \neq \emptyset$". This case can be proved by analogy to the previous case.

We are now ready to show the main result. Since $\rho(\alpha, \pi^*)$ for any $\alpha \in \Pi$ has a finite value then for any initial processing order $\pi^{(1)}$ there exists a finite sequence of processing orders $\pi^{(1)}, \pi^{(2)}, \ldots, \pi^{(r)}$ such that $\rho(\pi^{(i)}, \pi^*) > \rho(\pi^{(i+1)}, \pi^*)$, $\pi^{(i+1)} \in \{\pi_v : v \in \mathcal{U}(\pi^i)\}$, $i = 1, \ldots, r-1$, and $\rho(\pi^{(r)}, \pi^*) = 0$ or $C_{\max}(\pi^{(r)}) \leq C_{\max}(\pi^*)$. Consequently, $\pi^{(r)} = \pi^*$ or $C_{\max}(\pi^{(r)}) = C_{\max}(\pi^*)$. In both cases $\pi^{(r)}$ is some optimal processing order. ∎

There are some implications of the Property 3.10. First, even a relatively small move set ($O(n/m)$ on average) ensures the connectivity property. To that end there is sufficient to move each job from the block on at most two machines (the one immediately preceding and the one immediately suceeding) to any single position there, and to move this job forward (backward) in the leftmost (rightmost) position of this block. Clearly, both $\mathcal{W}(\pi)$ and $\mathcal{V}(\pi)$ satisfy this property, however application of Properties 3.8–3.9 in $\mathcal{W}(\pi)$ can release it. On the other hand, experimental tests show that the move made on machines $u-1$ and u need not be performed either on the remain machines, i.e. $i \in \mathcal{M} \setminus \{u-1, u, u+1\}$, or in a random position y but in the position selected in a reasonable way. Similarly,

moves to the leftmost (rightmost) positions inside the block should be performed still further. Fortunately, the spreading of $\mathcal{U}(\pi)$ to $\mathcal{W}(\pi)$ (slightly greater) has been made in the problem considered at small expenses paid for the calculations. For problems where calculations are more expensive, $\mathcal{U}(\pi)$ is recommended.

The move set $\mathcal{U}(\pi)$ is a tipical example of the explicit application of some useful, particular properties of the problem, see Case 1 in the proof of Property 3.10. For the considered problem, these are the properties derived from the critical path and block notions. If no particular properties for the analysed problem have been found, it follows from the proof of Property 3.10 then the minimal move set with the connectivity property can has the following form

$$\mathcal{Z}(\pi) = \bigcup_{i \in \mathcal{M}} \bigcup_{j \in \mathcal{N}_i} \{(j, i, x_j - 1), (j, i-1, y^*), (j, i+1, y^*)\},$$

where the move $(j, i, 0)$ does not exist and the moves (j, i, y) either if $i \notin \mathcal{M}$, and y^* denotes any (random) position on the appropriate target machine. Intuitively, the move set $\mathcal{Z}(\pi)$ contains moves where each part j is moved in four opposite directions *up*, *down*, *left*, *right* only by single elementary step, i.e. by single machine up (down) in an arbitral position, and by single position to the left (right). The last type of moves is called pairwise interchanges, and due to redundancy many of these moves can be eliminated. The move set $\mathcal{Z}(\pi)$ contains $O(n)$ moves.

Combined algorithm $TS + A$

Following a two-level decomposition of the problem pointed out in Section 3.1 and a good quality of approximation algorithms for single-machine problem one can propose a two-level approximation algorithm that combines TS approach with an approximation algorithm A dedicated to solving m single-machine auxiliary subproblems. On the upper level, the TS approach is used to find the partition $\mathcal{N}_i$, $i \in \mathcal{M}$, i.e. machine workload. Similar technique has been used by Hubscher and Glover [147] for balancing workload of parallel identical machines, however with jobs without heads and tails. On the lower level, for a given partition, m machine sequences are generated by m-times application of an auxiliary approximation algorithm A. An algorithm from these already enumerated as S, P, HS, NS can be recommended to this aim.

There are some essential differences between this algorithm and algorithm TS from the previous section. The main one refers to the computational complexity of the search. Indeed, for the given π calculation of $C_{\max}(\pi)$ needs a time $O(n)$ and this is the cost of checking a single solution. Moreover, by application of fast calculations of the makespans in the neighbourhood, this cost can be reduced to

$O(1)$ per neighbour. On the other hand finding π for the given workload can be found in a time $O(n \log n)$, see Table 3.1, which cannot be reduced. Hence, this method should not be considered as competitive with algorithm TS.

Other special algorithms

If $a_j = b_j$, the single-machine problem formulated in Property 3.1 is equivalent to that of minimising maximum discrepancy, see Section 3.2 which has polynomial-time algorithm [9]. Therefore for this case only, one can propose another two-level algorithm $TS + G$, where on the upper level only the machine workload is searched using, e.g. TS method, whereas on the lower level detailed schedules for all machines are found using the method (G) proposed by Garey et al. [78].

3.2.2 Computational results

A single machine problem

An initial test was made to evaluate the quality of advanced TS versus S, P, HS algorithms for the single-machine problem with heads, tails and the makespan criterion, Smutnicki [302]. Test instances were generated using the scheme of Carlier [36]. For each $n = 50, 100, 150, \ldots, 1000$ and $F = 16, 17, 18, \ldots, 25$ a sample of 10 instances was obtained. The chosen values r_j, p_j, q_j had uniform distributions between 1 and $r_{\max}$, $p_{\max}$, $q_{\max}$, respectively. We set $p_{\max} = 50$, $r_{\max} = q_{\max} = nF$. Thus, 10,000 instances were tested. The instances with $F = 18, 19, 20$ were reported by Carlier as the hardest. Algorithm TS (advanced) was started from the permutation provided by algorithm S. For all instances tested (100%) algorithm TS provided the best result ($C_{\max}(\pi^{TS}) \leq C_{\max}(\pi^{HS})$), whereas for 0.2% of cases only it provided strictly better result ($C_{\max}(\pi^{TS}) < C_{\max}(\pi^{HS})$). Algorithm HS is the best one among S, P, HS, NS both from the worst-case and experimental points of view. Note that in the tested configuration TS has $O(n^2)$ computational complexity, whereas HS has original $O(n^2 \log n)$ computational complexity. It means that for instances with $n = 50, \ldots, 1000$ algorithm TS runs $5 \ldots 10$ times faster than HS. Algorithm TS essentially improves the initial solution in 96.5% of cases, and provides the optimal solution in 99% of cases.

Delivery performance. Algorithm SA.

The main part of performed tests were exactly dedicated to the problem of delivery performance. Test instances were generated using own scheme selected to cover a broad class of real examples. For each $n = 50, 100, 150, \ldots, 1000$ and

[9] This algorithm can be extended to cover the case $b_j - a_j$ =const.

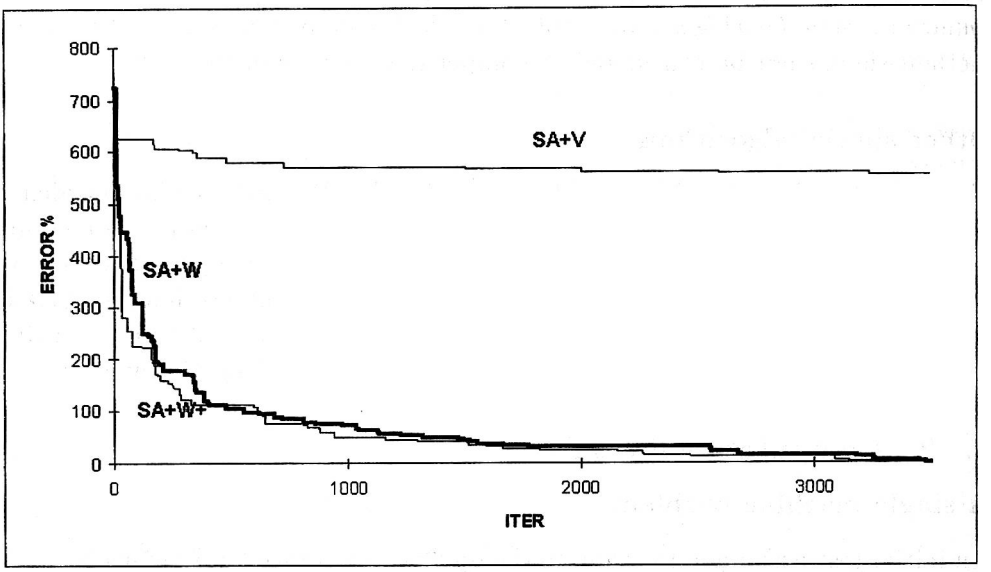

Figure 3.2: Behaviour of algorithm SA in different neighbourhoods (typical run, $n = 300$, $m = 5$, $k = 1.25$)

$m = 2, \ldots, 5$ a sample of 10 instances was obtained. Values a_j, b_j, p_{ij} were chosen as follows. First, there were selected normative processing times p_j with uniform distribution between 1 and $p_{\max}$, delivery dates d_j and deviations from these dates Δ_j with uniform distribution between 0 and $d_{\max}$, $\Delta_{\max}$, respectively. We set $p_{\max} = 50$, $d_{max} = knp_{\max}/(2m)$, $\Delta_{\max} = p_{\max}/2$, and $k = 1.0, 1.25, 1.5, 1.75, 2.0$. The parameter k is used to control the level of difficulty, problems with $k = 1.0$ are much more restrictive than those with $k = 2.0$. Values p_{ij} were obtained by random disorder of p_j within the range $\pm 20\%$, rounded up to the nearest integer. Values a_j, b_j are set as follows $a_j = d_j - \Delta_j$ and $b_j = d_j + \Delta_j$. Algorithm SA was started from the permutation provided by algorithm LS+MWR adjusted to cover unrelated machines case, according to remarks already given. Other constructive heuristics has been skipped in experiments, since both TS and SA outperform them.

At the initial stage, several test were performed to select the best configuration for SA method, observe its behaviour in Fig. 3.3. In all tests, solution times are about few seconds per whole run on a PC (up to 1,000 parts), so all of them are acceptable. Therefore, evaluation of the algorithm performance was concentrated chiefly on other measures of the algorithm performance, namely the quality of generated solutions and speed of convergence.

Some experiments were performed for SA algorithm to establish the influence of control parameters, the neighbourhood, initial temperature, cooling scheme and limit of iterations on the algorithm quality. For $SA+W$ and $SA+(W+)$ we set experimentally the initial (T_0) and final (T_N) temperatures which were 25.0 and 1.0, respectively. This is in accordance with findings of Osman and Potts [237]. It has been suggested to set T_0 approximately $0.1p_{\max}$. Indeed, $TS+V$ works the best for $T_0 = 5.0$. However, it has been verified experimentally that $SA+W$ (also $SA+(W+)$) needs the initial temperature approximately m times greater than $SA+V$. The limit of iterations is set to $N = 10,000$. The logarithmic scheme of cooling is found to be preferable with parameter β calculated from the dependence $\beta = (T_0 - T_N)/(NT_0T_N)$. Algorithm $SA+V$ is found to be very sensitive to proper selection of cooling scheme, which however has not been observed for $SA+W$ and $SA+(W+)$. If we assume a slow reduction of the temperature for $SA+V$, too distant solutions are accepted in the primal phase of the run, and therefore algorithm very poorly converges to a good solution. If we assume, in turn, high reduction of temperature, behaviour of $SA+V$ tends towards descending search with drift methods, which has poor performance. Very subtle differences were found by varying the cooling scheme for $SA+W$ and $SA+(W+)$, chiefly within the first 1,000 iterations. However, no other dominant cooling scheme has been selected. The general conclusions are in conformity with these observed in Fig. 3.2, which presents a selected (typical) run for $n = 300$, $m = 5, k = 1.25$. In Fig. 3.2, there is shown the relative error $100\%\cdot(L(\pi^i)-L)/L$, where $L(\pi^i)$ is the criterion value found at iteration i, and L is the reference value. We set $L = L(\pi^{10,000})$ for the best solutions among all tested variants of SA. Some conclusions are surprising. Observe that the deviation from the initial solution can be relatively high (over 600%) from the best found. This is a consequence of small (near zero) criterion value for the optimal solution and should not be considered as an exception but as a rule. The initial solution can be essentially improved, and the most significant part (80%) of this improvement occurs within first 1,000–2,000 iterations. Generally $SA+V$ very poorly converges to good solution, and within first 2,000 iterations less than 5% improvements appear. Behaviours of $SA+W$ and $SA+(W+)$ are similar, and both algorithms offer quick and efficient reduction of the goal function value. Since the initial solutions are generally very distant from the optimal ones, these valuable properties are especially welcome.

Delivery performance. Algorithm TS.

TS algorithm was tested using the same instances. After some initial runs the best configuration was selected: $maxtT = 8$, $maxiter = 1,000$ and $LS+MWR$

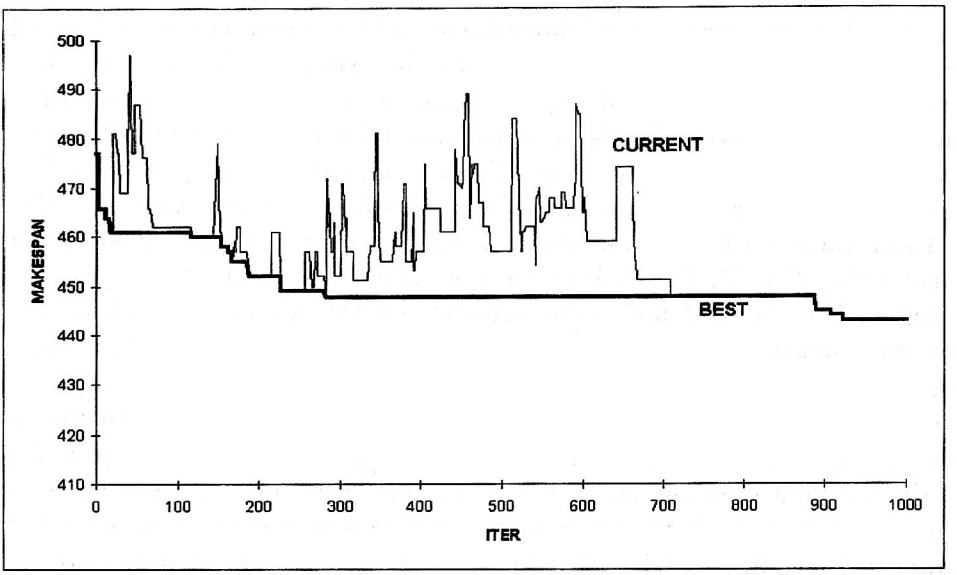

Figure 3.3: First 1,000 iterations of algorithm $SA + W$ (typical run, $n = 50$, $m = 5$, $k = 1.25$)

to find the initial solution (the same as for SA), see also Fig. 3.4. The advanced TS algorithm was found to be superior to basic TS for the reasons pointed out in the previous subsection. No essential differences were found between the use of the neighbourhood $W(\pi)$ and $W(\pi)$ extended by Property 3.8–3.9.

Testing SA versus TS, we found that advanced TS generally generates significantly better solutions than $SA+W$. Moreover, it tends more quickly (compared iterations) to the first best solution, so the initial search trajectory is very steep. In Figure 3.5, there is shown the relative error $100\% \cdot (L(\pi^i) - L)/L$, where $L(\pi^i)$ is the criterion value found at iteration i and L is the reference value for an illustrative instance with $n = 300$, $m = 5$, $k = 1.25$. We set $L = L(\pi^{100})$ where π^{100} is the best solutions from among the tested algorithms SA and TS. The most significant improvements (80%) of TS appear within first 50–100 iterations, which is quite different than in the case of SA. In practice, SA suppresses its activity in a local extremum, not so far from the global one, however because of the low temperature it was unable (in a short time) to exit from this extremum to continue the search. On the other hand, TS can easy exit from such extrema without any difficulty. For all tested instances (100%) algorithm TS provided the better result than SA, and for more than 90% strictly better result. Moreover, TS provided optimal solutions (zero deviations) in many cases. Note, the

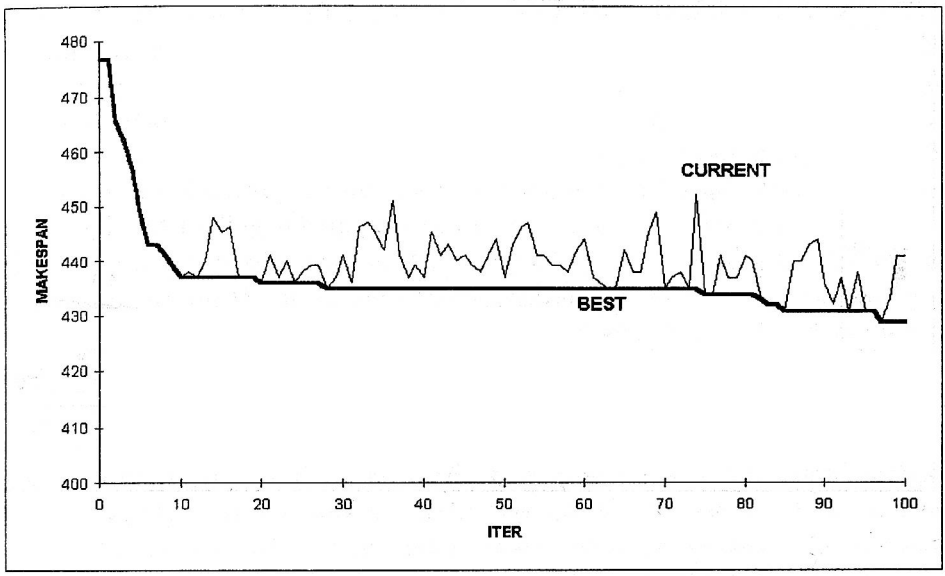

Figure 3.4: First 100 iterations of algorithm $TS + W$ (typical run, $n = 50$, $m = 5$, $k = 1.25$)

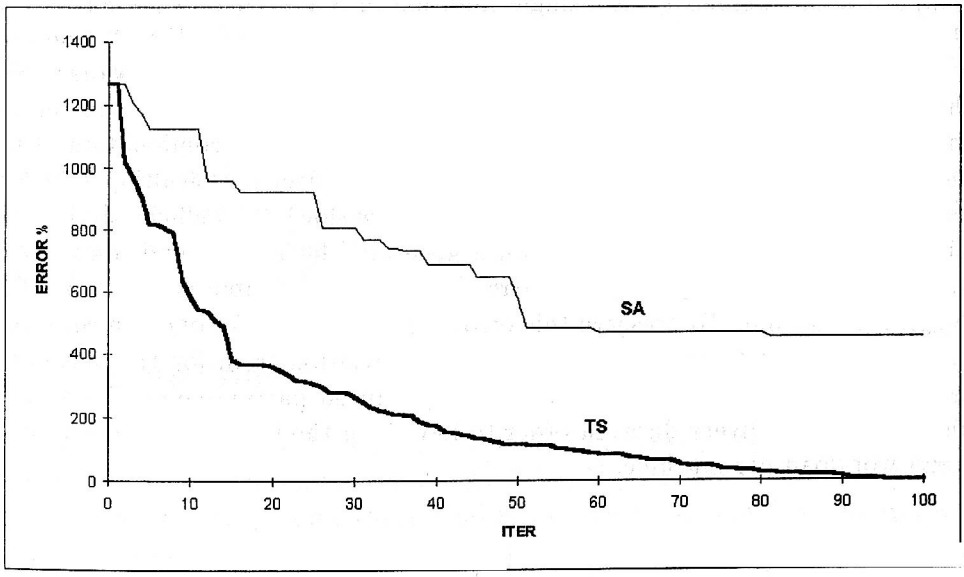

Figure 3.5: Comparisons of $SA + W$ and advanced TS (typical run, $n = 300$, $m = 5$, $k = 1.25$)

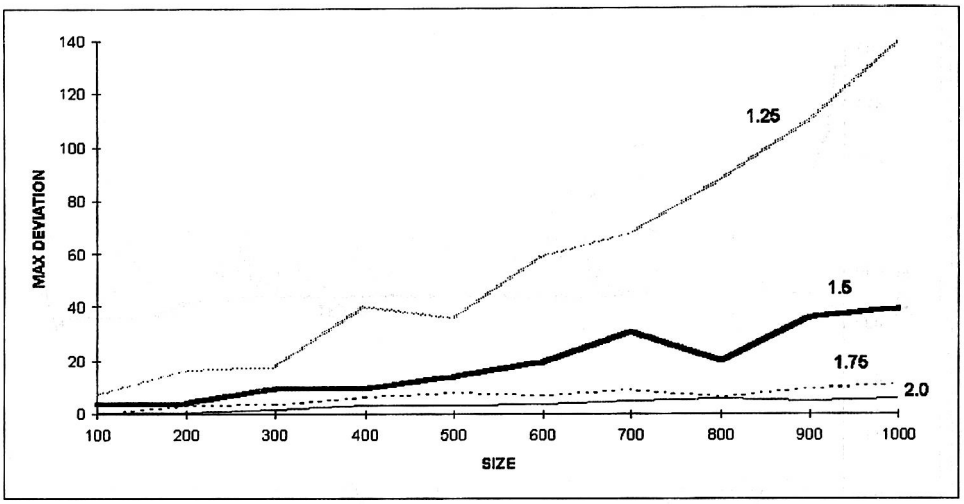

Figure 3.6: Quality of generated solution (advanced TS) depending on k=1.25, 1.5, 1.75, 2.0.

computational complexity of a single iteration of SA is $O(n/m)$, whereas TS is $O(n^2/m)$. Even assuming equal running times for TS and SA (i.e. SA performs n times more iterations), the result provided by SA is essentially worse. Since the time necessary to perform 1,000 iterations for SA is usually shorter than that necessary for 100 iterations of TS, one can propose a combination of both methods. First, SA is run on 1,000 iterations, and then the resulting solution is used as initial one for TS algorithm, which more slowly but efficiently improves the solution. Observed value $L(\pi)$, for π generated by TS, depends on n, m and k. For small n and $k \geq 1.5$, the mean (best) criterion value is about 5–10% of $\Delta_{\max}$, see Fig. 3.6. To preserve this error for the greatest n, one can set greater k. It means that to ensure small maximum deviation from for the delivery interval for overloaded cell (large number of produced parts) one can increase the distribution of delivery dates in order to have it in the interval broader than the mean workload per machine.

3.3 Set-up times in a single-cell model

To increase applicability of the used model, let us introduce additionally machine set-ups to the problem from Section 3.1, formulated in the form derived from

Property 3.2. For this purpose, we accept the following assumption: if the part l is processed immediately after part k on the same machine i ($t_l = t_k = i$), a set-up time Δ^i_{kl} must be taken into considerations, i.e. $C_k + \Delta^i_{kl} \leq S_l$. In practice, due to similarity or identity of produced parts, the set of possible set-up values is finite and small. Quite often, for each part there is introduced a membership to the family of types $\mathcal{F} = \{1, \ldots, f\}$, where $f_j \in \mathcal{F}$ is a type of part j. In this case,y set-ups are defined for these parts which are of different types. Neglecting the philosophy of finding set-up values, one can treat Δ^i_{kl} as given.

Skipping the trivial observation that the model is still working for the set-ups, let us concentrate on the most significant problem of the variety and quality of available solution methods. Algorithms S,P,HS,NS as well as $TS + A$ are hardly adapted to this case, particularly while considering non-uniform machines. Hence, only TS and SA techniques, among these from Section 3.2, remain. Both methods refer to the use of a neighbourhood and eventually to some of Properties 3.3–3.7. Let us discuss this subject in detail. The application of the neighbourhood $\{(\pi)_v : v \in \mathcal{V}(\pi)\}$ is always valid due to its high generality. Among the properties, only Property 3.3 holds without any additional conditions. Property 3.4, the most useful for a reduced neighbourhood, remains valid if only the following condition holds

$$\Delta^i_{kl} \leq \Delta^i_{kj} + \Delta^i_{jl}, \tag{3.36}$$

for any $k, l \in \mathcal{N}$ and $i \in \mathcal{M}$. In practice, this requirement is usually satisfied. It states that making a set-up immediately from part k to part l we cannot do worse than making two consecutive set-ups from k to j and from j to l for any other part j. None of the remain properties need hold. Hence, we can propose another significantly reduced neighbourhood with the size of order $O(n^2/m)$ and having the connectivity property, however slightly greater than that with the move set (3.17). Additionally, the approach described in subsection "Advanced TS algorithm" for the fast makespan calculation can be used after an appropriate modification with all advantageous properties yielding higher search speed.

3.4 Conclusions

Quality of both SA and TS methods has been significantly improved by application of the reduced neighbourhood. Additionally, TS has been significantly accelerated by the use of some theoretical properties of the problem. Hence, we are allowed to formulate the following (hypo)thesis: just the special properties of the problems should be considered as the key for the success of the solution algorithms. Indeed, these properties play the fundamental and very important role in guiding the search through the most promising regions of the solution

space. This fact has been verified and confirmed in many researches conducted by Nowicki and Smutnicki [230, 231, 232].

The obtained transformation of the stated delivery performance problem into a problem with heads, tails and the makespan criterion can be performed for almost all workload/scheduling problems, even with more general production structures. This allows us to solve some problems of delivery using classic scheduling problems. All advantageous properties and algorithms formulated for the latter problem can immediately be applied. Additionally, many results found for classic scheduling problems with the makespan criterion can easily be extended to cover cases with heads and tails, see the best known algorithms in [230, 231, 232].

Application of the proposed min-max measure of delivery performance yields certain profits, but simultaneously it leads to some disadvantages. The used model is simply enough to be solved using very fast and easily implementable approximation algorithms with a high accuracy. This model is also useful while considering problems having e independently minimised criteria, each one for a separate customer order. From this point of view it is much more justifiable application of this model on the borderline between purchaser and client (final product stage) than inside the manufacturing system where it seems difficult to include the fluctuations of delivery times in considerations. On the other hand, minimisation of the maximal deviation permits all jobs to tend towards extreme deviations, which is not distinguished by the criterion value, but unreasonable from the practical point of view. To eliminate these shortcomings one can propose to modify the goal function by adding, e.g., a term with the sum of deviations. Unfortunately, this modification essentially complicates solution methods and goes over the power of theoretical tools used in this section (this subject will be discussed in next sections).

3.5 Delivery performance by max/total cost

In this section we discuss more advanced scheduling models based on the maximum cost and total cost scheduling criteria, which can be applied in JIT environment. The considered models as well as proposed algorithms still remain inside conventional tools of ST.

3.5.1 Scheduling with cost function

Consider a production cell having m machines and let $\mathcal{M} = \{1, \ldots, m\}$ be the set of these machines. Each machine i is available starting from some date g_i, because of not finished production tasks. The set of n parts $\mathcal{N} = \{1, 2, \ldots, n\}$ has to be processed by this cell. Each part $j \in \mathcal{N}$ requires a single machine for

processing, needs a time $p_{ij} \geq 0$ for processing on machine $i \in \mathcal{M}$, has weight w_j (interpreted as priority/urgency weight or the cost of inventory per time unit), and has to be completed *before* the given due date b_j, however as close as possible to this date. Each part can be performed on any machine from $\mathcal{M}$. Once started, a part cannot be interrupted. Each machine can execute at most one part at a time, each part can be processed on at most one machine at a time. A *feasible schedule* is defined by a couple of vectors (S, T) which are the same as in Section 3.1.

To measure the delivery which is not in time we can use the maximum weighted deviation

$$\max_{j \in \mathcal{N}} [w_j(b_j - C_j)], \tag{3.37}$$

and the total weighted deviation

$$\sum_{j \in \mathcal{N}} [w_j(b_j - C_j)], \tag{3.38}$$

since for all $j \in \mathcal{N}$ we must have $C_j \leq b_j$. Note, this criterion concentrates only on a measure of waiting times (between the time of part completion and date of delivery) neglecting processing times, which in the case of unrelated machines does not necessarily minimises WIP. Alternatively, to minimise both delivery performance and WIP in the environment with multiple unrelated machines, one can propose to replace C_j by S_j in formulae (3.37) and (3.38). However, the criteria obtained in such a way generally are not equivalent, which leads to a surprising finding that better delivery performance does not necessarily minimises WIP.

We also refer to Section 3.1 for the notions: machine workload $(\mathcal{N}_i, i \in \mathcal{M})$, machine processing orders $(\pi_i, i \in \mathcal{M})$, overall processing order π, set of all processing orders Π. The feasible schedule (S, T) generated by π implies $t_j = i$, $j = 1, \ldots, n_i$, $i \in \mathcal{M}$. Appropriate starting time $S = (S_1, \ldots, S_n)$ follows from the optimisation problem

$$M(\pi) = \min_{S \in \mathcal{S}(\pi)} \max_{j \in \mathcal{N}} [w_j(b_j - C_j)] \tag{3.39}$$

with the set $\mathcal{S}(\pi)$ introduced by constraints (3.4). Denote by $J(\pi)$ the optimisation problem for the criterion (3.38). Similarly as in the Section 3.1, for fixed π both problems can be decomposed into m independent single-machine subproblems, denoted as $M(\pi_i)$ and $J(\pi_i$, respectively. Next, these subproblems can be expressed in terms of ST.

Property 3.11. Each subproblem $M(\pi_i)$ ($J(\pi_i)$ respectively) is equivalent to the problem of scheduling jobs from the set $\mathcal{N}_i$ on a single machine with heads $r_j =$

$K - b_j$ and a common dead line $d_i = K - g_i$, where $K = \max_{j \in \mathcal{N}} b_j$, to minimise maximum machine-dependent cost $\max_{j \in \mathcal{N}_i} f_{ij}(C_j)$ (total cost $\sum_{j \in \mathcal{N}_i} f_{ij}(C_j)$ respectively), where $f_{ij}(t) = w_j t + v_{ij}$, $v_{ij} = w_j(b_j - K - p_{ij})$. □

Easy proof follows from observation that with respect to a single machine the problem stated originally can be "reversed" providing that from Property 3.10. The problem stated can be interpreted as that of scheduling jobs on parallel unrelated machines with machine-independent heads, machine due dates and machine-dependent cost functions combined either as *max* or *sum* criteria.

3.5.2 Some remarks on solution methods

The dead-line restriction stated in Property 3.10 can be modelled by modifying cost function to obtain a general non-linear or piece-wise linear function having appropriately high barrier value in the dead-line point. Obviously, a processing order π is feasible if the associated cost does not exceed a threshold value. Although such a modification has no obligatory character, it allows us to forget about dead-lines, which complicate the problems a feasible solution existence (this problem is NP hard). In manufacturing practice, many constraints assumed as fixed can be violated at some additional costs. This approach is close to the local search method where unfeasible moves [10] can be accepted with some additional penalty only to ensure the continuation of the search process. This penalty can be included explicitly in the goal function (as above) or can be added to the original criterion value as an independent term. Note, due to restricting the search with feasible moves only we usually obtain an algorithm of superfluous stiffness and thus poor convergence to the best (feasible) solution.

Since the problem can be decomposed into m independent subproblems, one can consider algorithms recommended for single-machine problems with heads and general maximum cost or total cost as the useful tool for constructing solution algorithm. Not specifying here any particular solution method, we only refer to these properties and solution approaches that can be recommended for the construction of the solution algorithm. Based on the conclusions from Section 3.4, we consider only local search approaches as the most promising ones.

Problems with general max-cost criterion have been treated marginally in the literature, see the excellent review by Gupta and Kyparisis [128]. Only maximum *lateness* and *tardiness* are considered more frequently than other max-cost criteria. Nevertheless, research conducted by Smutnicki [291], Grabowski and Smutnicki [102, 103], Grabowski et al. [112], for a general cost function provides its very useful properties, analogous to the so-called *block properties*. These

[10] Moves that lead to an unfeasible solutions.

properties constitute the base for the definition of a reduced neighbourhood and for a suitable method of acceleration of the neighbourhood search. TS approach combined with extended block properties and original search accelerator have provided many algorithms with excellent numerical properties see, for example, papers of Nowicki and Smutnicki [230, 231, 232]. This approach can also be easily extended to many other more general structures of the production systems.

Problems with total cost criterion have appeared more frequently than those with max-cost, since they have better economic interpretation. Problems with heads and total weighted sum of completion times as well as problems with more general production structure are considered as particularly difficult. Generally, Properties 3.1–3.9 do not hold for this problem. Moreover, except some special cases, no significant general properties have been found so far. (Some experiences with constructive and DS algorithms for a single-machine and flow lines have been presented by Nowicki et al. [220].) It means that in the local search algorithm we cannot use the move set $\mathcal{W}(\pi)$, but we can apply either $\mathcal{V}(\pi)$ or $\mathcal{Z}(\pi)$, see Section 3.2 for details. The move set $\mathcal{V}(\pi)$ immediately implies very expensive search time $O(n^3)$ per neighbourhood. In turn, $\mathcal{Z}(\pi)$ (with the connectivity property) contains $O(n)$ moves and requires a significantly shorter search time $O(n^2)$ per neighbourhood. As to search accelerator, excluding the case of zero heads, no changes of the criterion value are predictable in advance. Therefore, we need to re-calculate the goal function value for the newly generated solution to evaluate its quality explicitly. Conclusions drawn in Section 3.4 lead to the statement that any reduction of the move set is especially desirable, first because of too slow SA convergence, next because of too high computational complexity of the single iteration of TS. Since the cost of the whole neighbourhood search becomes too high, SA method is considered as the preferable one. On the other hand TS has been found better than SA taking into account the performance of generated solutions. To prevent too high cost of the single neighbourhood search, there has been proposed the idea of the *restricted neighbourhood*, Diaz [59]. The size of this neighbourhood is controlled by certain parameter, and decreases if the number of performed iterations increases. The restriction prevents the moves from being distant. This restriction has an arbitrary character, quite different than philosophy of the reduction by eliminating useless moves.

Several additional properties, advantageous for constructing a reduced neighbourhood for the problem (3.38), can also be provided. We outline only some of them. Let C_j be the completion times found for processing order π.

Property 3.12. For any move $v = (j, i, y) \in \mathcal{V}(\pi)$ such that $i \neq t_j$ and $1 \leq y \leq k = \max\{1 \leq s \leq n_i : r_j \geq C_{\pi_i(s)}\}$ there exist a move $v' = (j, i, y') \in \mathcal{V}(\pi)$, $y' > k$, such that $J((\pi)_{v'}) \leq J((\pi)_v)$. □

Appropriate, more sophisticated, property exists for the case $i = t_j$ as well. This property restricts the number of considered positions y on each machine i. In practice, assuming an almost uniform distribution of heads in the processing interval and uniform distribution of parts on machines, approximately half of moves from $\mathcal{V}(\pi)$ can be eliminated in this way. Other properties can be found to reduce the number of calculations. Observe, schedule on a single machine with criterion $J(\pi)$ can naturally be decomposed into several *blocks*, each of which corresponds to a subsequence of parts processed consecutively on this machine without inserted idle time. Assume that moves $v = (j, i, y)$ are performed in the inverse order $y = n_i + 1, n_i, \ldots, k + 1$ for some fixed j and $i \neq t_j$. Insertion of j in position $n_i + 1$ can be evaluated in a time $O(1)$ on the base of $J(\pi_i)$ and $f_j(t)$. If only this part adheres to the last block in the resulting processing order, the insertion in position n_i can be interpreted as a swap of immediate adjacent elements, hence the new total cost can be found in a time $O(1)$ by modification of the oldest one [11]. Then the process is repeated. Unfortunately, some exclusion need to be made if the insertion is performed on a position "between" the blocks. Nevertheless, the profits obtained remain significant.

3.6 Applications

One can find a lot of applications of the model proposed and solution methods.

First, they can be used immediately in an interactive system to support scheduling decisions, like that developed by Nowicki and Smutnicki [227]. Values a_j, b_j, w_j can be used as the control parameters (aspiration, reservation level) which allow the user to express his own preferences in the process of developing the satisfactory solution. Note, the user can smooth the faults associated with the use of min-max delivery measure by appropriate setting his own preferences: expected delivery date, deviations from these dates, urgency.

Since the proposed algorithms reduce the criteria value very fast and efficiently, they are highly recommended for the pre-processing phase in more advanced algorithms, designed specially to minimise other, more complex non-term delivery measures.

The pure problem of scheduling jobs on parallel machines with heads, tails and the makespan criterion (auxiliary for our aim), and also its special cases, can be also used for resolving problems of scheduling jobs on critical machines and/or in critical machine nests, in the conventional manufacturing systems. Each time when a bottleneck production unit (machine, machine cell, etc.) has been detected, an additional parallel, identical or more often non-identical machine is

[11] Some additional conditions are necessary to ensure correctness of this manipulation.

added to increase the production power and to remove the bottleneck mark from this production unit. The proposed model and algorithm allow us to analyse and evaluate all *what ... if* cases for any extended machine set.

Another application one can find in a transportation system. Let this system contain m (independent) vehicles. These vehicles have to deliver n products to n points, each of which need to be visited in the time window $[a_j, b_j]$. Once visited, a vehicle spends time p_j at the point j, and the travel from point i to point j needs a time Δ_{ij} (set-up time). Skipping the vehicles capacity, we would like to find the schedule of vehicle routes, which minimises a measure of not on time delivery. In fact, limited vehicle capacities can also be included in this model by imposing a restriction on the feasible machine workload or auxiliary penalty per vehicle overload.

These models can also be used to design and analyse the inventory control in a manufacturing systems. Cyclic manufacturing can be approximated by analysing flows of parts for some, small number of cycles. Fixed delivery dates, like that in the model from Section 3.5, allow us to propose a self-operated JIT inventory control with pull strategy, see Fig. 2.1 (c).

Chapter 4

Delivery performance by using E/T penalties

In this chapter we discuss more complex measures of delivery performance, which can be modelled by using some advanced extensions of scheduling problems, namely the ones having so called *non-regular criteria* or *general earliness and tardiness (E/T) penalty function.* The study of these models is relatively recent area of inquiry in ST, and is strictly associated with the developing of the JIT philosophy. In this models, each customer order (groups of these orders) has determined individual penalty function for non-term delivery. The primal role of this function is to guide solutions toward the target of meeting all required delivery dates exactly. From this point of view, it represents the customer/host expectations. Nevertheless, this function may also expresses explicitly some production costs. Although penalty functions may reflect customer views more accurate, solution methods recommended for these problems remain still unsatisfactory, chiefly because of too high computational complexity. Most of considered hereafter problems can be modelled with the help of linear programming (LP), however already solution methods (particularly these enumerative and local search) need to solve some LP problems a huge number of times. Since one can doubt about effectiveness and rationality of this approach, then generally it will be avoided. The fundamental aim of this chapter is to select problems, their properties, and methods, which allow us to reach assumed goal (the best performance of delivery) under a minimal cost of computations.

At the begin, we introduce the general framework of using E/T penalties, the basic properties of problems, as well as a classification of problems and approaches. Next, some results for a single-machine problem with min-max earliness penalties is presented, Smutnicki [305]. These studies go over basic results formulated for problems with min-max E/T penalties, Sidney [286], Lakshiminarayan

et al. [184], Achuthan et al. [2], and are immediately applicable for problems with more general structure of the production, Grabowski and Smutnicki [110], Smutnicki [299, 300]. In the end, we discuss some aspects of application the total and extended E/T penalties to single-cell scheduling. All approaches will be presented on examples of selected problems, having immediate application in JIT systems.

4.1 E/T measures

In the JIT scheduling environment, final products (semi-products) need to be completed as close as possible to the due date (delivery interval), and equally the early as well as tardy completion is undesirable. The former because of cost of inventory that must be held, the latter because of customer dissatisfaction. Generally, variation around the due date can measured in various ways, see e.g. Section 3.1, nevertheless in this section we consider only these pointers that use nontrivial, appropriately defined E/T penalty functions. There have be selected two sub-classes of such problems: these where due dates are given (fixed), and those where due date is a variable that has to be chosen. The latter case can be used to negotiate the term of delivery. Another two sub-classes are distinguished by admitting that either all jobs can start earlier than time zero, or cannot. These subclasses are called *unrestricted* and *restricted* version of E/T problem, respectively.

All known in the literature problems deal with the single machine, static model, although some of these results have been extended to at most parallel uniform machines case, see the review by Baker and Scudder [18] supplemented by papers since 1990 [4, 7, 34, 78, 131, 135, 139, 175, 209, 321]. Some of these results appear also in the book of Brucker, [31]. Although, one can found in the literature very few more general problems, either no particular algorithm has been proposed or the proposed ad hoc algorithm has not been implemented and verified experimentally. In general, the research concentrate on special, polynomialy solvable cases, or models as simply as possible, because of too high computational complexity of the proposed solution methods. Indeed, problems with E/T penalties are much more troublesome than classic scheduling problems [1]. Because of the *non-regular* form of the goal function, all traditional approaches from ST are useless, and therefore new solution methods are intensively looking for .

Classification of problems will be made assuming that the set of independent

[1] In classic scheduling there are known at least five classes of schedules: *left-justified*, *weakly active*, *strongly active*, *left optimal* and *optimal*, see e.g. [323]. Solutions with E/T penalties generally are not left-justified.

products (jobs) $\mathcal{N} = \{1, \ldots, n\}$ is to be performed in the system, each of which requires a machine (dedicated or selected among a given set) during some uninterrupted processing time. Specification of the machine set has no influence on the classification, so is skipped. Each product has a delivery interval $[r_j, d_j]$ within which it should be completed in the ideal case. Let C_j denote the completion time of this product. Clearly this completion time depends on the schedule. We define also the job earliness $E_j = [r_j - C_j]^+$ and tardiness $T_j = [C_j - d_j]^+$, where $[x]^+ = max\{0, x\}$. A lot of measures have been considered so far in the literature. Although models with delivery tolerance (i.e. such that $r_j < d_j$) are better justified in practice, Cheng [45], almost all known results refer to the case without the tolerance $r_j = d_j$, where delivery interval reduces simply to the *due date* denoted hereafter d_j. There are four distinct classes of measures based on: maximum E/T, linear combination of E/T, non-linear combination of E/T, variations of mutual job completion times. The last class does not introduce explicitly required delivery interval, but extort the schedule to be as much as compact on the time axis. One of the simplest and early recognised is a function of maximum deviation, Sidney [286],

$$\max_{j \in \mathcal{N}} [\max\{g(E_j), f(T_j)\}] \tag{4.1}$$

where $g(t)$ and $f(t)$ are some nondecreasing, continuous, convex functions. Some very special cases of (4.1) have been also considered, e.g. Garey et al. [78], Brucker [31]. A nontrivial generalisation on the case of various functions $g_j(t)$, $f_j(t)$, under relatively weak assumptions about penalty functions, has been provided by Grabowski and Smutnicki [110], and Smutnicki [299, 300, 305]. Note, these measures are quite promising since admit various due dates yet. Further papers provided a lot of measures based on total deviation, as an example simple total absolute deviation from about common due date, Kanet [158],

$$\sum_{j \in \mathcal{N}} |C_j - d| = \sum_{j \in \mathcal{N}} [E_j + T_j] \tag{4.2}$$

with the understanding that $d_j = d$, $j \in \mathcal{N}$. Many of the authors have used such measure, see [18]. By leaving the common due date and introducing two weights w and v we obtain a slightly more general measure, Bagchi et al. [13],

$$\sum_{j \in \mathcal{N}} [vE_j + wT_j] \tag{4.3}$$

The next generalisation extends (4.3) by introducing, nearby the variable common due date, an element of negotiation represented by maximal acceptable due date

d^*, Panwalker et al. [240],

$$\sum_{j\in\mathcal{N}}[vE_j + wT_j + u[d - d^*]^+]. \tag{4.4}$$

Another variation of (4.3) on additional penalties with the common due date refers to the measure introduced by the same authors

$$\sum_{j\in\mathcal{N}}[vE_j + wT_j + uC_j] \tag{4.5}$$

where also flow time is taken into account. By assuming a common, fixed due date, the measures (4.2)–(4.5) have, from some point of view, a limited application in JIT systems where many products need to be delivered in various time moments. This note refers also to cases with variable, negotiable due dates. Moreover, solution of these problems appears to be intrinsically different (significantly simpler) from solutions of problems with distinct due dates. All these enumerated measures can be represented by the following general linear one, containing also models with distinct due dates, considered by many authors, see the review [18],

$$\sum_{j\in\mathcal{N}}[w_jE_j + v_jT_j + u_jC_j + z_jd_j] \tag{4.6}$$

Nonlinear penalties are represented by squared deviation

$$\sum_{j\in\mathcal{N}}(C_j - d)^2 = \sum_{j\in\mathcal{N}}[E_j^2 + T_j^2] \tag{4.7}$$

or a variation of (4.7) admitting the use of the weights v and w, or v_j and w_j. The last class of measures is based on variances between job completion times, Kanet [158],

$$\sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}}|C_i - C_j| \tag{4.8}$$

or

$$\sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}}(C_i - C_j)^2. \tag{4.9}$$

This class has an important application in assembly line leveling problem, see Kubiak [172, 174].

Generally, the total penalty measure can be written as

$$\sum_{j\in\mathcal{N}}[g_j(E_j) + f_j(T_j)] \tag{4.10}$$

where $f_j(t)$ and $g_j(t)$ are some nondecreasing, continuous, convex functions. In this formulation, there are no differences between restricted and unrestricted versions, since the job starting before the time moment zero can be avoided by defining appropriately the penalty functions $g_j(t)$. Note, measures based on total deviation are constructed very often in such a way as to provide the convex goal function. This causes that, the problem for fixed job processing order has a single extremum, not necessarily unique. Otherwise, this problem can own several local extrema, which becomes much more troublesome. From this point of view, some penalty functions as an example barrier functions are not basically allowed and require specific solution techniques. Note also, the measure based on min-max penalty need not the assumption about convexity.

There are a few solution methods applied. Almost all of them use the so-called *V-shaped property.* Briefly, this property states that there is an optimal schedule where jobs concentrate around due date(s). There are many particular forms of this property, developed by many authors, with the dependence on used measure and the assumptions about the problem. Skipping the review of their variety, we only send back the reader to papers already cited. Whereas the form of this property is well understood in the case of the common due date, it is much more complicated when we deal with the distinct due dates. Up to now, a lot of polynomial algorithms have been proposed for special, simply problems, chiefly these having a common due date. More advanced problems, among which are those with distinct due dates, have been solved either by B&B [2] or heuristic algorithms. The search for the optimal schedule is usually decomposed into two subproblems: finding a job sequence and finding job starting times (for the given sequence). Given a job sequence, the optimal job starting times can be produced by solving a linear programming problem. However, for some problems there can exist more efficient polynomial-time procedures, see for example Garey et al. [78], Smutnicki [291]. As to the heuristic methods, most of them use a constructive algorithm followed by the descending search method with the neighbourhood based on job pairwise interchange.

E/T models can successfully be applied in JIT environment, provided that the following conditions are fulfilled: (a) the tolerance of delivery date is preferable, i.e. $r_j < d_j$, (b) the model with distinct delivery dates should be considered as basic one, (c) solution algorithm should be applicable to instances of practical size (100..1000 jobs), (d) in models with variable delivery dates, there should be a small variety of dates, each of them being selected from the given interval or from the given discrete subset of values, (e) considered models as well as algorithms

[2] It has been reported that problems with more than 20 jobs can already lead to exccessive solution times.

should be designed and tested for more general production systems, or at least the proposed approach can be easily extendable to cover such systems.

4.2 Min-max E/T penalties

In this section, we discuss a basic model with general min-max E/T penalties. The proposed approach has been extended to cover more complex production structures, e.g. flow line, Grabowski and Smutnicki [110], and the job shop structure, Smutnicki [300].

The set of n different jobs should be processed on a single machine, which can execute at most one job at a time. Each job j, $j \in \mathcal{N} = \{1, \ldots, n\}$, requires an uninterrupted time p_j for processing and has a cost function $h_j(t)$. It is assumed that the function $h_j(t)$ owns single, not necessarily unique, minimum. The job j which started at the time moment S_j, implies the job cost $h_j(S_j)$. We wish to find the optimal starting times S_j^*, $j \in \mathcal{N}$ of the jobs that minimise the maximum cost $\max_{j \in \mathcal{N}} h_j(S_j)$.

Rough assesment of the formulated model suggests the application of a general non-linear optimisation method in order to solve the stated problem. However, we will provide a more fitting solution method using an alternative formulation with the notion of *job processing order*. The processing order of jobs is represented by a permutation π on $\mathcal{N}$, and let Π be the set of all permutations. Denote by $\mathcal{S}(\pi) = \{S : \ S = (S_1, \ldots, S_n), \ S_{\pi(i)} + p_{\pi(i)} \leq S_{\pi(i+1)}, \ i = 1, \ldots, n-1\}$ the set of feasible starting times of jobs in π, and by $H(\pi)$ the lowest cost obtained for π. $H(\pi)$ can be found by solving the following optimisation problem

$$H(\pi) = \min_{S \in \mathcal{S}(\pi)} \max_{j \in \mathcal{N}} h_j(S_j). \tag{4.11}$$

Now, the problem $1 \,||\, h_{\max}$ can be formulated as that of finding the permutation $\pi^* \in \Pi$, such that

$$H(\pi^*) = \min_{\pi \in \Pi} H(\pi), \tag{4.12}$$

and can naturally be decomposed into two subproblems: find the optimal permutation π^* (optimality is measured by $H(\pi)$) and find the optimal starting times $S^* \in \mathcal{S}(\pi^*)$ by solving (4.11) for π^*.

For needs of the algorithm we will introduce another (equivalent) form of function $h_j(t)$. Let $[x]^+ = \max\{0, x\}$ and $h_j^* = \min_{-\infty < t < \infty} h_j(t)$. Define r_j and d_j so that $r_j = \min\{t : h_j(t) = h_j^*\}, d_j = \max\{t : h_j(t) = h_j^*\}$. Clearly $r_j \leq d_j$. The earliness of the job j started at the time t with respect to r_j is $E_j(t) = [r_j - t]^+$, whereas the tardiness of the job j with respect to d_j is $T_j(t) = [t - d_j]^+$. The earliness cost function is given by $g_j(t) = h_j(r_j - t)$, while

the tardiness cost function by $f_j(t) = h_j(t + d_j)$. Both functions are defined for $t \geq 0$ and are nondecreasing. Note that $h_j(t) = \max\{g_j(E_j(t)), f_j(T_j(t))\}$, and

$$H(\pi) = \min_{S \in \mathcal{S}(\pi)} \max_{j \in \mathcal{N}} \max\{g_j(E_j(S_j)), f_j(T_j(S_j))\}. \tag{4.13}$$

In the sequel, for simplicity in notations we will use E_j, T_j instead of $E_j(S_j)$ and $T_j(S_j)$.

Problems with $h_{\max}$ criterion were considered by Achuthan et al. [2], Lakshiminarayan et al. [184], Sidney [286]. The problem examined in [286] and [184] is a special case of (4.11)–(4.12). All cited papers deal with some properties of the problem and exact algorithms (the problem is NP-hard). Approximation algorithms have not been considered so far.

4.2.1 Algorithm SOL

Algorithm SOL solves the optimisation problem (4.11) or more precisely its equivalent form (4.13). The value $H(\pi)$ can be found from the formula

$$H(\pi) = \max_{(a,b) \in I_{1n}} H_{ab}(\pi) \tag{4.14}$$

where

$$I_{ab} = \{(i,j) : a \leq i \leq j \leq b\}, \tag{4.15}$$

$$H_{ab}(\pi) = \min_{0 \leq t \leq \Delta_{ab}(\pi)} \max\{g_{\pi(a)}(t), f_{\pi(b)}(\Delta_{ab}(\pi) - t)\} \tag{4.16}$$

and

$$\Delta_{ab}(\pi) = [\delta_{ab}(\pi)]^+ = [r_{\pi(a)} - d_{\pi(b)} + \sum_{s=a}^{b-1} p_{\pi(s)}]^+. \tag{4.17}$$

Job starting times $S^* \in S(\pi)$ which ensure minimum of (4.13) can be found in the following way. First, find for $j \in N$ the values

$$E'_j = \max\{t : g_j(t) \leq H(\pi)\} \tag{4.18}$$

where $H(\pi)$ is found by (4.14). Next, find the job starting times $S^* \in \mathcal{S}(\pi)$ using the following recursive formula

$$S^*_{\pi(1)} = r_{\pi(1)} - E'_{\pi(1)}, \tag{4.19}$$

$$S^*_{\pi(j)} = \max\{S^*_{\pi(j-1)} + p_{\pi(j-1)}, r_{\pi(j)} - E'_{\pi(j)}\}, \quad j = 2, .., n. \tag{4.20}$$

Note that this method does not require any additional assumptions (as, e.g. continuity) concearning the cost functions $h_j(t)$. If the functions $h_j(t), j \in \mathcal{N}$ are

continuous and $h_j^* = 0$, then the optimisation problem (4.16) can be replaced by the problem of solving the equation $g_{\pi(a)}(t) = f_{\pi(b)}(\Delta_{ab}(\pi) - t)$. For frequently used piecewise linear cost function

$$h_j(t) = v_j[r_j - t]^+ + w_j[t - d_j]^+, \tag{4.21}$$

the solution of (4.16) takes the form

$$H_{ab}(\pi) = \frac{v_{\pi(a)} w_{\pi(b)}}{v_{\pi(a)} + w_{\pi(b)}} \Delta_{ab}(\pi). \tag{4.22}$$

Since $|I_{1n}| = n(n+1)/2$, then $H(\pi)$ can be found in a time $O(n^2\omega)$ where ω is the number of iterations necessary to solve the problem (4.16) for given a, b and $\Delta_{ab}(\pi)$. The job starting times $S^* \in \mathcal{S}(\pi)$ can be found in a time $O(n\tau)$ where τ is the number of iterations necessary to find E_j' for the fixed j, see (4.18). For the cost function (4.21), both ω and τ are $O(1)$.

Property 4.1. Algorithm SOL is correct. □

Proof. Without losing generality we can assume that $\pi(i) = i$, $i = 1, \ldots, n$. We will prove (4.14) by showing two complementary inequalities "$H(\pi) \geq y$" and "$H(\pi) \leq y$" where $y = \max_{(a,b)\in I_{1n}} H_{ab}(\pi)$, and $H(\pi)$ is defined by (4.13).

Case "$H(\pi) \geq y$". By virtue of (4.13) we have

$$H(\pi) \geq \min_{S\in S(\pi)} \max_{j\in\{a,b\}} \max\{g_j(E_j), f_j(T_j)\} \geq \min_{S\in S(\pi)} \max\{g_a(E_a), f_b(T_b)\}. \tag{4.23}$$

Since we have $T_b \geq 0$ and

$$T_b = T_b + E_a - E_a = [S_b - d_b]^+ + [r_a - S_a]^+ - E_a$$

$$\geq [r_a - d_b + S_b - S_a]^+ - E_a \geq [r_a - d_b + \sum_{s=a}^{b-1} p_s]^+ - E_a = \Delta_{ab}(\pi) - E_a, \tag{4.24}$$

for any $1 \leq a \leq b \leq n$, then $T_b \geq [\Delta_{ab}(\pi) - E_a]^+$. Applying this result in (4.23) we obtain

$$H(\pi) \geq \min_{S\in S(\pi)} \max\{g_a(E_a), f_b([\Delta_{ab}(\pi) - E_a]^+)\}$$

$$= \min_{0\leq E_a<\infty} \max\{g_a(E_a), f_b([\Delta_{ab}(\pi) - E_a]^+)\} = \min\{\min_{0\leq E_a\leq\Delta_{ab}(\pi)} \max\{g_a(E_a),$$

$$f_b([\Delta_{ab}(\pi) - E_a]^+)\}, \min_{\Delta_{ab}(\pi)<E_a<\infty} \max\{g_a(E_a), f_b(0)\}\}$$

$$\geq \min\{\min_{0\leq E_a\leq\Delta_{ab}(\pi)} \max\{g_a(E_a), f_b([\Delta_{ab}(\pi) - E_a]^+)\}, \max\{g_a(\Delta_{ab}(\pi)), f_b(0)\}\}$$

$$= \min_{0\le t\le \Delta_{ab}(\pi)} \max\{g_a(t), f_b(\Delta_{ab}(\pi) - t)\} = H_{ab}(\pi). \tag{4.25}$$

Since (4.25) holds for any $(a,b) \in I_{1n}$, then $H(\pi) \ge \max_{(a,b)\in I_{1n}} H_{ab}(\pi) = y$ which completes the proof of this case.

Case "$H(\pi) \le y$". Let us denote $E_j^* = E_j^*(S_j^*), T_j^* = T_j^*(S_j^*)$. From the definition of y and from (4.18) we have $E_j' = \max\{t : g_j(t) \le y\}$, $g_j(E_j') \le y$, and $g_j(t) > y$ for $t > E_j'$, $j \in N$. Next, from (4.19)–(4.20) it follows that $S_j^* \ge r_j - E_j'$ and therefore $E_j^* = [r_j - S_j^*]^+ \le E_j'$, $j \in N$. Since $g_j(t)$ are nondecreasing, we obtain

$$g_j(E_j^*) \le g_j(E_j') \le y, \ j \in N. \tag{4.26}$$

Next, we will show that

$$f_j(T_j^*) \le y, \ j \in N. \tag{4.27}$$

This fact will be proved by contradiction. Assume that there exists the job v such that $f_v(T_v^*) > y$. Then, find a job with the smallest index u such that the machine is not idle in the time interval $[S_u^*, S_v^*]$, i.e.

$$u = \max\{i \le v : \ S_{i-1}^* + p_{i-1} < S_i^*\}, \tag{4.28}$$

where $S_0^* + p_0 = -\infty$. Such an index always exists and by virtue of (4.19)–(4.20) it has to be $E_u^* = E_u'$, $S_u^* = r_u - E_u^*$, and

$$T_v^* = [S_u^* + \sum_{s=u}^{v-1} p_s - d_v]^+ = [r_u - d_v + \sum_{s=u}^{v-1} p_s - E_u^*]^+ \le [\Delta_{uv}(\pi) - E_u']^+. \tag{4.29}$$

If $E_u' < \Delta_{uv}(\pi)$, we obtain

$$\begin{aligned} y \ge H_{uv}(\pi) &= \min_{0\le t\le \Delta_{uv}(\pi)} \max\{g_u(t), f_v(\Delta_{uv}(\pi) - t)\} = \min\{\min_{0\le t\le E_u'(\pi)} \max\{g_u(t), \\ &f_v(\Delta_{uv}(\pi) - t)\}, \min_{E_u'<t\le \Delta_{uv}(\pi)} \max\{g_u(t), f_v(\Delta_{uv}(\pi) - t)\}\} \\ &\ge \min\{\min_{0\le t\le E_u'(\pi)} f_v(\Delta_{uv}(\pi) - t), \min_{E_u'<t\le \Delta_{uv}(\pi)} g_u(t)\} \\ &\ge \min\{f_v(\Delta_{uv}(\pi) - E_u'), \min_{E_u'<t\le \Delta_{uv}(\pi)} g_u(t)\} \ge \min\{f_v(T_v^*), \min_{E_u'<t\le \Delta_{uv}(\pi)} g_u(t)\} > y. \end{aligned} \tag{4.30}$$

Otherwise, if $E_u' \ge \Delta_{uv}(\pi)$, we have

$$\begin{aligned} y \ge H_{uv}(\pi) &= \min_{0\le t\le \Delta_{uv}(\pi)} \max\{g_u(t), f_v(\Delta_{uv}(\pi) - t)\} \\ &\ge \min_{0\le t\le \Delta_{uv}(\pi)} f_v(\Delta_{uv}(\pi) - t) = \min_{0\le t\le \Delta_{uv}(\pi)} f_v([\Delta_{uv}(\pi) - t]^+) \end{aligned}$$

$$\geq \min_{0\leq t\leq E'_u} f_v([\Delta_{uv}(\pi)-t]^+) = f_v([\Delta_{uv}(\pi)-E'_u]^+) \geq f_v(T^*_v) > y. \qquad (4.31)$$

Both (4.30) and (4.31) imply the contradiction. So, it has to be (4.27). Now we are ready to show the final result. By virue of (4.13) and employing (4.26)–(4.27) we immediately obtain

$$H(\pi) \leq \max_{j\in N} \max\{g_j(E^*_j), f_j(T^*_j)\} = \max\{\max_{j\in N} g_j(E^*_j), \max_{j\in N} f_j(T^*_j)\}$$

$$\leq y = \max_{(a,b)\in I_{1n}} H_{ab}(\pi)$$

which completes the proof of this case. ■

4.2.2 Approximation algorithms

Priority rules

Among many approximation algorithms with priority rules we consider only those based on the classical two-steps technique: (1) find a permutation using job priorities, and next (2) find job starting times using the algorithm SOL. At step (1) one can propose the following obvious rules:

(PR) arrange jobs according to the nondecreasing values r_j,

(PD) arrange jobs according to the nondecrasing values d_j,

(PA) arrange jobs according to the nondecrasing values $(r_j + d_j)/2$.

Although several other rules were considered and investigated by us, the experimental evaluation of the performance was similar. Therefore, in the sequel, we will refer to only one of them, namely PA.

Insertion technique

The approximation algorithm INS has been designed using the so-called insertion technique, introduced primarily for flow shop scheduling with the makespan criterion, Nawaz et al. [212], and applied to many other scheduling problems with regular criteria. Both experimental analysis and worst-case analysis prove that this technique, as the best, can be recommended for problems with the makespan criterion, Nowicki and Smutnicki [226, 231]. Due to its properties the insertion approach seems to be particularly suitable for the problems with irregular criteria.

Without losing generality we assume that jobs are indexed so that $p_j \geq p_{j+1}$, $j = 1,\ldots,n-1$. The algorithm INS generates a sequence of partial permutations $\sigma_1,\ldots,\sigma_n$ where $\sigma_l = (\sigma_l(1),\ldots,\sigma_l(l))$ is a partial permutation on the job set $\{1,\ldots,l\}$, $l = 1,\ldots,n$. Clearly, $\sigma_1 = (1)$. The permutation σ_{l+1}

is created from σ_l in the following way. At first the set Π_{l+1} containing $l+1$ auxiliary partial permutations on the set $\{1,...,l+1\}$ is generated

$$\Pi_{l+1} = \{\sigma_{l+1}^m = (\sigma_l(1),\ldots,\sigma_l(m-1), l+1, \sigma_l(m),\ldots,\sigma_l(l)) : \ m = 1,\ldots,l+1\}. \tag{4.32}$$

The permutation σ_{l+1}^m is created from σ_l by inserting the job $l+1$ in m-th position in σ_l (this insertion moves the jobs $\sigma_l(j)$, $j = m,\ldots,l$ one position to the right). For $m = 1$ the inserted job precedes, whereas for $m = l+1$ succeeds all jobs from σ_l. Next, we select $\sigma_{l+1}^{m'} \in \Pi_{l+1}$ such that $H(\sigma_{l+1}^{m'}) = \min_{1\le m\le l+1} H(\sigma_{l+1}^m)$, and thus we set $\sigma_{l+1} = \sigma_{l+1}^{m'}$ for the next iteration. Obviously we have $\pi^{INS} = \sigma_n$. The computational complexity of this algorithm is $O(n^2\nu)$ where ν is the computational complexity of the algorithm SOL. Unfortunately, already for the cost function (4.21), ν is $O(n^2)$. Therefore the computational complexity of INS is generally $O(n^4\omega)$, whereas for the cost function (4.21) is $O(n^4)$.

Efficient insertion technique

The computational complexity $O(n^4\omega)$ of the algorithm INS is too high for practical applications. Therefore we will provide another more efficient version INS/E of the insertion method, which runs in a time $O(n^3\omega)$. Unfortunately, the computational complexity proposed simultaneously implies more complicated description of the method.

First note that auxiliary permutations σ_{l+1}^m, $m = 1,..,l+1$ in INS are checked in a time $O(l^3\omega)$, which requires $l+1$ runs of SOL. We propose a method that checks these $l+1$ permutations in a time of the single run of SOL, i.e. $O(l^2\omega)$.

Let us begin with some notions and notations. Besides I_{ab}, see (4.15), we define for $a,b,c,d \in \{1,\ldots,n\}$ the following sets

$$J_{ab}^{cd} = \{(i,j) : a \le i \le b,\ c \le j \le d\}, \tag{4.33}$$

and assume $J_{ab}^{cd} = \emptyset$ if $a > b$ or $c > d$. Let σ_l be the permutation obtained in the l-th iteration of the algorithm INS. Analyse an auxiliary permutation σ_{l+1}^m (for a fixed m) obtained in $l+1$-st iteration. Observe that, see also Fig.1,

$$\sigma_{l+1}^m(j) = \sigma_l(j),\ 1 \le j \le m-1, \tag{4.34}$$

$$\sigma_{l+1}^m(j) = \sigma_l(j-1),\ m+1 \le j \le l+1, \tag{4.35}$$

$$\sigma_{l+1}^m(m) = l+1. \tag{4.36}$$

By virtue of (4.34) and (4.35), we have

$$\Delta_{ab}(\sigma_{l+1}^m) = \Delta_{ab}(\sigma_l),\ H_{ab}(\sigma_{l+1}^m) = H_{ab}(\sigma_l) \text{ for } (a,b) \in I_{1,m-1}, \tag{4.37}$$

$$\Delta_{ab}(\sigma_{l+1}^m) = \Delta_{a-1,b-1}(\sigma_l),\ H_{ab}(\sigma_{l+1}^m) = H_{a-1,b-1}(\sigma_l) \text{ for } (a,b) \in I_{m+1,l+1}. \tag{4.38}$$

Next, by virtue of (4.34)–(4.36) and (4.17) we obtain for $(a,b) \in J_{1,m-1}^{m+1,l+1}$

$$\Delta_{ab}(\sigma_{l+1}^m) = [\delta_{a,b-1}(\sigma_l) + p_{l+1}]^+ = \tilde{\Delta}_{a,b-1}(\sigma_l), \tag{4.39}$$

and let us denote

$$\tilde{H}_{ab}(\sigma_l) = \min_{0 \le t \le \tilde{\Delta}_{ab}(\sigma_l)} \max\{g_{\sigma_l(a)}(t), f_{\sigma_l(b)}(\tilde{\Delta}_{ab}(\sigma_l) - t)\}. \tag{4.40}$$

Although $\tilde{\Delta}_{ab}(\sigma_l)$ and $\tilde{H}_{ab}(\sigma_l)$ have to be calculated for $(a,b) \in J_{1,m-1}^{ml}$, we decide to calculate $\tilde{H}_{ab}(\sigma_l)$ for all $(a,b) \in I_{1l}$ employing the obvious inclusion $J_{1,m-1}^{ml} \subseteq I_{1l}$, $m = 1,\dots,l+1$. This can be done at the beginning of the $l+1$-st iteration in a time $O(l^2\omega)$.

$H(\sigma_{l+1}^m)$ can be found from (4.14) by maximization $\max_{(a,b)\in I_{1,l+1}} H_{ab}(\sigma_{l+1}^m)$. Let us decompose the set $I_{1,l+1}$ into six subsets in the following way

$$I_{1,l+1} = I_{1,m-1} \cup I_{m+1,l+1} \cup J_{1,m-1}^{mm} \cup J_{mm}^{m+1,l+1} \cup J_{1,m-1}^{m+1,l+1} \cup \{(m,m)\} \tag{4.41}$$

and consider the problem of maximization separately, according to the sets listed [3]

Due to (4.37) we have

$$\max_{(a,b)\in I_{1,m-1}} H_{ab}(\sigma_{l+1}^m) = \max_{(a,b)\in I_{1,m-1}} H_{ab}(\sigma_l) = R_{m-1} \tag{4.42}$$

where

$$R_j = \max_{(a,b)\in I_{1j}} H_{ab}(\sigma_l). \tag{4.43}$$

Since $I_{1j} = I_{1,j-1} \cup J_{1,j-1}^{jj} \cup \{(j,j)\}$, then

$$\begin{aligned} R_j &= \max\{\max_{(a,b)\in I_{1,j-1}} H_{ab}(\sigma_l), \max_{(a,b)\in J_{1,j-1}^{jj}} H_{ab}(\sigma_l), H_{jj}(\sigma_l)\} \\ &= \max\{R_{j-1}, \max_{(a,b)\in J_{1,j-1}^{jj}} H_{ab}(\sigma_l), H_{jj}(\sigma_l)\},\ j = 1,\dots,l. \end{aligned} \tag{4.44}$$

Clearly, $R_0 = 0$ and $\left|J_{1,j-1}^{jj}\right| = j-1$. Observe that R_j, $j = 1,\dots,l$, does not depend on m and can be found recursively on the base of σ_l in a time $O(l^2\omega)$, only once at the beginning of $l+1$-st iteration.

[3] It is naturally assumed as $\max_{i\in A} a_i = 0$ if $A = \emptyset$.

The second problem of maximization, due to (4.38), can be written as

$$\max_{(a,b)\in I_{m+1,l+1}} H_{ab}(\sigma^m_{l+1}) = \max_{(a,b)\in I_{ml}} H_{ab}(\sigma_l) = Q_m \tag{4.45}$$

where

$$Q_j = \max_{(a,b)\in I_{jl}} H_{ab}(\sigma_l). \tag{4.46}$$

From definitions (4.15) and (4.33) we have $I_{jl} = I_{j+1,l} \cup J^{j+1,l}_{jj} \cup \{(j,j)\}$. In consequence, we obtain

$$Q_j = \max\{\max_{(a,b)\in I_{j+1,l}} H_{ab}(\sigma_l), \max_{(a,b)\in J^{j+1,l}_{jj}} H_{ab}(\sigma_l), H_{jj}(\sigma_l)\}$$

$$= \max\{Q_{j+1}, \max_{(a,b)\in J^{j+1,l}_{jj}} H_{ab}(\sigma_l), H_{jj}(\sigma_l)\},\ j = l, l-1, \ldots, 1. \tag{4.47}$$

Clearly, $Q_{l+1} = 0$ and $\left|J^{j+1,l}_{jj}\right| = l - j$. By analogy to R_j, we conclude that Q_j, $j = 1, \ldots, l$, can be found in a time $O(l^2\omega)$.

Maximizations on sets $J^{mm}_{1,m-1}$ and $J^{m+1,l+1}_{mm}$, see (4.41), can be performed in the time $O((m-1)\omega)$ and $O((l-m+1)\omega)$, respectively.

Next, consider the maximization on the set $J^{m+1,l+1}_{1,m-1}$. Observe that by (4.40) we have

$$\max_{(a,b)\in J^{m+1,l+1}_{1,m-1}} H_{ab}(\sigma^m_{l+1}) = \max_{1\le i\le m-1} \max_{(a,b)\in J^{m+1,l+1}_{ii}} H_{ab}(\sigma^m_{l+1})$$

$$= \max_{1\le i\le m-1} \max_{(a,b)\in J^{ml}_{ii}} \tilde{H}_{ab}(\sigma_l) = \max_{1\le i\le m-1} U_{im} = V_m, \tag{4.48}$$

where $V_1 = V_{l+1} = 0$ and

$$U_{ij} = \max_{(a,b)\in J^{jl}_{ii}} \tilde{H}_{ab}(\sigma_l),\ i = 1, \ldots, j-1,\ j = 2, \ldots, l. \tag{4.49}$$

The values U_{ij} can be found reccursively using the following dependence

$$U_{i,j-1} = \max_{(a,b)\in J^{j-1,l}_{ii}} \tilde{H}_{ab}(\sigma_l) = \max_{(a,b)\in J^{jl}_{ii}\cup\{(i,j-1)\}} \tilde{H}_{ab}(\sigma_l)$$

$$= \max\{\tilde{H}_{i,j-1}(\sigma_l), U_{ij}\},\ i = 1 \ldots j-1,\ j = l, l-1, \ldots, 2. \tag{4.50}$$

All values of U_{ij} can be found in a time $O(l^2\omega)$, whereas all V_j, $j = 1, \ldots, l+1$, in a time $O(l^2)$.

Finally, maximization on the set $I_{1,l+1}$ can be found as

$$H(\sigma_{l+1}^m) = \max_{(a,b)\in I_{1,l+1}} H_{ab}(\sigma_{l+1}^m) = \max\{R_{m-1}, Q_m,$$

$$\max_{(a,b)\in J_{1,m-1}^{mm}} H_{ab}(\sigma_{l+1}^m), \max_{(a,b)\in J_{mm}^{m,l+1}} H_{ab}(\sigma_{l+1}^m), V_m, H_{mm}(\sigma_{l+1}^m)\} \tag{4.51}$$

in $O(l\omega)$ time. Since the values R_j, Q_j, V_j are calculated in advance, hence all $H(\sigma_{l+1}^m)$, $m = 1, \ldots, l+1$ can be found in $O(l^2\omega)$ time.

At the end of this section we briefly summarize the main steps of algorithm INS/E. We generate a sequence of permutations in the same way as in INS. For the permutation σ_l we calculate additionally the values R_j, Q_j, V_j, $j = 1, \ldots, l$, from (4.44), (4.47), (4.48)–(4.50), respectively. Then we calculate the values $H(\sigma_{l+1}^m)$ using (4.51). The computational complexity of algorithm INS/E is $O(n^3\omega)$.

Local search

The local search method works in a neighbourhood $\mathcal{N}(\pi)$ of π, which contains a set of permutations obtained from π using for example job interchanges, pairwise interchanges or insertions. The neighbourhood is entirely searched in order to find the best permutation (in terms of the goal function value). Many recent researches have suggested that the so-called *insertion neighborhoods* are highly recommended for the problems with minimax criteria. These neighborhoods can be defined using the *insertion move*. The move (x, y) is a pair of positions $1 \leq x \leq n, 1 \leq y \leq n, x \neq y$. This move when applied to the permutation π generates new permutation $\pi_{(x,y)}$ in the following way: if $x < y$ then the job $\pi(x)$ is deleted from the position x, the jobs $\pi(x+1), \ldots, \pi(y)$ are shifted to the left by a single position and job $\pi(x)$ is inserted in the position y; if $x > y$ then the job $\pi(x)$ is deleted from the position x, the jobs $\pi(y), \ldots, \pi(x-1)$ are shifted to the right by a single position and the job $\pi(x)$ is inserted in the position y. For the problem $1||h_{\max}$, we propose some specific neighbourhood. Let $(u, v) \in I_{1n}$ be a pair which maximizes the formula (4.14) for the given π. The neighbourhood $\mathcal{U}(\pi)$ of π is defined as follows

$$\mathcal{U}(\pi) = \{\pi_{(x,v)} : u \leq x < v\} \cup \{\pi_{(x,u)} : u < x \leq v\}. \tag{4.52}$$

Exceptionally, if $v - u > 1$ then we set $\mathcal{U}(\pi) = \{(u, v)\}$. This form of the neighborhood is justified by the following theorem.

Property 4.2. For any permutation $\gamma = (\gamma(1), \ldots, \gamma(n))$ such that $\gamma(u') = \pi(u)$, $\gamma(v') = \pi(v)$, $u' \leq v'$, $\{\pi(u+1), \ldots, \pi(v-1)\} \subseteq \{\gamma(u'+1), \ldots, \gamma(v'-1\}$, we have $H(\gamma) \geq H(\pi)$. □

Proof. It is clear that $\sum_{s=u'}^{v'-1} p_{\gamma(s)} \geq \sum_{s=u}^{v-1} p_{\pi(s)}$. Therefore

$$\Delta_{u'v'}(\gamma) = [r_{\gamma(u')} - d_{\gamma(v')} + \sum_{s=u'}^{v'-1} p_{\gamma(s)}]^+ \geq [r_{\pi(u)} - d_{\pi(v)} + \sum_{s=u}^{v-1} p_{\pi(s)}]^+ = \Delta_{uv}(\pi). \tag{4.53}$$

In consequence, we obtain

$$H(\gamma) \geq H_{u'v'}(\gamma) = \min_{0 \leq t \leq \Delta_{u'v'}(\gamma)} \max\{g_{\gamma(u')}(t), f_{\gamma(v')}(\Delta_{u'v'}(\gamma) - t)\}$$

$$\geq \min_{0 \leq t \leq \Delta_{uv}(\pi)} \max\{g_{\pi(u)}(t), f_{\pi(v)}(\Delta_{uv}(\pi) - t)\} = H_{uv}(\pi) = H(\pi), \tag{4.54}$$

which completes the proof. ∎

From Property 4.2 it follows that insertion moves such that: (a) $x, y \in \{u+1, \ldots, v-1\}$, (b) $x \in \{1, \ldots, u-1\}$, $x < y$, (c) $x \in \{v+1, \ldots, n\}$, $y < x$, are "non-promising" if we take account of the immediate improvement. From among the remain ("promising") moves we selected a subset used in construction of the neighbourhood (4.52). This subset guarantees the *connectivity property* for $\mathcal{U}(\pi)$, which offers the "possibility" of finding an optimal solution by repetition of the local search method.

Property 4.3. For any $\pi^1 \in \Pi$ there exists a finite sequence of permutations $\pi^1, \ldots, \pi^r$ such that $\pi^{i+1} \in \mathcal{U}(\pi^i)$ for $i = 1, \ldots, r-1$, and π^r is an optimal permutation. □

Proof can be obtained by analogy to that from Section 3.1.

The neighbourhood $\mathcal{U}(\pi)$ contains at most $O(2(v-u-1))$ new permutations[4], and hence can be searched in a time $O((v-u)n^2\omega)$. In order to speed up searching, we propose some method of calculations' reduction.

The method will be outlined for *moves to the right*, i.e. (x, v), $u \leq x < v$, since *moves to the left* can be analysed by analogy. Denote the move performed by (x, v) and set $\sigma = \pi_{(x,v)}$. By the definition we have

$$\sigma(j) = \pi(j), \ j \in \{1 \ldots x-1\} \cup \{v+1, \ldots n\}, \tag{4.55}$$

$$\sigma(j) = \pi(j+1), \ x \leq j \leq v-1, \tag{4.56}$$

$$\sigma(v) = \pi(x). \tag{4.57}$$

[4]Two moves (x, y) and (y, x) provide the same permutations $\pi_{(x,y)} = \pi_{(y,x)}$. Therefore, in order to avoid redundancy, one of them will be eliminated from $\mathcal{U}(\pi)$.

In consequence of (4.55) and (4.56), we have

$$\Delta_{ab}(\sigma) = \Delta_{ab}(\pi),\ H_{ab}(\sigma) = H_{ab}(\pi) \text{ for } (a,b) \in I_{1,x-1} \cup I_{v+1,n}, \tag{4.58}$$

$$\Delta_{ab}(\sigma) = \Delta_{a+1,b+1}(\pi),\ H_{ab}(\sigma) = H_{a+1,b+1}(\pi) \text{ for } (a,b) \in I_{x,v-1}. \tag{4.59}$$

$H(\sigma)$ can be found from (4.14) by maximization $\max_{(a,b)\in I_{1n}} H_{ab}(\sigma)$. Let us decompose the set I_{1n} into subsets in the following way

$$I_{1n} = I_{1,x-1} \cup I_{v+1,n} \cup I_{x,v-1} \cup J_{1,x-1}^{v+1,n} \cup J_{1,v-1}^{vv} \cup J_{vv}^{v+1,n} \cup J_{1,x-1}^{x,v-1} \cup J_{x,v-1}^{v+1,n} \cup \{(v,v)\}, \tag{4.60}$$

and consider the problem of maximization separately, according to the sets listed.

Due to (4.55) we have

$$\max_{(a,b)\in I_{1,x-1}} H_{ab}(\sigma) = \max_{(a,b)\in I_{1,x-1}} H_{ab}(\pi) = R_{x-1}, \tag{4.61}$$

where R_j is defined by (4.43)–(4.44) (for π instead of σ_l). For all moves to the right we require the values R_j, $j = 1,\dots,v-1$, which can be found in a time $O((v-1)^2\omega)$.

Similarly, due to (4.55) we have

$$\max_{(a,b)\in I_{v+1,n}} H_{ab}(\sigma) = \max_{(a,b)\in I_{v+1,n}} H_{ab}(\pi) = Q_{v+1} \tag{4.62}$$

where Q_j is defined by (4.46)–(4.47). For all moves to the right only single value Q_{v+1} is realy needed, which can be found in a time $O((n-v)^2\omega)$. Note that, by symmetry, for moves to the left we will require the values Q_j, $j = u,\dots,v-1$.

Next, due to (4.56) we have

$$\max_{(a,b)\in I_{x,v-1}} H_{ab}(\sigma) = \max_{(a,b)\in I_{x+1,v}} H_{ab}(\pi) = P_{x+1}. \tag{4.63}$$

Since $I_{jv} = I_{j+1,v} \cup J_{jj}^{j+1,v} \cup \{(j,j)\}$, we obtain

$$P_j = \max_{(a,b)\in I_{jv}} H_{ab}(\pi) = \max\{\max_{(a,b)\in I_{j+1,v}} H_{ab}(\pi), \max_{(a,b)\in J_{jj}^{j+1,v}} H_{ab}(\pi), H_{j,j}(\pi)\}$$

$$= \max\{P_{j+1}, \max_{(a,b)\in J_{jj}^{j+1,v}} H_{ab}(\pi), H_{jj}(\pi)\},\ j = v, v-1,\dots,u \tag{4.64}$$

where $P_{v+1} = 0$. Since for all moves to the right we require the values P_j, $j = u,\dots,v$, the time needed is $O((u-v)^2\omega)$.

Now consider maximization on the set $J_{1,x-1}^{v+1,n}$. By virtue of (4.55) we have

$$\max_{(a,b)\in J_{1,x-1}^{v+1,n}} H_{ab}(\sigma) = \max_{(a,b)\in J_{1,x-1}^{v+1,n}} H_{ab}(\pi) = W_{x-1} \tag{4.65}$$

where

$$W_j = \max_{(a,b)\in J_{1j}^{v+1,n}} H_{ab}(\pi) = \max\{\max_{(a,b)\in J_{1,j-1}^{v+1,n}} H_{ab}(\pi), \max_{(a,b)\in J_{jj}^{v+1,n}} H_{ab}(\pi)\}$$

$$= \max\{W_{j-1}, \max_{(a,b)\in J_{jj}^{v+1,n}} H_{ab}(\pi)\}, \ j = 1,\ldots,v-1, \tag{4.66}$$

and $W_0 = 0$. The required values W_j, $j = 1,\ldots,v-1$ can be found in a time $O((v-1)^2\omega)$.

Maximizations on the sets $J_{1,v-1}^{vv}$ and $J_{vv}^{v+1,n}$ can be performed in a time $O((v-1)\omega)$ and $O((n-v)\omega)$, respectively. Unfortunately, maximizations on the sets $J_{1,x-1}^{x,v-1}$ and $J_{x,v-1}^{v+1,n}$ cannot be simplified and require $O((x-1)(v-x)\omega)$ and $O((v-x)(n-v)\omega)$ time.

Finally, we obtain

$$H(\sigma) = \max\{R_{x-1}, Q_{v+1}, P_{x+1}, W_{x-1}, \max_{(a,b)\in J_{1,v-1}^{vv}\cup J_{vv}^{v+1,n}} H_{ab}(\sigma),$$

$$\max_{(a,b)\in J_{1,x-1}^{x,v-1}\cup J_{x,v-1}^{v+1,n}} H_{ab}(\sigma), H_{vv}(\sigma)\} \tag{4.67}$$

At the end of this section we briefly summarize the main steps of the single neighbourhood search using moves to the right. For the permutation π we calculate additionally the values $(R_j, j = 1,\ldots,v-1)$, Q_{v+1}, $(P_j, j = u,\ldots,v)$, $(W_j, j = 1,\ldots,v-1)$ from (4.43)–(4.44), (4.46)–(4.47), (4.64), (4.66), respectively. Then we calculate the values $H(\sigma)$ using (4.67).

Descending search

It has been verified experimentally that the priority rules for the problems with irregular criteria have significantly worse intuitive justification and provide algorithms of poor performance. Threfore many authors have appended various descending search (DS) supporting proceduresm to the aforementioned rules. To evaluate the quality of such an approach to the problem stated, we also design an algorithm of this type.

The proposed DS algorithm starts with a given permutation π and searches at each iteration the neighbourhood $\mathcal{U}(\pi)$ of π. If the permutation better than $H(\pi)$ has been found, this permutation replaces π and the process of generating the neighbourhood is repeated, otherwise the algorithm ends its activity. The number of iterations performed by the algorithm DS is a priori unknown. The starting permutation for the algorithm DS can be found by any method.

Tabu search

In this section, we refer only to the basic version of this approach, see Section 2.3 for details. The procedure for solving our problem by means of the proposed algorithm consists in starting from an initial permutation π and searching through its neighbourhood $\mathcal{U}(\pi)$ for a permutation $\sigma \in \mathcal{U}(\pi)$ with the lowest cost $H(\sigma)$. Then the search repeats from a new starting permutation found as the best, and the process is continued. To avoid cycling, becoming trapped into a local optimum, and to guide the search to "good regions" of the solution space, a memory of the search history is introduced. We employ only the tabu list that records the selected attributes of subsequently visited permutations, treating them as a form of prohibition for the future search.

The used tabu list is represented by the cyclic list T of the length *maxt*. An element of T is a pair of jobs (g,h), $g,h \in N$. The list is initiated by empty elements. The new element introduced to T replaces the oldest one. Each move, performed from a processing order π, introduces to T the single element $(\pi(x+1),\pi(x))$ if the move (x,v) has been performed or $(\pi(x),\pi(x-1))$ if the move (x,u) has been performed. The move (g,h), $g<h$, has the tabu status if at least one pair $(\pi(g),\pi(k))$, $k=g+1,\ldots,h$, belongs to T. The move (g,h), $g>h$, has the tabu status if at least one pair $(\pi(k),\pi(g))$, $k=h,\ldots,g-1$, belongs to T.

Let π^* denote the best permutation found up to the current iteration. The move to be performed is selected as the best one among all non-prohibited moves, and prohibited moves with the cost lower than $H(\pi^*)$. If there are no moves satisfying the required property (a very rare situation) we add the empty move $(0,0)$ to T and next repeat the selection until a move has been chosen.

Algorithm stops its activity after the given number of iterations. The starting permutation for the algorithm TS can be found by any method.

4.2.3 Experimental analysis

Algorithms PA, INS, PA+DS, INS+DS, INS+TS have been coded in Pascal, run on PC 586/90 and tested on random instances. Algorithm INS was implemented in its efficient version INS/E. The symbol A+DS means that algorithm DS was started from a permutation found by algorithm A. Algorithm TS was run from permutation found by INS, with $maxt = 8$ and iteration limit 50 for $n \leq 50$ and 100 in other cases. Since there are no standard benchmarks for considered problem, we generated some tested instances in the following way: p_j, r_j, d_j, v_j, w_j as the integers with uniform distribution in intervals $[1,100]$, $[0,nk]$, $[0,nk]$, $[1,5]$, $[1,5]$, respectively where $k = 16,\ldots,25$. For each chosen $n = 10,20,40,80,100$ and $k = 16,\ldots,25$ a sample of 10 instances was generated. We found experimen-

tally that the chosen range of k provides particularly hard problems. Thus 500 instances were tested. For each instance and for each algorithm A the values of relative performance error

$$\eta^A = (H(\pi^A) - H)/H, \tag{4.68}$$

where $H = \min\{H(\pi^A) : A \in \{PA, INS, PA + DS, INS + DS, INS + TS\}$, were calculated. Algorithms were evaluated from two points of view: the mean relative error η^A and the mean running time. Results of comparison are shown in Tables 4.1–4.3. Table 4.1 present the mean running time for instances of the given size. Table 4.2 contains mean values of errors η^A, and mean numbers of neighbourhoods searched by DS in combination with INS and PA (denoted as $IT\,INS$ and $IT\,PA$, respectively). Table 4.3 repeats the results from Table 4.2 however, classified, according to the range k used in generation of random instances.

The results obtained are surprising. The classical priority rules tested are completely useless (mean error about 250 percent). Moreover, this error strongly increases if the size of instances increases. On the other hand, the insertion technique INS provides results with the mean error about 45 percent. This error also increases with n, but the increase is rather weak. A variation of the error is dependent on k, see Table 4.3. Although the error η^{INS} remains too high, if we take into account the priority algorithm PA, then it can be accepted as "relatively good solution obtained within a reasonable time". Two remain methods based on the descending search technique provide results of execellent quality (see η^A, $A \in \{INS + DS, PA + DS, INS + TS\}$ in Table 4.2), however $PA + DS$ runs in a longer time than $INS + DS$. Observe that in this case the error η^{INS+DS} weakly depends on the instance size. Algorithm DS started from PA, performs about twice iterations more and provides worse results than the combination $INS + DS$. Algorithm $INS + TS$ is the absolute champion with respect to the solution performance.

4.2.4 Conclusions

The experimental analysis performed shows that only insertion technique and local neighbourhood search technique are those which can be recommended to construction of approximation algorithms for problems with min-max irregular criteria. An approximation algorithm based on these approaches has been provided and experimentally shown to be much better than other known approaches when both running time and solution quality are taken into account.

The appraoch proposed can be extended in an easy way to a single production cell model as well as problems with more general production structures,

Table 4.1: Mean solution times (in sec) on a PC 586/90 (each entry refers to 100 instances of the given size)

	CPU time		Neighbourhoods		
n	INS	DS	$IT\,INS$	$IT\,PA$	$IT\,TS$
10	0.01	0.00	2.03	5.32	50
20	0.01	0.01	4.86	12.39	50
30	0.06	0.05	9.43	21.21	50
40	0.12	0.11	14.25	30.58	50
50	0.23	0.22	19.92	41.08	50
60	0.41	0.39	25.76	49.85	100
70	0.65	0.62	29.90	59.92	100
80	1.01	0.93	37.09	70.94	100
90	1.41	1.33	43.43	80.71	100
100	1.91	1.80	51.45	94.30	100

INS : mean time of INS,
DS : mean time of a single search of the neighbourhood $\mathcal{N}(\pi)$,
$IT\,INS$: mean number of neighbourhoods searched by algorithm INS+DS,
$IT\,PA$: mean number of neighbourhoods searched by algorithm PA+DS,
$IT\,TS$: the number of neighbourhoods searched by algorithm INS+TS.

Table 4.2: Mean relative error of algorithms (each entry refers to 100 instances of the given size)

	Mean η^A				
n	PA	INS	$PA+DS$	$INS+DS$	$INS+TS$
10	133.27	13.33	0.91	1.98	0.00
20	191.36	27.43	0.77	2.21	0.00
30	233.86	36.02	1.22	2.30	0.00
40	242.67	43.96	2.05	1.72	0.00
50	259.80	47.55	3.52	1.63	0.06
60	262.62	52.14	3.03	2.17	0.01
70	271.52	53.20	4.09	3.04	0.00
80	274.06	59.00	4.69	3.88	0.00
90	284.95	59.18	7.59	3.40	0.00
100	289.63	61.27	6.65	2.90	0.00
all	244.37	45.31	3.45	2.52	0.01

Table 4.3: Comparison of algorithm performance in groups of random instances for $k = 16 \ldots 26$; each entry refers to 80 instances, 10 for each size n=10, 20, 30, 40, 50, 60, 80, 100

	Mean η^A				
k	PA	INS	$PA+DS$	$INS+DS$	$INS+TS$
16	204.01	29.67	2.63	1.93	0.03
17	198.71	29.90	2.92	1.26	0.00
18	199.95	31.67	2.32	2.68	0.00
19	204.35	36.60	3.69	2.07	0.02
20	210.40	33.14	3.93	2.20	0.00
21	191.17	36.86	1.24	1.27	0.00
22	195.79	39.46	2.35	2.31	0.01
23	190.90	40.20	3.16	2.09	0.00
24	180.52	39.26	1.57	2.05	0.00
25	191.41	41.77	2.04	2.59	0.00

see Grabowski and Smutnicki [110], Smutnicki [291, 299, 300]. These extensions preserve the reduced neighbourhood structure, the possibility of accelerating the search and the computational complexity $O(n^2\omega)$ of finding the criteria value for a given solution [5]. So really, the single neighbourhood search is still considered as disquitely high that makes hard the application of this algorithm to greater n. One can hope that the approach of Garey et al. [78] can be extended and applied to this case in order to reduce this cost to $O(nlogn)$ per single solution (at least for some special cases). Futher research need to be made in this subject.

4.3 Total and extended E/T penalties

To show how other E/T penalties can be embedded in JIT systems, we consider the already known single-cell performance delivery model, however with quite different penalty function.

Consider a cell having m machines, and let $\mathcal{M} = \{1, \ldots, m\}$ be the set of these machines. The set of n parts $\mathcal{N} = \{1, 2, \ldots, n\}$ has to be processed by this cell. Each part $j \in \mathcal{N}$ requires a single machine for processing, owns the processing time $p_{ij} \geq 0$ on machine $i \in \mathcal{M}$, and has the delivery interval $[r_j, d_j]$ within which this part has to be completed in the ideal case. Each part can be performed on any machine from $\mathcal{M}$. Once started, a part cannot be interrupted. Each machine can execute at most one part at a time, each part can be processed

[5] This cost is $O(n)$ for most of classic problems with a *regular* criterion.

at most on one machine at a time. A *feasible schedule* is defined by a couple of vectors (S,T), $S=(S_1,\ldots,S_n)$, $T=(t_1,\ldots,t_n)$, such that the above constraints are satisfied, where the part j is started on machine $t_j \in \mathcal{M}$ at time $S_j \geq 0$ and is completed at the time $C_j = S_j + p_{t_j j}$.

In this section we use the measure based on total weighted deviation from delivery interval, namely

$$\sum_{j\in\mathcal{N}}(v_j[a_j - C_j]^+ + w_j[C_j - b_j]^+),. \tag{4.69}$$

We wish to find the feasible schedule that minimises (4.69). Note, if $a_j = b_j$ and $v_j = w_j$ the measure (4.69) can be simply written as $\sum_{j\in\mathcal{N}} w_j\,|C_j - b_j|$. Garey et al. [78] made additional assumption that w_j =const and then they considered the single machine problem with the latter measure, see problem with *total discrepancy*. We also refer to Section 3.1 for the notions: machine workload $(\mathcal{N}_i, i\in\mathcal{M})$, machine processing orders $(\pi_i, i\in\mathcal{M})$, overall processing order π, set of all processing orders Π. The feasible schedule (S,T) generated by π implies $t_j = i$, $j=1,\ldots,n_i$, $i\in\mathcal{M}$. Appropriate starting times $S=(S_1,\ldots,S_n)$ follow from the optimisation problem

$$H(\pi) = \min_{S\in\mathcal{S}(\pi)} \sum_{j\in\mathcal{N}}(v_j[a_j - C_j]^+ + w_j[C_j - b_j]^+) \tag{4.70}$$

with the set $\mathcal{S}(\pi)$ introduced by constraints (3.4). In the context of consideration from Section 3.1, the problem can be decomposed into two subproblems: (1) find the processing order $\pi\in\Pi$, (2) for the given π find the minimal criterion value by solving (4.70), (3.4). Clearly, the best processing order from Π is looked for. However, each such a processing order need to be evaluated in more complex manner. Similarly as in the Section 3.1, for fixed π the problem can be decomposed into m independent single-machine subproblems. Next, each single-machine subproblem can be written as a LP problem and then solved. However, for the single machine case there exists a polynomial-time algorithm, Smutnicki [291]. Also the method provided by Garey et al. [78] can be adapted to this case, that ensures the competitive computational complexity $O(n_i \log n_i)$ for a given π_i.

As to the searching for the optimal (best) processing order, one can propose a local search algorithm, for example by using SA method with the neighbourhood $\mathcal{V}(\pi)$. Indeed, it should work. However, the expected significant reduction of the neighbourhood size is not possible in this case since neither Property 4.2 nor its extensions hold. Hence, having in mind a disadvantegous influence of the too big move set on the behaviour of SA algorithm, see Section 3.1, we propose to

employ the move set

$$\mathcal{U}(\pi) = \bigcup_{j \in \mathcal{N}} \bigcup_{i \in \mathcal{M}} \mathcal{U}_{ji}(\pi) \tag{4.71}$$

where the sets $\mathcal{U}_{ji}(\pi)$ are defined in the following manner. If $i = t_j > 1$ then $\mathcal{U}_{ji} = \{(j, i, t_j - 1)\}$, if $i = t_j = 1$ then $\mathcal{U}_{ji} = \emptyset$. If $i \neq t_j$ then $\mathcal{U}_{ji}(\pi)$ contains only the single move (j, i, y) selected from the set $\mathcal{V}_{ji}(\pi)$ so that $S_{\pi_i(y-1)} \leq (r_j + d_j)/2 \leq S_{\pi_i(y)}$, where S_j are job starting times found for the processing order π. The set $\mathcal{U}(\pi)$ contains $mn - 1$ moves, posseses the connectivity property (its proof can be performed by analogy to that of Property 3.10) and is significantly less than $\mathcal{V}(\pi)$.

Also TS with $\mathcal{V}(\pi)$ is not too promising since it requires a time $O(n^3 log n)$ per single neighbourhood search. TS with $\mathcal{U}(\pi)$ is much more justified from this point of view since it requires only $O(n^2 m log n)$ time per neighbourhood. Unfortunately, currently there are not good and simple ideas how to reduce superfluous calculations in order to accelerate the TS speed. Further research need to be made in this subject since these computational complexities are considered as disquitely high in the context of local search methods.

Extensions of this approach to more complex (general) production structures immediately lead to LP models (assuming linear penalty function) constructed separately for each fixed processing order, see for example Toczyłowski [319]. Since many processing orders need to be considered by means of a local search method, an algorithm of this type must solve at each iteration an LP program (at least one per neighbourhood for SA method, and many of them for TS method) of the size proportional to the number of parts that flow through the system. Assuming that a move introduces only a small perturbation of the current processing order, none can hope that the new LP problem (for the descending processing order) can easily be found by modyfing solution of the oldest one. Even if such a method is found, the cost of the local neighbourhood search will remain large. From this point of view, the SA or SJ method is much more preferable than TS.

The decision which type of the E/T penalty can be applied in each particular case depends finally on the user. Nevertheless, one can observe that problems with total E/T penalties are much more troublesome than those with max E/T penalties from many points of view. On the other hand, total penalty may be better economically interpreted (WIP, storage, capital costs, etc.). Let us come back to the fundamental role of the E/T penalty function, which is guidance of solutions towards the target of meeting exactly all delivery dates required. Consider the problem of minimizing n independent criteria $h_j(S_j)$ each with the target zero. To obtain the effective solutions (Pareto optimal), a scalarisation of these objective functions is required. Because of nonconvexity of the problem,

the function e.g.

$$\max_{j\in\mathcal{N}} \rho_j h_j(S_j) + \epsilon \sum_{j\in\mathcal{N}} h_j(S_j) \tag{4.72}$$

can be proposed. Observe, the problem obtained in this way is relative to this with *sum* and that with *max* E/T penalties. Although the goal function (4.72) is much more complicated than each of its term, assuming ϵ sufficiently small, we can propose a two-phase solution algorithm. In the first phase, there is searched the good solution of the problem taking into account only *max* term. To this end the method from Section 4.2 can be applied. Next, starting from the solution obtained in the first phase, search is continued, however with the use of the whole criterion (4.72).

Chapter 5

Work in process reduction

5.1 Introduction

The philosophy of JIT forces the reduction of wastes associated with holding semi-products and products in the system. This can be obtained among others by the reduction of times lost by items (products) waiting in queues for processing centres. Each product waiting in the system creates superfluous in-process inventory, which the JIT strategy should minimise. If each product is manufactured in unique way (on an appropriate machine), then instead of reducing the volume of in-process inventory, we can reduce the WIP cost assuming that this cost depends on a product (it is constant for a product but varying for various products) and on a product waiting time (it linearly depends on the waiting time). Moreover, we assume the strategy of delivery in which each product must be delivered not later than the given time moment. In this chapter we present an approach, using a case study, which can be applied in order to solve the selected problems of this type.

Consider a flow manufacturing line having m stages, single machine at each stage and FIFO service rule in queues for all machines. This model known in the literature as flow-shop problem has good industrial application in many chemical branches [330, 331]. This line should deliver n different products and each of them has to meet its due date d_j. (Cyclic manufacturing and repetitive products can be modelled and solved due to high generality of this model.) Each product flows downstream the machines $1, \ldots, m$ in that order, and the product j spends the time p_{ij} on the machine i. Each machine can execute at most one product at a time. Processing of a product on a machine cannot be interrupted. We wish to find starting the times S_{ij} of products on machines such that the total cost of holding work-in-process is minimal. This cost is simply equal to the total (weighted) time spent by all products in the system, Fig. 5.1. Observe that the

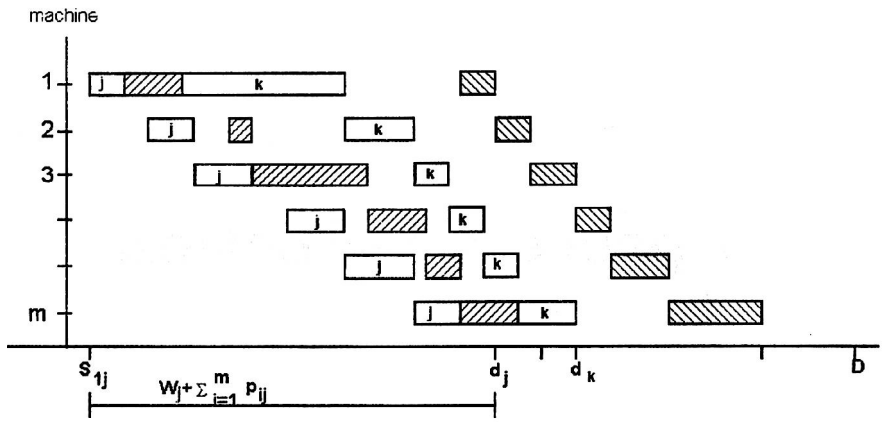

Figure 5.1: JIT scheduling of flow line

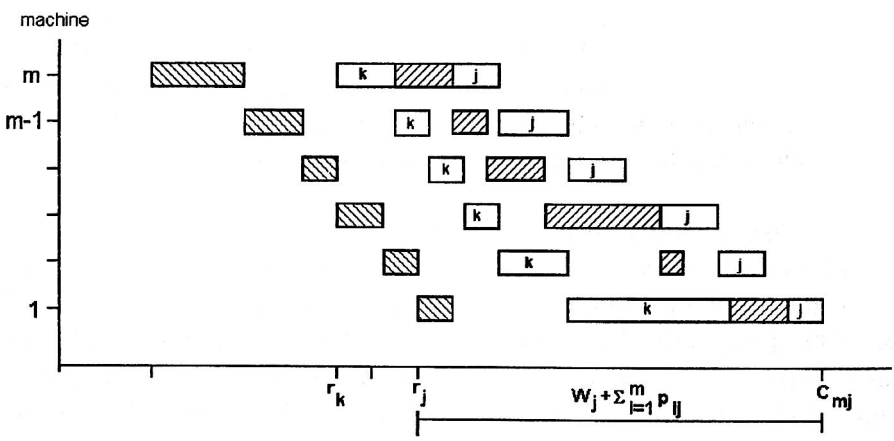

Figure 5.2: Forward-backward symmetry of flow line scheduling

optimal schedule need be neither compact nor shifted to the left on the time axis.

We will show that this problem can be analysed and solved using tools from OR field. First, employing its forward-backward symmetry, we will show the equivalence of this problem to some classic scheduling problems. Next, we provide a wide theoretical analysis of algorithms recommended for the latter problem. A new related theoretical results will also be pointed out.

On the base of original data n, m, p_{ij}, d_j, we introduce a new problem with data indexed by "prime". We define a problem having $m' = m$ machines, $n' = n$ products, processing times $p'_{ij} = p_{m-i+1,j}$ and product ready times $r'_j = D - d_j$, where $D = \max_{1 \leq j \leq n} d_j$. One can say that the latter problem has been obtained by "inverting" the former one, see Fig. 5.2. The original waiting time of the product j is equal to $W_j = d_j - S_{1j} - \sum_{i=1}^{m} p_{i,j}$, whereas in the new problem is equal to $W'_j = C'_{mj} - r'_j - \sum_{i=1}^{m} p'_{i,j}$, where C'_{ij} is the completion time of the product j on machine i. Moreover we clearly have $W'_j = W_j$. Hence, instead of minimising the (weighted) sum of W_j, we can minimise the (weighted) sum of W'_j in the transformed problem. The latter problem is known in the scheduling theory as the permutation flow-shop problem with ready times to minimise total (weighted) completion times, and can be transformed to the problem of minimising (weighted) completion times without ready times, because the sum of ready times provides a constant not having any influence on the optimal processing order. Finally, we obtain the equivalence between the former problem and the permutation flow-shop problem with the weighted sum of completion times (if costs of holding are different for various products) and the mean completion time (if cost of holding is the same for all products). These two pure problems will be discussed in detail in the next sections.

5.2 The flow line scheduling problem

The considered flow manufacturing line with m stages, single machine at each stage and FIFO service rule in queues for all machines can be modelled by the problem called in the scheduling theory the permutation flow-shop problem. It has the following mathematical formulation expressed by conventional notions. The set of n different jobs should be processed on m different machines. Each job $j, j \in J = \{1, \ldots, n\}$ passes through the machines $1, 2, \ldots, m$ in that order and requires an uninterrupted time p_{ij} for processing on the machine i, $i = 1, .., m$ [1]. Machine i, $i = 1, .., m$, can execute at most one job at a time and each machine processes the jobs in the same order. We wish to find the optimal job processing

[1] It is assumed that a zero processing time on a machine corresponds to a job performed by that machine within infinitesimal time.

order (represented by a permutation π on the job set J) which minimises the given criterion K.

From among many regular criteria defined for scheduling problems we consider only these used in the analysed algorithms, i.e. the weighted sum of completion times (C_{sum}), the mean completion time ($\overline{C}_{\text{sum}}$), and the makespan ($C_{\max}$), defined as follows

$$C_{\text{sum}}(\pi) = \sum_{j=1}^{n} w_{\pi(j)} C_{m\pi(j)}, \tag{5.1}$$

$$C_{\max}(\pi) = \max_{1 \le j \le n} C_{m\pi(j)}, \tag{5.2}$$

where

$$C_{i\pi(j)} = \max_{1 \le j_0 \le j_1 \le \ldots \le j_{i-1} \le j_i \le j} \sum_{s=1}^{i} \sum_{t=j_{s-1}}^{j_s} p_{s\pi(t)} \tag{5.3}$$

is the completion time of the job $\pi(j)$ on the machine i, $j = 1, \ldots, n$, $i = 1, \ldots, m$, see, e.g. [99]. The criterion $\overline{C}_{\text{sum}}$ is a special case of (5.1) such that $w_j = 1/n, j = 1, \ldots, n$. Note that (5.1)–(5.2) suffice to model a fairly broad class of scheduling criteria, e.g. these based on completion times, flow times (assuming that all jobs are available at time zero), job waiting times, machine idle times, machine utilisation, etc.

5.3 Algorithms performance

The stated problem is strongly NP-hard for $C_{\max}$ ($m \ge 3$) and $\overline{C}_{\text{sum}}, C_{\text{sum}}$ ($m \ge 2$) criteria, therefore a few exact and a lot of approximation algorithms have been developed to provide a good solution in a short time. Almost all exact algorithms are based on various B&B schemes, although few MILP models have also been considered without a great success. In practice, these algorithms can solve problems up to 50 jobs and 5 machines in a reasonable time [2]. Therefore, approximation methods remain still the most popular solution methods. For the criterion $C_{\max}$ there are *constructive methods* [35, 52, 124, 148, 212, 239, 264], and *improvement methods* [43, 52, 59, 140, 153, 231, 234, 237, 262, 314, 334, 336]. The respective methods for C_{sum} and $\overline{C}_{\text{sum}}$ one can find in [79, 140, 141, 144, 164, 183, 203, 204, 253, 254, 255, 256, 257, 58]. Two last papers deal with a multiple objective version of the problem – they consider simultaneously $C_{\max}$ and $\overline{C}_{\text{sum}}$, as well as other scheduling criteria.

Experimental analyses have been done for all afore mentioned algorithms, however the worst-case analyses have been performed only for the constructive

[2] Some problems having 50 jobs and 5 machines remain still too hard.

algorithms CDS [35], RA [52], P [239], RS [264], NEH [212], HR [148], G [124], TG [183], HK [144] and for the improvement algorithms $RACS$ and $RAES$ [52], see papers [219]–[228], [264], or Table 5.6. Besides theoretical implications, the latter analysis confirms the known experimental result that the *insertion algorithm* NEH can be considered as the champion among constructive algorithms with the criterion $C_{\max}$. Based on conclusions derived from the insertion approach, there have been proposed several improvement algorithms with very good performance verified in computer test on standard benchmarks up to 10,000 operations.

Known approximation algorithms recommended for the permutation flow-shop problem with the weighted sum of completion times have the worst-case performance ratio indefinite, or definite but not limited by a function of the instance size. We propose three new algorithms with these ratios equal m, $\lceil m/2 \rceil \rho_2$, and $\lceil m/k \rceil \rho_k$, where ρ_k is the worst-case performance ratio of an algorithm solving an auxiliary k-machine problem. Some results of the worst-case analysis for several other approximation algorithms with various scheduling criteria have been also presented. The results obtained supplement research from papers [219]–[228].

5.4 Algorithms

In this section, we briefly outline only these known algorithms which will be analysed in the next sections. New algorithms will be presented together with their analysis.

Many approximation methods developed for flow-shop problems refer to *job waiting times*, *delays between jobs*, *gaps*, *job matching*, or *job fitness*, e.g. the improving algorithm HC proposed in [140] for the permutation flow-shop problem with a general scheduling criterion, and experimentally investigated using criteria of the mean flow time, mean utilisation and makespan. This algorithm is based on a *gap* notion. Although, in our opinion, the philosophy of gaps has its origin in makespan problems, HC has been formulated and recommended for *any regular criterion*. The overall revised gap is defined as $d_{ij}^R = \Sigma_{k=1}^{m-1} d_{ij}^k \delta_{ij}^k$, where $d_{ij}^k = p_{k+1,i} - p_{kj}$ is the gap and

$$\delta_{ij}^k = \begin{cases} 1 & if \quad d_{ij}^k \geq 0 \\ 0.9\frac{m-k-1}{m-2} + 0.1 & if \quad d_{ij}^k < 0 \end{cases} \tag{5.4}$$

is the discount factor, $k = 1, \ldots, m-1$, $i, j = 1, \ldots, n$. Partitioning the set $\{1, \ldots, m-1\}$ into subsets $M_{ij} = \{1 \leq k < m : p_{k+1,i} \geq p_{kj}\}$ and $L_{ij} = \{1 \leq k < m : p_{k+1,i} < p_{kj}\}$ we can rewrite d_{ij}^R in another equivalent form that will be

more convenient for our applications

$$d_{ij}^R = \sum_{k\in M_{ij}} d_{ij}^k + \frac{0.9}{m-2}\sum_{k\in L_{ij}} d_{ij}^k(m-2-k+1) + 0.1\sum_{k\in L_{ij}} d_{ij}^k$$

$$= \sum_{k\in M_{ij}\cup L_{ij}} d_{ij}^k - \frac{0.9}{m-2}\sum_{k\in L_{ij}} d_{ij}^k(k-1) = \sum_{k=1}^{m-1} p_{k+1,i} - \sum_{k=1}^{m-1} p_{kj}$$

$$- \frac{0.9}{m-2}\sum_{k\in L_{ij}} d_{ij}^k(k-1) = b_i^{m-1} - a_j^{m-1} - \frac{0.9}{m-2}\sum_{k\in L_{ij}} (k-1)(p_{k+1,i} - p_{kj}) \quad (5.5)$$

where $a_t^{m-1} = \sum_{s=1}^{m-1} p_{st}, b_t^{m-1} = \sum_{s=1}^{m-1} p_{m-s+1,t}$, $t = 1,\ldots,n$ are processing times of the $(m-1)$-th auxiliary two-machine flow-shop employed by the algorithm CDS. Observe that the last sum in (5.5) can be performed over the redefined set $L_{ij} = \{1 < k < m : p_{k+1,i} < p_{kj}\}$ since the element for $k = 1$ has the weight of zero. Algorithm HC can be written in the following compact form. Let $\pi = (\pi(1),..,\pi(n))$ be an initial permutation on J. Denote by $\pi_{(x,y)}$ the permutation obtained from π by interchanging jobs in the positions x and y. Set initially $a = 1, b = n$.

Algorithm HC

1. If $b = a + 2$ then set $\pi^{HC} := \pi$ and stop.

2. Find values X, Y and indices g, h $(a < g < b, a < h < b)$ such that $X = d^R_{\pi(a)\pi(g)} = \max\{d^R_{\pi(a)\pi(j)} : a < j < b\}$, $Y = d_{\pi(h)\pi(b)} = \min\{d^R_{\pi(j)\pi(b)} : a < j < b\}$.

3. If $(((X > 0) \vee (Y < 0)) \wedge (|X| > |Y|)) \vee ((X < 0) \wedge (Y > 0) \wedge (|X| \leq |Y|))$ then go to step 4, otherwise go to step 5.

4. Set $a := a + 1$. If $K(\pi_{(a,g)}) < K(\pi)$ then set $\pi := \pi_{(a,g)}$. Go to step 1.

5. Set $b := b - 1$. If $K(\pi_{(h,b)}) < K(\pi)$ then set $\pi := \pi_{(h,b)}$. Go to step 1.

At each iteration of HC we need $2(b-a-1)$ values of $d^R_{\pi(a)\pi(j)}$ and $d^R_{\pi(j)\pi(b)}$, $j = a+1,\ldots,b-1$. Since in every iteration exactly one of two indices a or b changes the value, the total number of the values d_{ij}^R used in all iterations is equal to $(n-2)+(n-3)+\ldots+1 = (n^2-n-2)/2$. All these values can be calculated on request. The algorithm HC has the computational complexity $O(n^2m)$ [3] and requires an auxiliary approximation algorithm for preparing the

[3]The efficient implementation of NEH is also $O(n^2m)$, [314]. NEH shows in experiments very good performance for $C_{\max}$ as well as for $\overline{C}_{\text{sum}}$.

initial permutation. The original version of HC uses CDS (the commonly used algorithm) with the computational complexity $O(mn\log n + m^2n)$ at the start.

CDS: generates the permutation π^{CDS} selected from among $m-1$ auxiliary permuatations such that $C_{\max}(\pi^{CDS}) = \min_{\pi\in\{\pi^1,\ldots,\pi^{m-1}\}} C_{\max}(\pi)$, where π^k is the optimal permutation of an auxiliary two-machine flow-shop problem with the processing times $a_j^k = \sum_{i=1}^{i} p_{ij}$, $b_j^k = \sum_{i=m-k+1}^{m} p_{ij}$, $j = 1,\ldots,n$, on machines 1 and 2, respectively, $k = 1,\ldots,m-1$.

Any improvement algorithm (thus also HC) can be modified in several ways. First, using other quick constructive algorithms for generating the initial permutation. To this end, there are proposed either special priority indices [141, 255], or already known priority functions [124, 239], where a permutation is formed by arranging jobs in nondecreasing order of priority values

H : $\nu_j = \sum_{i=1}^{m}(m-i+1)p_{ij}$,

P : $\omega_j = \sum_{i=1}^{m}(m-2i+1)p_{ij}$,

G : $\beta_j = A_j/\min_{1\le i\le m-1}(p_{ij}+p_{i+1,j})$, where $A_j = -1$ if $p_{1j} \le p_{mj}$ and $A_j = 1$ in other cases,

or other constructive algorithms mentioned in Section 5.4, e.g. [52]

RA: find a permutation which is the optimal processing order of the auxiliary two-machine flow-shop problem with the processing times $a_j = \sum_{i=1}^{m}(m-i+1)p_{ij}$, $b_j = \sum_{i=1}^{m} ip_{ij}$ on machines 1 and 2, respectively, and the makespan criterion.

Algorithms G, P and RA have the computational complexity $O(nm + n\log n)$. The similarity between H and the algorithms P, RA, HR [4] has been pointed out in [141]. Second, a supporting improvement procedure is usually added at the end of the algorithm. There are few, well known types of these procedures, namely: interchanges of the adjacent pairs of jobs (API), interchanges of the non-adjacent pairs of jobs (NPI), and insertions (INS) [169]. Although API reveals serious drawbacks for the criterion $C_{\max}$, [226], it still remains the most popular method of improvement, chiefly for the criteria other than $C_{\max}$. (INS and some of their refined variants are considered as the most recommended for $C_{\max}$ [231, 314].)

Other approximation algorithms, called hereafter $RC1$, $RC2$, $RC3$, RCo, have been proposed in [253, 254] for the problem with the criterion $\overline{C}_{\text{sum}}$. Algorithms $RC1$, $RC2$, $RC3$ [253] can be interpreted as dynamic priority rules which

[4] Algorithm HR is a slight modification of P.

operate on the sets of scheduled (S) and unscheduled ($U = J\backslash S$) jobs. Jobs from the set S form a partial permutation σ, and let $C_{i\sigma(d)}$ denote the completion time of the last job from S on the machine $i, i = 1, \ldots, m, d = |S|$. The partial permutation σj, where $j \in U$, provides completion times of this job $C_{1j} = C_{1\sigma(d)} + p_{1j}$, $C_{ij} = \max\{C_{i\sigma(d)}, C_{i-1,j}\} + p_{ij}$, $i = 2, \ldots, m$,. Each of these algorithms select a job $k \in U$ to follow σ with the least appropriate priority value:

$RC1 : \omega'_k = \sum_{i=2}^{m} \max\{C_{i-1,k} - C_{i,\sigma(d)}, 0\}$,

$RC2 : \omega''_k = \sum_{i=2}^{m} |C_{i-1,k} - C_{i,\sigma(d)}|$,

$RC3 : \omega'''_k = \sum_{i=2}^{m} |C_{i-1,k} - C_{i,\sigma(d)}| + \sum_{i=1}^{m} C_{i,k}$.

Any ties are broken by choosing the job having the least completion time at the last machine. The algorithm RCo [254] employs a set of static priority rules.

RCo : generates the permutation π^{RCo} selected from among m auxiliary permutations such that $\overline{C}_{\text{sum}}(\pi^{RCo}) = \min_{\pi \in \{\pi^1, \ldots, \pi^m\}} \overline{C}_{\text{sum}}(\pi)$, where π^k is obtained by arranging jobs in nondecreasing order of priority values $\omega^o_{kj} = \sum_{i=1}^{m} \min\{m - i + 1, m - k + 1\} p_{ij}$.

Algorithms $RC1$, $RC2$, $RC3$ have the computational complexity $O(n^2 m)$, when RCo is $O(nm^2)$. Algorithm $RC1$ is a special case of *minimising between job delays* algorithm $KS2$ from [164] [5]. In the cited paper, to ensure a better solution performance, each job is additionally and successively considered as the first job in the primal partial sequence, i.e. the algorithm is repeated n times, each time starting from $\sigma = (j)$ for successive $j = 1, \ldots, n$.

We will also consider a very recently proposed algorithm R for a multiobjective problem with $C_{\max}$ and $\overline{C}_{\text{sum}}$ criteria [257]. This algorithm consists of three phases: (1) it starts from π^{CDS}, (2) it applies API procedure to improve π^{CDS} using criterion $C_{\max}$, and (3) it applies restricted API procedure which accepts only these solutions that improve either $C_{\max}$ or $\overline{C}_{\text{sum}}$. Details of the phase (3) will not be discussed here since, as we will show next, they are of no significance in its worst-case analysis.

In the sequel, a composition of algorithms A and B, such that the algorithm B starts from a permutation generated by the algorithm A, will be denoted by $A + B$.

5.5 The worst-case analysis

The data $n, m, (p_{ij}, i = 1, \ldots, m, j = 1, \ldots, n), (w_j, j = 1, \ldots, n)$, specify an instance Z of the problem. Denote by Π the set of all permutations on J, by

[5] There are five algorithms in this paper, all based on job delays.

$K(\pi; Z)$ the value of a criterion K for the job processing order π and the instance Z. The processing order $\pi^* \in \Pi$ such that $K(\pi^*; Z) = \min_{\pi \in \Pi} K(\pi; Z)$ is called the optimal processing order. Let $\pi^A \in \Pi$ denote a permutation generated by an algorithm A. The worst-case performance ratio of the algorithm A with the schedule criterion K is defined as $\eta^A(K) = \min\{y : K(\pi^A; Z)/K(\pi^*; Z) \leq y$, for each instance $Z\}$. In the sequel, the argument Z will be omitted in $K(\pi^*; Z)$ and $K(\pi^A; Z)$ if it is not necessary.

We will use some simplified notation analysing an instance Z. If the set J consists of $g \geq 1$ subsets (groups) $J_1, J_2, \ldots, J_g$ of identical jobs we assume that each job from a subset J_j is indexed by the same index j, $j = 1, 2, \ldots, g$. In such a case, the job processing order will be considered as a permutation with repetitions. Therefore, let π_I denote a permutation on a set $I \subset \{1, 2, \ldots, g\}$. We employ the symbol $(\pi_I)_s$ denoting the s-times concatenation of the permutation π_I. To simplify, the index I in π_I will be omitted if $I = J$. Denote also by $\lceil x \rceil$ the nearest integer not smaller than x.

5.5.1 The weighted sum of completion times criterion

The aim of this section is to analyse the worst behaviour of various known and some new algorithms for the criterion C_{sum}. The following general property provides a reference level for these evaluations.

Property 5.1. For any $\pi \in \Pi$ and Z, we have

$$C_{\text{sum}}(\pi; Z)/C_{\text{sum}}(\pi^*; Z) \leq 1 + (n-1)(\overline{w}/\underline{w}) \tag{5.6}$$

where

$$\underline{w} = \min_{j \in J} w_j, \qquad \overline{w} = \max_{j \in J} w_j. \tag{5.7}$$

□

Proof. Let π^* be the optimal processing order for the criterion C_{sum}. Applying the simple job-based bound $C_{mj} \geq \sum_{i=1}^{m} p_{ij}$ we get a lower bound of $C_{\text{sum}}(\pi^*)$

$$C_{\text{sum}}(\pi^*) \geq \sum_{j=1}^{n} w_{\pi^*(j)} \sum_{i=1}^{m} p_{i\pi^*(j)} \geq \underline{w} \sum_{j=1}^{n} \sum_{i=1}^{m} p_{ij}. \tag{5.8}$$

Next, using consecutively (5.1), (5.3) and (5.8) we obtain the following sequence of inequalities

$$C_{\text{sum}}(\pi) = \sum_{j=1}^{n} w_{\pi(j)} \max_{1 \leq j_0 \leq j_1 \leq \cdots \leq j_m \leq j} \sum_{s=1}^{m} \sum_{t=j_{s-1}}^{j_s} p_{s\pi(t)} \leq \sum_{j=1}^{n} w_{\pi(j)} \sum_{s=1}^{m} \sum_{t=1}^{j} p_{s\pi(t)}$$

$$\leq \sum_{j=1}^{n} w_{\pi(j)} \sum_{s=1}^{m}\sum_{t=1}^{n} p_{st} = (\sum_{s=1}^{m}\sum_{t=1}^{n} p_{st})(\sum_{j=1}^{n} w_{\pi(j)}) \leq \frac{C_{\text{sum}}(\pi^*)}{\underline{w}} \cdot ((n-1)\overline{w} + \underline{w})$$

which completes the proof. ∎

The value $1+(n-1)(\overline{w}/\underline{w})$ can be arbitrarily large and equals n if w_j=const, $j \in J$. Using Property 5.1 we can derive the worst-case performance ratio for the primal class of algorithms.

Theorem 5.1. For $A \in \{CDS, G, P, RA\}$ we have the tight bound

$$\eta^{A+HC}(C_{\text{sum}}) \leq \eta^{A}(C_{\text{sum}}) \leq 1 + (n-1)(\overline{w}/\underline{w}). \tag{5.9}$$

Proof. The upper bound in (5.9) follows immediately from Property 5.1. To complete the proof we will show, using an example, that this bound is tight.

Example 5.1. Let $m \geq 2$. The job set consists of two subsets J_1 and J_2 with cardinalities 1 and $n-1$, respectively. Weights are defined as $w_1 = \underline{w}, w_2 = \overline{w}$, where $\underline{w} \leq \overline{w}$ are any numbers. Two processing times are defined explicitly $p_{m1} = 1, p_{m2} = 0$. All the remaining (undefined explicitly) processing times are equal to ϵ, and ϵ is the sufficiently small number such that $\lim_{\epsilon \to 0} \epsilon \overline{w} n^2 = 0$. ◇

At first, analyse the behaviour of an algorithm $A \in \{CDS, G, P, RA\}$ in this example. For the k-th auxiliary problem in Algorithm CDS we have $a_1^k = a_2^k = k\epsilon$, $b_1^k = 1+(k-1)\epsilon$, $b_2^k = (k-1)\epsilon$, $k = 1, \ldots, m-1$. Therefore all $m-1$ auxiliary problems can provide the same result $\pi^k = (1)_1(2)_{n-1}$, $k = 1, \ldots, n$, and thus $\pi^{CDS} = (1)_1(2)_{n-1}$, see Fig. 5.3. In algorithm G, we have $\beta_1 = -1/\epsilon, \beta_2 = 1/\epsilon$ for $m = 2$, and $\beta_1 = -1/(2\epsilon), \beta_2 = 1/\epsilon$ for $m > 2$. It means that G can generate the permutation $\pi^G = (1)_1(2)_{n-1}$. Since $\omega_1 = -m+1+(m-1)\epsilon$ and $\omega_2 = (m-1)\epsilon$, algorithm P generates $\pi^P = (1)_1(2)_{n-1}$. In algorithm RA, the processing times of a single two-machine auxiliary flow-shop problem are defined as follows: $a_1 = 1 + \epsilon(m-1)(m+2)/2$, $b_1 = m + \epsilon(m-1)m/2$, $a_2 = \epsilon(m-1)(m+2)/2$, and $b_2 + \epsilon(m-1)m/2$. Therefore RA provides $\pi^{RA} = (1)_1(2)_{n-1}$. It is easy to prove that $C_{\text{sum}}(\pi^A) = \underline{w} + \overline{w}(n-1)$, $A \in \{CDS, G, P, RA\}$.

Now apply algorithm HC starting from $\pi = (1)_1(2)_{n-1}$. By virtue of (5.5) we have revised gaps $d_{12}^R = 1-\epsilon$ and $d_{22}^R = -0.1\epsilon$ since $L_{12} = \emptyset$ and $L_{22} = \{m-1\}$. Initially we set $a = 1$, $b = n$. At Step 2 we find $g = 2, h = 2$. Since $X = d_{12}^R = 1-\epsilon > 0$, $Y = d_{22} = -0.1\epsilon < 0$ and $|X| > |Y|$, the jump from Step 3 to Step 4 will be performed. This interchange does not cause any improvement, so job 1 remains in the position 1. Further interchanges operate on the set J_2 of uniform jobs, and therefore cannot introduce any improvement at all. Finally we have $\pi^{A+HC} = \pi^A$ and $C_{\text{sum}}(\pi^{A+HC}) = C_{\text{sum}}(\pi^A)$ for $A \in \{CDS, G, P, RA\}$. It can

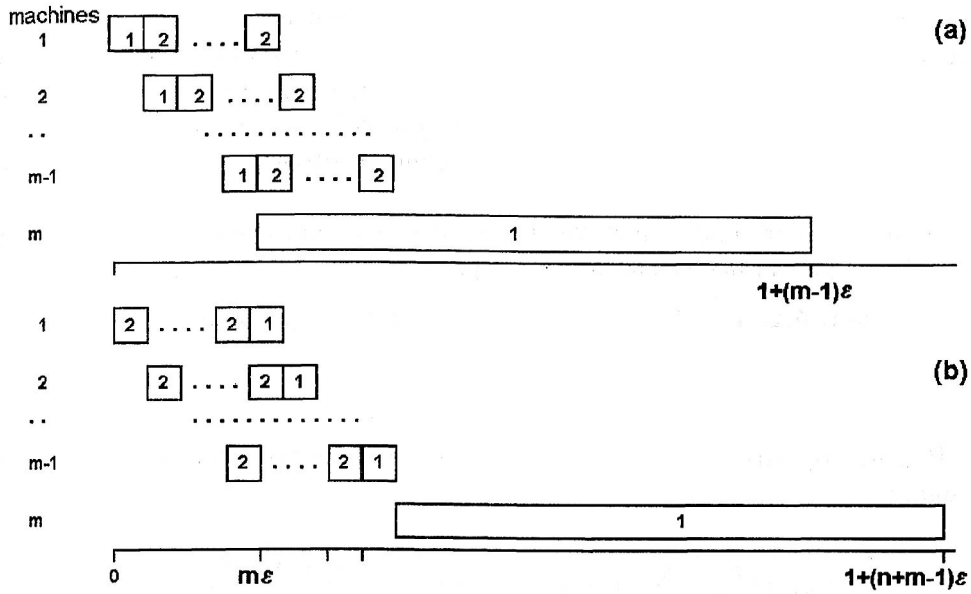

Figure 5.3: Schedules in Example 5.1: (a) for π^A, $A \in \{CDS, G, RA, CDS+HC, G+HC, P+HC, RA+HC\}$ and (b) for π^*

be proved that the permutation $\pi^* = (2)_{n-1}(1)_1$ is optimal and $C_{\text{sum}}(\pi^*) = \underline{w} + \underline{w}(m+n-3)\epsilon + \overline{w}(m-2)(n-1)\epsilon + \overline{w}n(n-1)\epsilon/2$. Finally, $C_{\text{sum}}(\pi^{A+HC})/C_{\text{sum}}(\pi^*)$ tends to $1+(n-1)(\overline{w}/\underline{w})$ if $\epsilon \to 0$, $A \in \{CDS, G, P, RA\}$. ■

It follows from Example 5.1 that some modifications of the basic scheme of algorithm HC are non-perspective from the worst-case point of view. As an example consider a modified form of Step 2: $d^R_{\pi(a)\pi(g)} = \max\{d^R_{\pi(a)\pi(j)} : a < j \leq b\}$, $d_{\pi(h)\pi(b)} = \min\{d^R_{\pi(j)\pi(b)} : a \leq j < b\}$. Another "unsuitable" modification replaces interchanges at Steps 4 and 5 by appropriate insertions. Also observe that the same result will be obtained if we use algorithm NEH (in its original form) as a method of finding the initial permutation.

The extremely high worst-case performance error is the direct consequence of some lacks observed in HC. First, information about p_{1i} and p_{mj} has been completely ignored in d^R_{ij}, see formulae (5.5). Second, the first and the last jobs in the permutation have never been moved somewhere, in spite of their evidently bad location. Third, neither CDS nor P, G, RS (or their variants) should be recommended for generating the initial permutation since they are specific for $C_{\max}$ problems.

In the context of previous considerations, the following question arises "does there exist an approximation algorithm for the stated problem with the worst-case performance ratio bounded by a function of m?". An algorithm obtained by natural extension of the well-known single-machine $WSPT$ rule provides the first positive answer. The next, more promising approach, is offered by an extension of CDS and by a new decomposition technique T discussed at the end of this section.

Let us consider algorithm F which generates a permutation π^F by sequencing the order of jobs in decreasing $w_j / \sum_{i=1}^{m} p_{ij}$.

Theorem 5.2. For the algorithm F we have the tight bound

$$\eta^F(C_{\text{sum}}) \leq m. \tag{5.10}$$

Proof. By virtue of (5.1) and (5.3) we can write the following sequence of inequalities

$$C_{\text{sum}}(\pi^F) = \sum_{j=1}^{n} w_{\pi^F(j)} \max_{1 \leq j_0 \leq j_1 \leq \ldots \leq j_m \leq j} \sum_{s=1}^{m} \sum_{t=j_{s-1}}^{j_s} p_{s\pi^F(t)}$$

$$\leq \sum_{j=1}^{n} w_{\pi^F(j)} \sum_{s=1}^{m} \sum_{t=1}^{j} p_{s\pi^F(t)}. \tag{5.11}$$

Since the permutation π^F is optimal for the single-machine problem with processing times $\sum_{s=1}^{m} p_{st}$, then

$$\sum_{j=1}^{n} w_{\pi^F(j)} \sum_{t=1}^{j} (\sum_{s=1}^{m} p_{s\pi^F(t)}) \leq \sum_{j=1}^{n} w_{\pi^*(j)} \sum_{t=1}^{j} (\sum_{s=1}^{m} p_{s\pi^*(t)}).$$

Therefore from (5.11), using the obvious machine-based bound

$$\sum_{j=1}^{n} w_{\pi^*(j)} \sum_{t=1}^{j} p_{s\pi^*(t)} \leq C_{\text{sum}}(\pi^*)$$

for $s = 1 \ldots, m$, we obtain

$$C_{\text{sum}}(\pi^F) \leq \sum_{j=1}^{n} w_{\pi^*(j)} \sum_{t=1}^{j} (\sum_{s=1}^{m} p_{s\pi^*(t)}) = \sum_{s=1}^{m} \sum_{j=1}^{n} w_{\pi^*(j)} \sum_{t=1}^{j} p_{s\pi^*(t)} \leq m C_{\text{sum}}(\pi^*).$$

To complete the proof we provide an example of the bound's tightness.

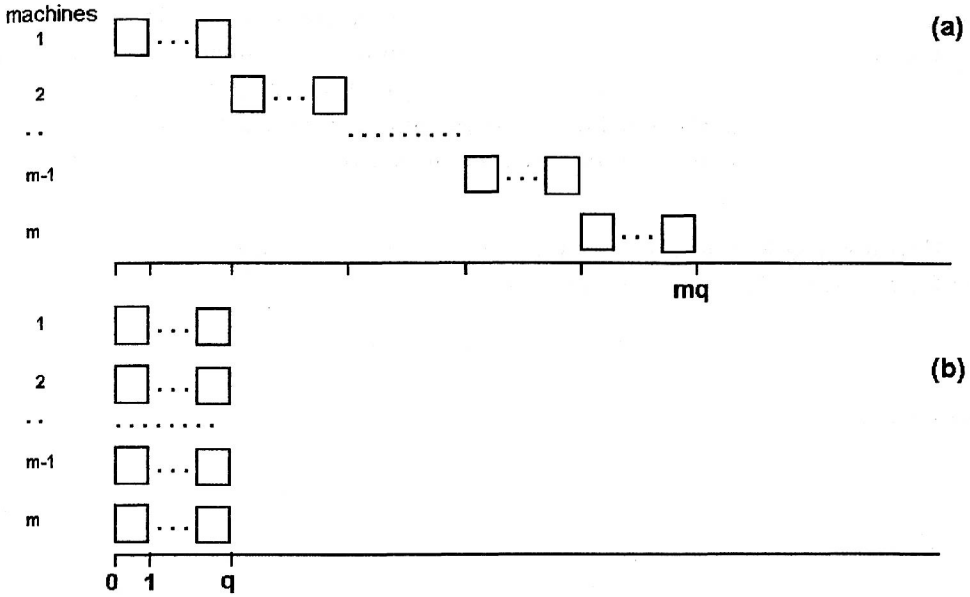

Figure 5.4: Schedules in Example 5.2: (a) for π^F, and (b) for π^*. All jobs on the machine i have index i and processing time $p_{ii} = 1$.

Example 5.2. Let $m \geq 2$. The job set J consists of m subsets $J_1, \ldots, J_m$, each with the cardinality q, where q is an integer. The processing times and weights are equal to $p_{ii} = 1, w_i = 1, i = 1, 2, \ldots, m$. All the remaining (undefined above) processing times are equal to zero. Thus, we have $n = mq$ jobs. $\diamond$

Since $w_j / \sum_{i=1}^{m} p_{ij} = 1$ for all $j = 1, 2, \ldots, m$, algorithm F can generate the permutation $\pi^F = (1)_q(2)_q \ldots (m)_q$ and $C_{\text{sum}}(\pi^F) = (1/2)[q^2 m(m-1) + qm(q+1)]$, see Fig. 5.4. One can prove that the optimal job processing order is $\pi^* = (m, m-1, \ldots 1)_q$ and $C_{\text{sum}}(\pi^*) = (1/2)mq(q+1)$. Hence $C_{\text{sum}}(\pi^F)/C_{\text{sum}}(\pi^*) = (m-1)\frac{q}{q+1} + 1$ and tends to m if $q \to \infty$. ■

Consider a new algorithm, called hereafter Q/X, designed for the criterion C_{sum} using scheme of CDS. We will show that Q/X can be better than F from the worst-case point of view. Let us assume that there is known an approximation algorithm X for the two-machine flow-shop problem with the criterion C_{sum} and the worst-case performance ratio $\eta^X(C_{\text{sum}})$. The proposed algorithm Q generates a permutation $\pi^Q \in \{\pi^1, \ldots, \pi^{m-1}\}$ such that $C_{\text{sum}}(\pi^Q) = \min_{\pi \in \{\pi^1, \ldots, \pi^{m-1}\}} C_{\text{sum}}(\pi^k)$ where π^k is the permutation generated by the algorithm X for an auxiliary two-machine flow-shop problem with the criterion C_{sum} and processing times $a_j^k = \sum_{i=1}^{k} p_{ij}$, $b_j^k = \sum_{i=m-k+1}^{m} p_{ij}$, $j = 1, \ldots, n$, on ma-

chines 1 and 2, respectively, $k = 1, \ldots, m-1$. Denote by Q_k the algorithm which generates the single permutation π^k and by Q_k/X the combination of Q_k with X. For the k-th auxiliary two-machine flow-shop problem, denote by Z^k the k-th instance obtained from the instance Z, by $C_{ij}(Z)$ and $C_{ij}(Z^k)$ the completion time of the job j on the machine i for instances Z and Z^k, respectively. Let $r = \lceil m/2 \rceil$. To carry out the analysis, we need two auxiliary results, which are found in Property 5.2.

Property 5.2. For any π, any Z, any $j = 1, \ldots, n$, and
(a) for $k = r, r+1, \ldots, m-1$, we have

$$C_{m\pi(j)}(Z) \leq C_{2\pi(j)}(Z^k), \tag{5.12}$$

(b) for $k = 1, 2, \ldots, m-1$, we have

$$C_{m\pi(j)}(Z) \geq \frac{C_{2\pi(j)}(Z^k)}{k}. \tag{5.13}$$

□

Proof. (a) Let $1 \leq u_0 \leq u_1 \leq \ldots \leq u_m \leq j$ be the sequence of integers minimising the right-hand side of formula (5.3) for $i = m$. Then, we obtain

$$C_{m\pi(j)}(Z) = \sum_{s=1}^{m} \sum_{t=u_{s-1}}^{u_s} p_{s\pi(t)} = \sum_{s=1}^{k} \sum_{t=u_{s-1}}^{u_s} p_{s\pi(t)} + \sum_{s=k+1}^{m} \sum_{t=u_{s-1}}^{u_s} p_{s\pi(t)}$$

$$\leq \sum_{t=u_0}^{u_k} \sum_{s=1}^{k} p_{s\pi(t)} + \sum_{t=u_k}^{u_m} \sum_{s=k+1}^{m} p_{s\pi(t)}$$

$$\leq \max_{1 \leq v_0 \leq v_1 \leq v_2 \leq j} \{ \sum_{t=v_0}^{v_1} (\sum_{s=1}^{k} p_{s\pi(t)}) + \sum_{t=v_1}^{v_2} (\sum_{s=m-k+1}^{m} p_{s\pi(t)}) \} = C_{2\pi(j)}(Z^k).$$

(b) Employing the definition (5.3) for $i = m$, we can write the following sequence of dependencies

$$kC_{m\pi(j)}(Z) = \sum_{i=1}^{k} \max_{1 \leq j_0 \leq j_1 \leq \ldots \leq j_m \leq j} \sum_{s=1}^{m} \sum_{t=j_{s-1}}^{j_s} p_{s\pi(t)}$$

$$\geq \sum_{i=1}^{k} \max_{1 \leq j_0 \leq j_1 \leq \ldots \leq j_m \leq j} (\sum_{t=j_{i-1}}^{j_i} p_{i\pi(t)} + \sum_{t=j_{m-k+i-1}}^{j_{m-k+i}} p_{m-k+i,\pi(t)})$$

$$= \sum_{i=1}^{k} \max_{1\le j_{i-1}\le j_i\le j_{m-k+i-1}\le j_{m-k+i}\le j} \Big(\sum_{t=j_{i-1}}^{j_i} p_{i\pi(t)} + \sum_{t=j_{m-k+i-1}}^{j_{m-k+i}} p_{m-k+i,\pi(t)} \Big)$$

$$\ge \sum_{i=1}^{k} \max_{1\le j_{i-1}\le j_i=j_{m-k+i-1}\le j_{m-k+i}\le j} \Big(\sum_{t=j_{i-1}}^{j_i} p_{i\pi(t)} + \sum_{t=j_{m-k+i-1}}^{j_{m-k+i}} p_{m-k+i,\pi(t)} \Big)$$

$$= \sum_{i=1}^{k} \max_{1\le v_0\le v_1\le v_2\le j} \Big(\sum_{t=v_0}^{v_1} p_{i\pi(t)} + \sum_{t=v_1}^{v_2} p_{m-k+i,\pi(t)} \Big)$$

$$\ge \max_{1\le v_0\le v_1\le v_2\le j} \Big\{ \sum_{t=v_0}^{v_1} \Big(\sum_{i=1}^{k} p_{i\pi(t)} \Big) + \sum_{t=v_1}^{v_2} \Big(\sum_{i=1}^{k} p_{m-k+i,\pi(t)} \Big) \Big\} = C_{2\pi(j)}(Z^k).$$

The first inequality is the direct consequence of a relaxation of job processing times. Clearly, since $i \le m-k+i+1$ for $m \ge 2$, any i and any $k \in \{1,\dots,m-1\}$, therefore $j_i \le j_{m-k+i-1}$. The second inequality follows from the commonly known fact $\max_{i\in A} a_i \ge \max_{i\in A'} a_i$ for any $A' \subset A$. Subsequent equality is the result of a simple substitution $v_0 = j_{i-1}, v_1 = j_i, v_2 = v_{m-k+i}$. The final inequality employs the well-known claim $\sum_{i\in A} \max_{j\in B} a_{ij} \ge \max_{j\in B} \sum_{i\in A} a_{ij}$. Note also that $\sum_{i=1}^{k} p_{m-k+i,j} = \sum_{i=m-k+1}^{m} p_{ij}$. ■

The inequality (5.12) provides an upper bound of $C_{m\pi^*(j)}(Z)$ whereas (5.13) a lower bound. Now we are ready to formulate subsequent evaluations of the worst-case performance ratio.

Property 5.3. For algorithms Q_k/X, we have
(a) tight bounds for $k = 1, 2, \dots, r-1$,

$$\eta^{Q_k/X}(C_{\text{sum}}) \le 1 + (n-1)(\overline{w}/\underline{w}), \tag{5.14}$$

(b) for $k = r, r+1, \dots, m$,

$$f_m(k)\eta^X(C_{\text{sum}}) \le \eta^{Q_k/X}(C_{\text{sum}}) \le k\eta^X(C_{\text{sum}}), \tag{5.15}$$

where

$$f_m(k) = \frac{k^2 + (m-k)^2}{m}. \tag{5.16}$$

□

Proof. (a) Due to Property 5.1, we need only an appropriate example.

Example 5.3. Let $m \ge 2$ and $r = \lceil m/2 \rceil$. The job set consists of two subsets J_1 and J_2 with cardinalities 1 and $n-1$, respectively. Weights are defined as

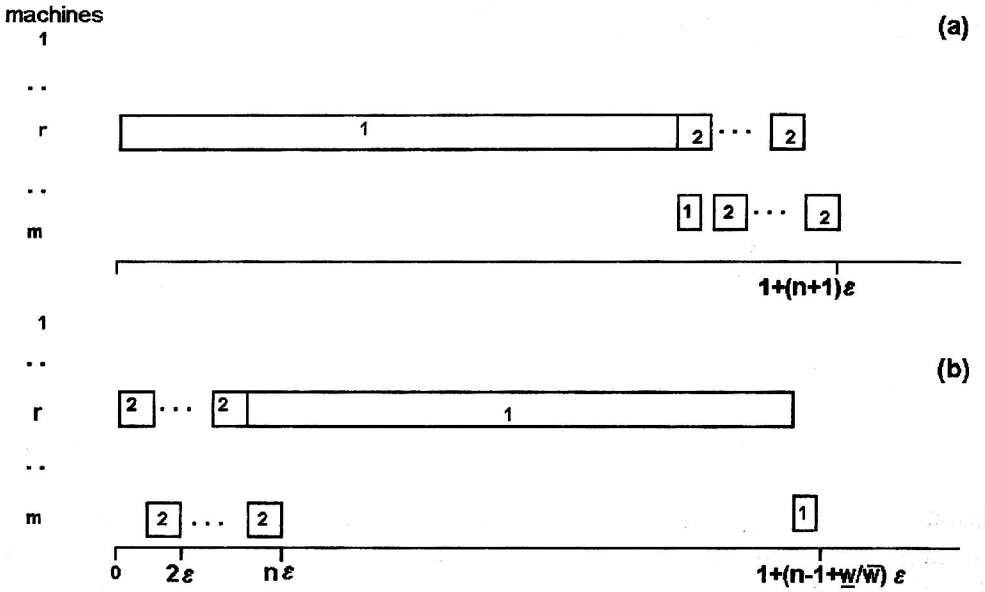

Figure 5.5: Schedules in Example 5.3: (a) for π^k, $k = 1, \ldots, r-1$, (b) for π^*

$w_1 = \underline{w}, w_2 = \overline{w}$ where $\underline{w} \leq \overline{w}$ are any numbers. The processing times are equal to $p_{r1} = 1, p_{r2} = \epsilon, p_{m1} = \epsilon\underline{w}/\overline{w}, p_{m2} = \epsilon$ where ϵ is the sufficiently small number such that $\lim_{\epsilon \to 0} \epsilon \overline{w} n^2 = 0$. All the remaining (undefined above) processing times are equal zero. $\diamond$

Suppose that algorithm Q has been combined with algorithm X producing optimal permutation. For the primal $r-1$ auxiliary problems we obtain $a_j^k = 0$, $j = 1, 2, b_1^k = \epsilon\underline{w}/\overline{w}, b_2^k = \epsilon, k = 1, \ldots, r-1$. Each of these problems can be solved using $WSPT$ rule on the second machine. Since $w_1/b_1^k = w_2/b_2^k$, algorithm X can provide $r-1$ identical permutations $\pi^k = (1)_1(2)_{n-1}$, $k = 1, \ldots, r-1$, and $C_{\text{sum}}(\pi^k) = \underline{w} + \underline{w}^2\epsilon/\overline{w} + \overline{w}n\epsilon(n-1)/2 + \overline{w}(n-1)(1+\epsilon)$, see Fig. 5.5. One can prove that the permutation $\pi^* = (2)_{n-1}(1)_1$ is optimal and $C_{\text{sum}}(\pi^*) = \overline{w}\epsilon + \overline{w}\epsilon n(n-1)/2 + \underline{w}\epsilon(n-1) + \underline{w} + \underline{w}^2\epsilon/\overline{w}$. Hence, $C_{\text{sum}}(\pi^k)/C_{\text{sum}}(\pi^*)$ tends to $1 + \overline{w}/\underline{w}(n-1)$ if $\epsilon \to 0, k = 1, \ldots, r-1$.

(b) Applying the formulae (5.12) for $\pi = \pi^k$ in the definition (5.1), we obtain

$$C_{\text{sum}}(\pi^k; Z) \leq C_{\text{sum}}(\pi^k; Z^k).$$

By virtue of the definition of $\eta^X(C_{\text{sum}})$ we have $C_{\text{sum}}(\pi^k; Z^k)/C_{\text{sum}}(\pi^{*k}; Z^k) \leq \eta^X(C_{\text{sum}})$ where π^{*k} is the optimal permutation of the k-th two-machine flow-shop problem. Next applying the formulae (5.13) for $\pi = \pi^*$ in the definition

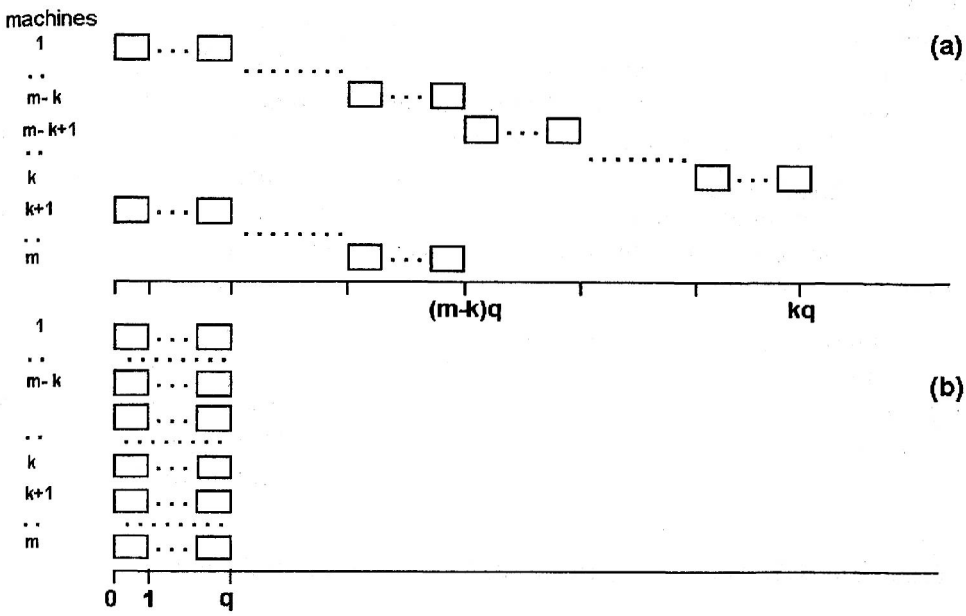

Figure 5.6: Schedules in Example 5.4: (a) for π^k, $k = r, \ldots, m$, (b) for π^*. All jobs on the machine i have the index i and the processing time $p_{ii} = 1$

(5.1) we obtain

$$C_{\text{sum}}(\pi^*; Z) = \sum_{j=1}^{m} w_{\pi^*(j)} C_{m\pi^*(j)}(Z) \geq \frac{1}{k} \sum_{j=1}^{n} w_{\pi^*(j)} C_{2\pi^*(j)}(Z^k)$$

$$= \frac{C_{\text{sum}}(\pi^*; Z^k)}{k} \geq \frac{C_{\text{sum}}(\pi^{*k}; Z^k)}{k} \geq \frac{C_{\text{sum}}(\pi^k; Z^k)}{k\eta^X(C_{\text{sum}})}.$$

Combining two last results we get

$$C_{\text{sum}}(\pi^k; Z) \leq k\eta^X(C_{\text{sum}}) C_{\text{sum}}(\pi^*; Z)$$

which yields the upper bound in (5.15). The lower bound in (5.15) follows from the example.

Example 5.4. Let $m \geq 2$. The job set J consists of m subsets $J_1, \ldots, J_m$, each with the cardinality q, and q is an integer. The processing times and weights are equal to $p_{jj} = 1, w_j = 1, j = 1, 2, \ldots, m$. All the remaining (undefined above) processing times are equal to zero. Hence, the total number of jobs is $n = mq$. $\diamond$

Processing times of the k-th auxiliary flow-shop for $k = r, r+1, \ldots, m-1$, are defined as follows: $a_j^k = 1, j = 1, 2, \ldots, k, a_j^k = 0, j = k+1, \ldots, m, b_j^k = 0, j = 1, 2, \ldots, m-k, b_j^k = 1, j = m-k+1, \ldots, m$. Assume that $\eta^X(C_{\text{sum}}) = 1$, i.e. every auxiliary problem is solved optimally. Hence X can generate permutations $\pi^k = (k+1, 1)_q(k+2, 2)_q \ldots (m, m-k)_q(m-k+1)_q(m-k+2)_q \ldots (k)_q$, and $C_{\text{sum}}(\pi^k) = \sum_{i=1}^{kq} i + \sum_{i=1}^{(m-k)q} i = kq(kq+1)/2 + (m-k)^2q^2/2 + (m-k)q/2$, see Fig. 5.6. The optimal permutation is $\pi^* = (m, m-1, \ldots, 1)_q$, and $C_{\text{sum}}(\pi^*) = \sum_{i=1}^{q} mi = mq(q+1)/2$. Finally, $C_{\text{sum}}(\pi^k)/C_{\text{sum}}(\pi^*)$ tends to $[k^2+(m-k)^2]/m = f_m(k)$ if $q \to \infty$. ■

The function $f_m(k)$ is nondecreasing, $f_m(k) \geq m/2$ for any k, and $m-2 \leq f_m(m-1) \leq m-1$. A more precise analysis shows that $f_m(r) = r$ for even m, and $f_m(r) = r - \frac{1}{2}(1 - \frac{1}{2m})$ for odd m. Presumably, a more extensive research may provide better lower and/or upper bounds of $\eta^{Q_k/X}(C_{\text{sum}})$, however this considerable effort will not change the final worst-case evaluation of the algorithm Q/X.

Theorem 5.3. For the algorithm Q/X we have the tight bound

$$\eta^{Q/X}(C_{\text{sum}}) \leq \lceil m/2 \rceil \eta^X(C_{\text{sum}}). \tag{5.17}$$

Proof. By virtue of Property 5.3 we have

$$\eta^{Q/X}(C_{\text{sum}}) = min_{1 \leq k \leq m} \eta^{Q_k/X}(C_{\text{sum}}) \leq \min_{r \leq k \leq m-1} k\eta^X(C_{\text{sum}})$$

$$= \lceil m/2 \rceil \eta^X(C_{\text{sum}})$$

that implies (5.17). To complete the proof we consider the following example.

Example 5.5. Let $m \geq 2$. The job set J consists of r subsets, each with the cardinality q and q is an integer. The processing times and weights are equal to $p_{m-r+i,i} = 1, w_i = 1, i = 1, 2, \ldots, r$. All the remaining (undefined above) processing times are equal to zero. The total number of jobs is $n = rq$. ◇

Calculation of the processing times for auxiliary flow-shops provides the following results: (a) for $k = 1, \ldots, m-r$ we have $a_j^k = 0, j = 1, \ldots, r, b_j^k = 0, j = 1, \ldots, k$ and $b_j^k = 1, j = k+1, \ldots, r$, (b) for $k = m-r+1, \ldots, m-1$, we have $a_j^k = 1, j = 1, \ldots, k-m+r, a_j^k = 0, j = k-m+r+1, \ldots, r$ and $b_j^k = 1, j = 1, \ldots, r$. Assume that $\eta^X(C_{\text{sum}}) = 1$, i.e. every auxiliary problem is solved optimally. In consequence, X can generate $\pi^1 = \ldots = \pi^{m-r} = (1)_q(2)_q \ldots (r)_q$ and $\pi^{m-r+1} = \ldots = \pi^{m-1} = (r)_1(1)_q \ldots (r-1)_q(r)_{q-1}$. Hence $C_{\text{sum}}(\pi^1) = \sum_{i=1}^{rq} i = rq(rq+1)/2$ and $C_{\text{sum}}(\pi^{m-r+1}) = 1 + \sum_{i=1}^{rq-1} i = 1 + (rq-1)rq/2 \leq C_{\text{sum}}(\pi^1)$, see Fig. 5.7.

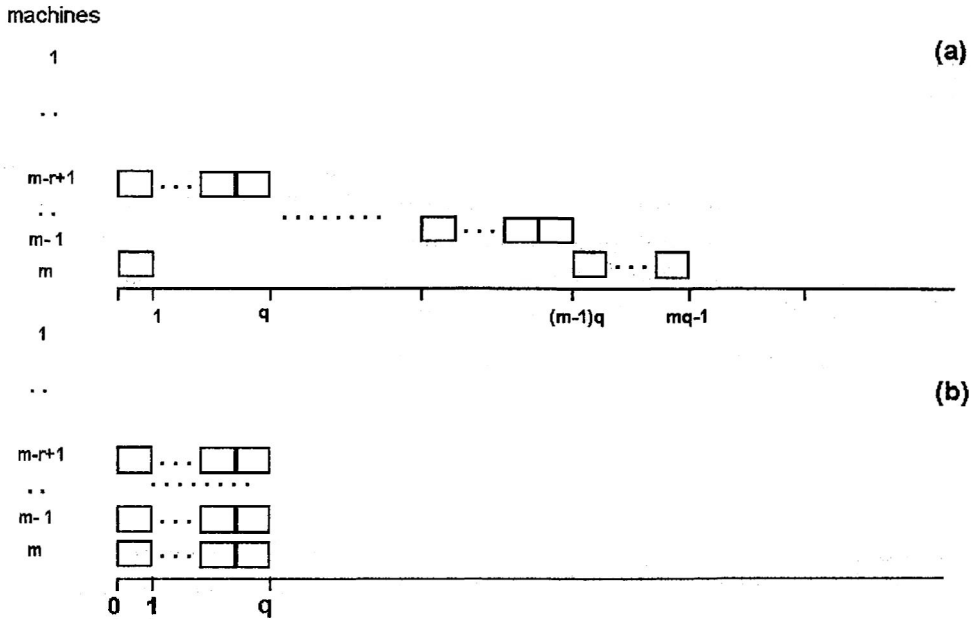

Figure 5.7: Schedules in Example 5.5: (a) for π^{m-r+1}, (b) for π^*. All jobs on the machine $i > m - r$ have the index $i - m + r$ and the processing time $p_{m-r+i,i} = 1$

It is easy to prove that the optimal permutation is $\pi^* = (r, r-1, \ldots, 1)_q$, and $C_{\text{sum}}(\pi^*) = \sum_{i=1}^{q} ri = rq(q+1)/2$. Finally, $C_{\text{sum}}(\pi^{m-r+1})/C_{\text{sum}}(\pi^*)$ tends to r if $q \to \infty$. ■

It is clear that algorithm Q/X will provide better results if and only if there exists an approximation algorithm X for the two-machine problem with $\eta^X(C_{\text{sum}})$ less than 2. Up to now, such algorithms have been found for a few special cases, see Table 5.6. Hence, an algorithm X for the general case with $\eta^X(C_{\text{sum}}) < 2$ is still looked for.

At the end of this section we propose another, general family of algorithms, called hereafter T/Yk, which can be better than Q/X from the worst-case point of view. An algorithm from this family is based on an approximation of m-machine problem by some k-machine flow-shop problem for $k \leq m$. Let $m_1, \ldots, m_k$ be a sequence of integers such that $m_s \geq 1$, $s = 1, \ldots, k$, $\sum_{s=1}^{k} m_s = m$, for some $k \leq m$. We define $l_i = \sum_{s=1}^{i} m_s$, $i = 1, \ldots, k$, and $l_0 = 0$. The auxiliary k-machine flow-shop problem has the processing times defined as follows

$$t_{ij} = \sum_{s=l_{i-1}+1}^{l_i} p_{sj}. \tag{5.18}$$

Although the latter problem is defined in a univocal way by the sequence of machine aggregations $(m_1, \ldots, m_k)$, for the sake of notation simplicity we will identify it by means of k. Since this problem for $k > 1$ is NP-hard, we assume that it is solved by an approximation algorithm Yk with the worst-case performance ratio $\eta^{Yk}(C_{\text{sum}})$. Clearly, the proposed algorithm T/Yk generates a permutation $\pi^{T/Yk}$ by solving, with the worst-case bound $\eta^{Yk}(C_{\text{sum}})$, the k-machine flow-shop problem with processing times defined by (5.18). Let us denote this k-machine instance by $\overline{Z}^k$, and the appropriate job completion times by $C_{ij}(Z)$ and $C_{ij}(\overline{Z}^k)$. Two preliminary results found in Properties 5.4 and 5.5 are necessary to prove the final evaluation of the performance T/Yk.

Property 5.4. For any π, any Z, any $j \in N$, and any $(m_1, \ldots, m_k)$, we have

$$C_{m\pi(j)}(Z) \leq C_{k\pi(j)}(\overline{Z}^k). \tag{5.19}$$

□

Proof. Let $1 \leq u_0 \leq u_1 \leq \ldots \leq u_m \leq j$ be the sequence of integers minimising the right-hand side of formula (5.3) for $i = m$. Then, we obtain

$$C_{m\pi(j)}(Z) = \sum_{s=1}^{m} \sum_{t=u_{s-1}}^{u_s} p_{s\pi(t)} = \sum_{r=1}^{k} \sum_{s=l_{r-1}+1}^{l_r} \sum_{t=u_{s-1}}^{u_s} p_{s\pi(t)}$$

$$\leq \sum_{r=1}^{k} \sum_{s=l_{r-1}+1}^{l_r} \sum_{t=u_{l_r-1}}^{u_{l_r}} p_{s\pi(t)}$$

$$= \sum_{r=1}^{k} \sum_{t=u_{l_{r-1}}}^{u_{l_r}} \sum_{s=l_{r-1}+1}^{l_r} p_{s\pi(t)} \leq \max_{1 \leq v_0 \leq \ldots \leq v_r \leq j} \sum_{r=1}^{k} \sum_{t=v_{r-1}}^{v_r} \sum_{s=l_{r-1}+1}^{l_r} p_{s\pi(t)}$$

$$= \max_{1 \leq v_0 \leq \ldots \leq v_r \leq j} \sum_{r=1}^{k} \sum_{t=v_{r-1}}^{v_r} t_{r\pi(t)} = C_{r\pi(j)}(\overline{Z}^k),$$

which completes the proof. ■

Property 5.5. For any π, any Z, any $j \in N$, and any $(m_1, \ldots, m_k)$, we have

$$C_{r\pi(j)}(\overline{Z}^k) \leq m_{\max} C_{m\pi(j)}(Z) \tag{5.20}$$

where

$$m_{\max} = \max_{1 \leq i \leq k} m_i. \tag{5.21}$$

□

Proof. Employing (5.3) for $i = m$ we obtain the following sequence of inequalities

$$m_{\max} C_{m\pi(j)}(Z) = \sum_{i=1}^{m_{\max}} \max_{1\le j_0\le j_1\le\ldots\le j_m\le j} \sum_{s=1}^{m} \sum_{t=j_{s-1}}^{j_s} p_{s\pi(t)}$$

$$= \sum_{i=1}^{m_{\max}} \max_{1\le j_0\le j_1\le\ldots\le j_m\le j} \sum_{r=1}^{k} \sum_{s=l_{r-1}+1}^{l_r} \sum_{t=j_{s-1}}^{j_s} p_{s\pi(t)}$$

$$\ge \sum_{i=1}^{m_{\max}} \max_{1\le j_0\le j_1\le\ldots\le j_m\le j} \sum_{r=1}^{k} \sum_{t=j_{x_{ri}-1}}^{j_{x_{ri}}} p_{x_{ri}\pi(t)}$$

where $x_{ri} = \min\{l_{r-1} + i, l_r\}$. Note that for any fixed i we have $x_{r-1,i} \le x_{ri}$. Next note that for the fixed i, maximisation can be done over indices $v_r = x_{ri}$, then we can continue as follows

$$= \sum_{i=1}^{m_{\max}} \max_{1\le v_0\le v_1\le\ldots\le v_k\le j} \sum_{r=1}^{k} \sum_{t=v_{r-1}}^{v_r} p_{x_{ri}\pi(t)}$$

$$\ge \max_{1\le v_0\le v_1\le\ldots\le v_k\le j} \sum_{r=1}^{k} \sum_{t=v_{r-1}}^{v_r} \sum_{i=1}^{m_{max}} p_{x_{ri}\pi(t)}$$

$$\ge \max_{1\le v_0\le v_1\le\ldots\le v_k\le j} \sum_{r=1}^{k} \sum_{t=v_{r-1}}^{v_r} \sum_{i=1}^{m_r} p_{x_{ri}\pi(t)}$$

$$= \max_{1\le v_0\le v_1\le\ldots\le v_k\le j} \sum_{r=1}^{k} \sum_{t=v_{r-1}}^{v_r} \sum_{i=l_{r-1}+1}^{l_r} p_{i\pi(t)}$$

$$= \max_{1\le v_0\le v_1\le\ldots\le v_k\le j} \sum_{r=1}^{k} \sum_{t=v_{r-1}}^{v_r} t_{r\pi(t)} = C_{k\pi(j)}(Z^k).$$

The final inequality has been obtained using the well-known *sum-max* $\ge$ *max-sum* fact $\sum_{i\in A} \max_{j\in B} a_{ij} \ge \max_{j\in B} \sum_{i\in A} a_{ij}$. ■

Using the introduced properties we are able to evaluate the worst-case performance ratio of algorithms T/Yk.

Theorem 5.4. For any algorithm T/Yk defined by the sequence $m_1, \ldots, m_k$ we have

$$\eta^{T/Yk}(C_{\text{sum}}) = m_{max}\, \eta^{Yk}(C_{\text{sum}}). \tag{5.22}$$

Proof. Applying (5.19) for $\pi = \pi^k$ in the definition (5.1), we obtain

$$C_{\text{sum}}(\pi^k; Z) \leq C_{\text{sum}}(\pi^k; \overline{Z}^k).$$

By virtue of the definition of $\eta^{Yk}(C_{\text{sum}})$ we have $C_{\text{sum}}(\pi^k; \overline{Z}^k)/C_{\text{sum}}(\pi^{*k}; \overline{Z}^k) \leq \eta^{Yk}(C_{\text{sum}})$ where π^{*k} is the optimal permutation of the k-machine flow-shop problem. Now, applying the formulae (5.20) to $\pi = \pi^*$ in the definition (5.1), we obtain

$$m_{max}\, C_{\text{sum}}(\pi^*; Z) = m_{max} \sum_{j=1}^{m} w_{\pi^*(j)} C_{m\pi^*(j)}(Z)$$

$$\geq \sum_{j=1}^{n} w_{\pi^*(j)} C_{k\pi^*(j)}(\overline{Z}^k) = C_{\text{sum}}(\pi^*; \overline{Z}^k) \geq C_{\text{sum}}(\pi^{*k}; \overline{Z}^k) \geq \frac{C_{\text{sum}}(\pi^k; \overline{Z}^k)}{\eta^{Yk}(C_{\text{sum}})}.$$

By combining these results we get the following upper bound on $\eta^{T/Yk}(C_{\text{sum}})$

$$C_{\text{sum}}(\pi^k; Z) \leq m_{max}\, \eta^{Yk}(C_{\text{sum}})\, C_{\text{sum}}(\pi^*; Z).$$

Let consider also a lower bound.

Example 5.6. Let $m \geq 2$. Without losing generality we can assume that $m_{\max} = m_1$. To simplify notation we set $c = m_{\max}$. The job set J consists of c subsets $J_1, \ldots, J_c$ each with cardinality w each and w is an integer. The processing times and weights are equal to $p_{ii} = 1$, $w_i = 1$, $i = 1, 2, \ldots, c$. All the remaining (undefined above) processing times are equal to zero. Thus, we have $n = cw$ jobs. $\diamond$

For this instance we have $t_{1j} = \sum_{i=1}^{b} p_{ij} = 1$ and $t_{ij} = 0$, $i = 2, \ldots, k$, for all $j = 1, 2, \ldots, c$. Weights are equal to $w_j = 1$, $j = 1, \ldots, c$. Due to special structure, the auxiliary problem can be solved in such a way as to obtain the optimal one using the well-known $WSPT$ rule. Therefore $\eta^{Yk} = 1$ and the algorithm T/Yk can generate the permutation $\pi^{T/Yk} = (1)_w(2)_w \ldots (c)_w$. Consequently, $C_{\text{sum}}(\pi^{T/Yk}) = (1/2)[w^2c(c-1) + wc(w+1)]$. One can prove that the the optimal job processing order is $\pi^* = (c, c-1, \ldots 1)_w$ and $C(\pi^*) = (1/2)cw(w+1)$. Finally $C_{\text{sum}}(\pi^{T/Yk})/C_{\text{sum}}(\pi^*) = (c-1)\frac{w}{w+1} + 1$ which tends to $c = m_{max}$ if $w \to \infty$. ∎

Theorem 5.4 provides an unexpected theoretical result.

Collorary. There exists the algorithm T/Yk such that

$$\eta^{T/Yk}(C_{\text{sum}}) \leq \lceil m/k \rceil \eta^{Yk}(C_{\text{sum}}). \tag{5.23}$$

For $k = 1$ we obtain the well-known result $\eta^{T/Y1}(C_{\text{sum}}) = m$, since $T/Y1$ is equivalent to F and $\eta^{Y1}(C_{\text{sum}}) = 1$. For $k = 2$ we get the evaluation $\eta^{T/Y2}(C_{\text{sum}}) = \lceil m/2 \rceil \eta^{Y2}(C_{\text{sum}})$ that had been previously found for other algorithms, e.g. Q/X. In practice, solving the two- or three-machine case is usually easier than the general m-machine problem. Therefore $k = 2, 3$ can be recommended for applications. For example, assuming $m = 9$, $k = 3$, $m_1 = m_2 = m_3 = 3$, and $\eta^{Y3}(C_{\text{sum}}) = 1$ (i.e. the auxiliary three-machine problem is solved in order to obtain its optimal solution), we obtain $\eta^{T/Y3}(C_{\text{sum}}) = 3$, whereas the best known up to now result refers to $\eta^{T/Y1}(C_{\text{sum}}) = 9$ or $\eta^{T/Y2}(C_{\text{sum}}) = \lceil m/2 \rceil = 5$. It is clear that algorithm T/Yk will provide better results than F and Q/X if and only if there exists an approximation algorithm for the k-machine problem for $k > 2$ with the worst-case performance ratio sufficiently good.

5.5.2 The mean completion time criterion

Recently, a considerable attention has been concentrated on the flow-shop problem with the criterion $\overline{C}_{\text{sum}}$, chiefly due to its industrial applications. In this section, we discuss some basic results of the worst-case analysis for a few approximation algorithms recommended for this problem.

Theorem 5.5. For algorithms $A \in \{RC1, RC2\}$ we have tight bounds

$$\eta^{A}(\overline{C}_{\text{sum}}) \leq n. \tag{5.24}$$

Proof. The upper bound follows from Property 5.1. So, using the following example we will show that these bounds are tight.

Example 5.7. Let $m \geq 2$. The job set consists of two subsets J_1 and J_2 with cardinalities 1 and $n - 1$, respectively. The processing times are equal to $p_{m1} = 1, p_{m-1,2} = p_{m2} = \epsilon$ where ϵ denotes the sufficiently small number such that $\lim_{\epsilon \to 0} \epsilon n^2 = 0$. All the remaining (undefined above) processing times are equal to zero. $\diamond$

Let us start the algorithms $RC1$ and $RC2$ with $d = 0$, i.e. $S = \emptyset$. We have $\omega_1' = \omega_1'' = 0$ and $\omega_2' = \omega_2'' = \epsilon$. So both algorithms schedule at the first job 1, i.e. $\sigma = (1)_1$. The remain $n - 1$ jobs are uniform and therefore $RC1$ and $RC2$ provide the same permutation $\pi^{RC1} = \pi^{RC2} = (1)_1(2)_{n-1}$, see Fig. 5.8.

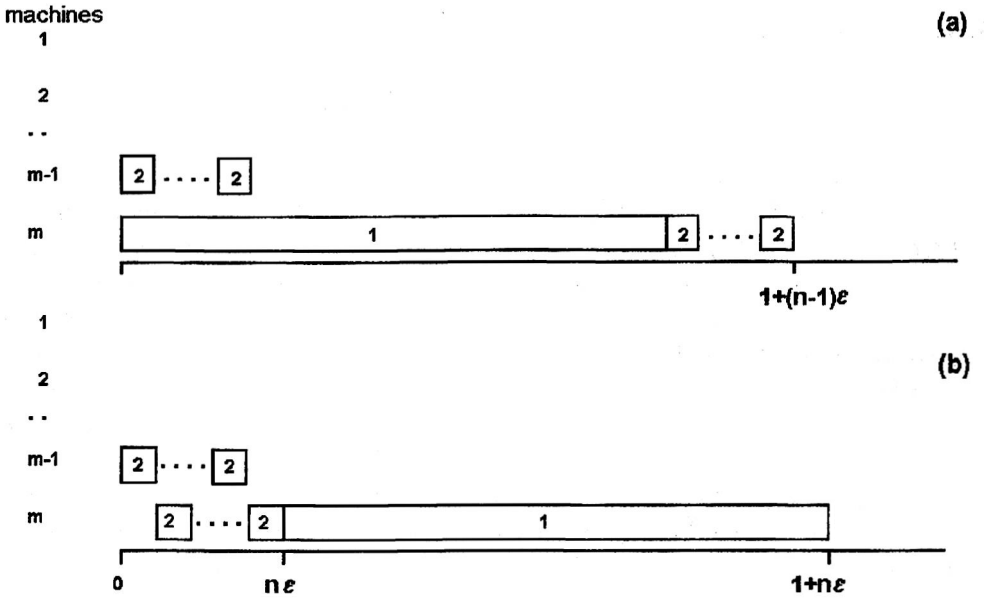

Figure 5.8: Schedules in Example 5.7: (a) for π^A, $A \in \{RC1, RC2\}$, (b) for π^*.

Hence, $\overline{C}_{\text{sum}}(\pi^{RC1}) = \overline{C}_{\text{sum}}(\pi^{RC2}) = 1 + \frac{1}{2}\epsilon(n-1)$. It can be proved that the permutation $\pi^* = (2)_{n-1}(1)_1$ is optimal and $\overline{C}_{\text{sum}}(\pi^*) = \frac{\epsilon(n-1)(n+2)}{2n} + \epsilon + \frac{1}{n}$. Finally, $\overline{C}_{\text{sum}}(\pi^A)/\overline{C}_{\text{sum}}(\pi^*)$ tends to n if $\epsilon \to 0, A \in \{RC1, RC2\}$. ■

This extremely high performance error of $RC1$ and $RC2$ is a direct consequence of the single mistake made at time moment zero. Since the waiting time for the big job is shorter (by ϵ) than the waiting time for each of many small jobs, so this big job will be scheduled first blocking completely the machine. This case owns some similarities to the worst-case instance of Schrage's algorithm for the single-machine problem with ready and delivery times and the makespan criterion. However, the worst-case performance of Schrage's method is deteriorated by such a mistake only twice, but of $RC1$, $RC2$ – up to n-times.

One can say that the detected mistake of $RC1$ ($RC2$) can be avoided by considering each job in turn as the first one in the primal sequence, see algorithm $KS2$. However, the following example shows that such a thinking is illusory.

Example 5.8. Let $m \geq 2$. The job set consists of two subsets J_1 and J_2 with the cardinalities 1 and $n-1$, respectively. The processing times are equal to $p_{m1} = 1, p_{m-1,1} = p_{m-1,2} = p_{m2} = \epsilon$ where ϵ is the sufficiently small number such that $\lim_{\epsilon \to 0} \epsilon n^2 = 0$. All the remaining (undefined above) processing times

are equal to zero. ◇

Starting from $\sigma = (1)$ we have to generate the job processing order $\overline{\pi}^{KS2} = (1)_1(2)_{n-1}$ since there are no other alternatives. In turn, starting from $\sigma = (2)$ we can generate the job processing order $\pi^{KS2} = (2)_1(1)_1(2)_{n-2}$. Indeed, if $\sigma = (2)$ we can schedule next the job from J_1 or a job from J_2. In both cases delays between σ and next scheduled job are zero, so the partial processing order for the next iteration can be $\sigma = (2)_1(1)_1$. Next, there are no other choices. Then, $n\overline{C}_{\text{sum}}(\overline{\pi}^{KS2}) = n + \frac{1}{2}\epsilon n(n+1)$ and $n\overline{C}_{\text{sum}}(\pi^{KS2}) = 2\epsilon + \frac{1}{2}\epsilon(n-2)(n-1) + (n-1)(1+2\epsilon)$. It can be proved that the permutation $\pi^* = (2)_{n-1}(1)_1$ is optimal and $n\overline{C}_{\text{sum}}(\pi^*) = 1 + \frac{1}{2}\epsilon n(n-1) + 2n\epsilon - \epsilon)$. Clearly $\overline{C}_{\text{sum}}(\pi^{KS2}) < \overline{C}_{\text{sum}}(\overline{\pi}^{KS2})$. Finally $\overline{C}_{\text{sum}}(\pi^{KS2})/\overline{C}_{\text{sum}}(\pi^*)$ tends to $n-1$ if $\epsilon \to 0$.

Since Example 5.8 can be applied to *all five* algorithms from [164] with the same result, we immediately have the following evaluation.

Theorem 5.6. For the algorithm A, $A \in \{KS1, \ldots, KS5\}$ we have

$$n - 1 \leq \eta^A(\overline{C}_{\text{sum}}) \leq n. \tag{5.25}$$

Although a more precise evaluation has been done, in the context of (5.25) this proof is of minor importance and therefore will be omitted.

Analysis of the algorithm $RC3$ and RCo has provided more promising result.

Theorem 5.7. For algorithm $RC3$ we have

$$\frac{2m}{3} + \frac{1}{3m} \leq \eta^{RC3}(\overline{C}_{\text{sum}}) \leq n. \tag{5.26}$$

Proof. We present only an example showing the lower bound in (5.26).

Example 5.9. Let $m \geq 2$. The job set consists of m subsets $J_i, i = 1, \ldots, m$. The subset i has cardinality $(2m - 2i + 1)q$, where q is an integer. Thus, the total number of jobs is $n = \sum_{i=1}^{m}(2m - 2i + 1)q$. The processing times are equal to $p_{ii} = \frac{2m-1}{2m-2i+1}, i = 1, \ldots, m$. All the remaining (undefined above) processing times are equal to zero. ◇

One can prove that the algorithm $RC3$ can generate the permutation $\pi^{RC3} = (1)_{(2m-1)q}(2)_{(2m-3)q}\ldots(m)_q$, then $\overline{C}_{\text{sum}}(\pi^{RC3}) = \frac{1}{n}\sum_{i=1}^{m}\sum_{j=1}^{(2m-2i+1)q}[(2m-1)q(i-1) + \frac{2m-1}{2m-2i+1}j] = \frac{1}{6n}(2m-1)q^2m(2m^2-3m+1) + \frac{1}{2n}(2m-1)qm(mq+1)$, see Fig. 5.9. The optimal permutation is $\pi^* = ((m)_1\ldots(2)_{2m-3}\,(1)_{2m-1})_q$, and $\overline{C}_{\text{sum}}(\pi^*) = \frac{1}{n}\sum_{j=1}^{q}[\sum_{i=1}^{m}(2m-2i+1)](2m-1)j = \frac{1}{2n}m^2(2m-1)q(q+1)$. It means that $\overline{C}_{\text{sum}}(\pi^{RC3})/\overline{C}_{\text{sum}}(\pi^*) = (\frac{2}{3}m + \frac{1}{3m} + \frac{1}{qm})/(1+\frac{1}{q})$ which tends to $\frac{2m}{3} + \frac{1}{3m}$ if $q \to \infty$. ■

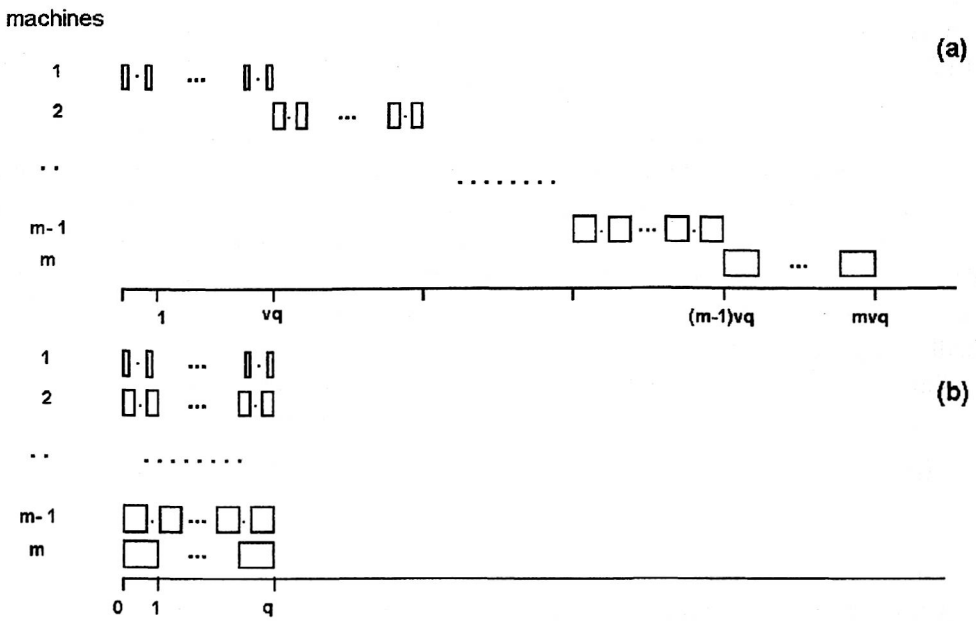

Figure 5.9: Schedules in Example 5.9: (a) for π^{RC3}, (b) for π^*. All jobs on the machine i have the index i and the processing time $p_{ii} = (2m-1)/(2m-2i+1)$, $v = 2m-1$

Theorem 5.8. For algorithm RCo we have

$$\frac{2m}{3} + \frac{1}{3} \leq \eta^{RCo}(\overline{C}_{\text{sum}}) \leq m. \tag{5.27}$$

Proof. First note that $\pi^m = \pi^F$ assuming $w_j = 1/n, j = 1, \ldots, n$. Therefore, from the definition of RCo and $\eta^F(\overline{C}_{\text{sum}})$, we have

$$\overline{C}_{\text{sum}}(\pi^{RCo}) \leq \overline{C}_{\text{sum}}(\pi^m) \leq m\overline{C}_{\text{sum}}(\pi^*) \tag{5.28}$$

which provides the upper bound in (5.27). The lower bound follows from the example.

Example 5.10. Let $m \geq 2$. The job set consists of m subsets $J_1, \ldots, J_m$, such that the set J_i has the cardinality $(m-i+1)q, i = 1, \ldots, m$, where q is an integer. The job processing times are equal to $p_{ii} = 1/(m-i+1), i = 1, \ldots, m$. All the remaining (undefined above) processing times are equal to zero. $\diamond$

Priority values for RCo can easily be found equal to $\omega_j^k = \min\{1, \frac{m-k+1}{m-j+1}\}$, $j = 1, \ldots, n$, $k = 1, \ldots, m$. Since $\omega_j^k \leq \omega_{j+1}^k, j = 1, \ldots, n-1$, for any $k =$

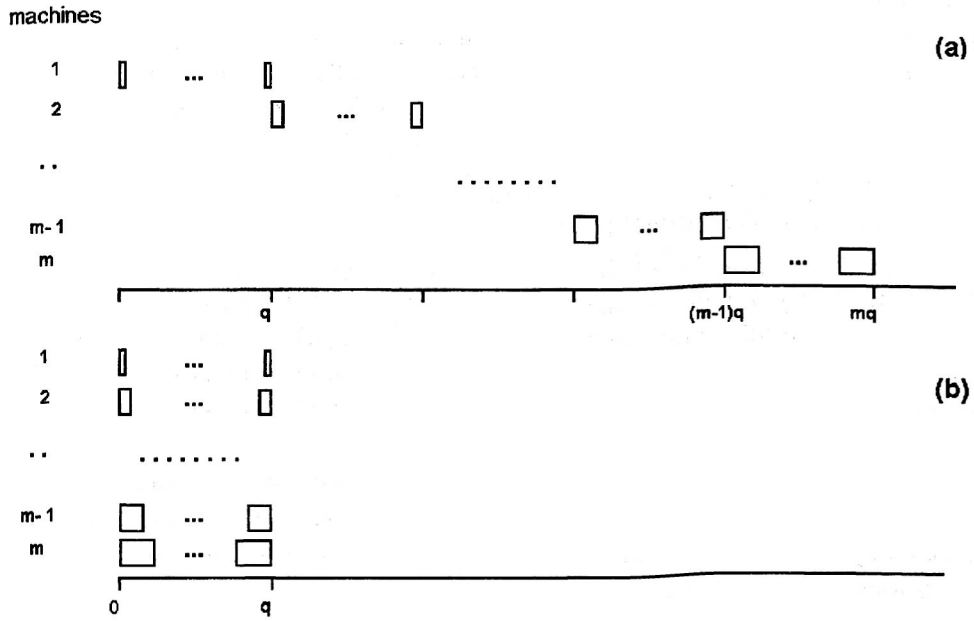

Figure 5.10: Schedules in Example 5.10: (a) for π^{RCo}, (b) for π^*. All jobs on the machine i have the index i and the processing time $p_{ii} = 1/(m-i+1)$

$1,\ldots,m$, all auxiliary permutations in RCo may have the same form $\pi^k = (1)_{mq}(2)_{(m-1)q}\ldots(m-1)_{2q}(m)_q, k = 1,\ldots,m$, see Fig. 5.10. Hence $\pi^{RCo} = (1)_{mq}\ldots(m)_q$ and $\overline{C}_{\text{sum}}(\pi^{RCo}) = \frac{1}{n}\sum_{i=1}^{m}\sum_{j=1}^{(m-i+1)q}[(i-1)q + \frac{j}{m-i+1}] = \frac{1}{12n}q^2m[(m+1)(2m+1)+6/q]$. The optimal permutation is $\pi^* = ((m)_1(m-1)_2\ldots(1)_m)_q$ and $\overline{C}_{\text{sum}}(\pi^*) = \frac{1}{n}\sum_{i=1}^{q}\sum_{j=1} m[(m-j+1)i] = \frac{1}{4n}m(m+1)q(q+1)$. Finally, one can check that the ratio $\overline{C}_{\text{sum}}(\pi^{RCo})/\overline{C}_{\text{sum}}(\pi^*)$ tends to $2m/3+1/3$ if $q \to \infty$. ■

In the context of Theorems 5.3, 5.6 and 5.7, we conclude that Algorithm Q/X for $\overline{C}_{\text{sum}}$ can use RCo or $RC3$ instead of X. Indeed for $m = 2$ we have $5/3 \leq \eta^{RCo}(\overline{C}_{\text{sum}}) \leq 2$. Since $\pi^2 = \pi^F$ and $\eta^F(\overline{C}_{\text{sum}}) = 2$, one can expect that the small value of $\eta^{RCo}(\overline{C}_{\text{sum}})$ is reached due to π^1. However, restricting our attention only to π^1 we also get this ratio as 2. Finally, using specific following example we can make our previous evaluations more precise.

Example 5.11. Let $m = 2$. The job set consists of four subsets J_1, J_2, J_3, J_4 of identical jobs. The job processing times and cardinalities of subsets are given in Table 5.1. ◇

Priority values ω_j^1 and ω_j^2, $j = 1,2,3,4$ for RCo are shown in Table 5.1. Thus, permutations generated by RCo may have the following form: $\pi^1 = (3)_{10q}(4)_{10q}(1)_{16q}(2)_{2q}$ and $\pi^2 = (1)_{16q}(3)_{10q}(4)_{10q}(2)_{2q}$. Since $16q + 2 \cdot 2q < 10q + 2 \cdot 10q$ then in π^1 all jobs from J_1 and J_2 finish at time $30q$. Therefore we have, $n\overline{C}_{\sum}(\pi^1) = \sum_{j=1}^{10q} j + \sum_{j=1}^{10q}(10q + 2j) + \sum_{j=1}^{18q}(30q) = 790q^2 + 15q$. Next, $n\overline{C}_{\sum}(\pi^2) = \sum_{j=1}^{16q} j + \sum_{j=1}^{10q}(16q + j) + \sum_{j=1}^{10q}(26q + 2j) + \sum_{j=1}^{2q}(46q) = 790q^2 + 23q$. We can show that, the optimal permutation is $\pi^* = (3,1)_{10q}(4,(1)_2)_{3q}(4,2)_{2q}(4)_{5q}$ and $n\overline{C}_{\sum}(\pi^*) = \sum_{j=1}^{10q}(2j) + \sum_{j=1}^{3q} 3(10q + 2j) + \sum_{j=1}^{2q} 2(16q + 2j) + \sum_{j=1}^{5q}(20q + 2j) = 414q^2 + 28q$. Finally, $\overline{C}_{\sum}(\pi^{RCo})/\overline{C}_{\sum}(\pi^*)$ tends to $\frac{790}{414} \approx 1.908$ if $q \to \infty$.

Although, we have only shown $1.908 \leq \eta^{RCo}(\overline{C}_{\sum}) \leq 2$ for $m = 2$, one can expect that η^{RCo} is close or simply equals 2. Example 5.11 also suggests that the lower bound on $\eta^{RCo}(\overline{C}_{\sum})$ for general m can be improved, however using special approaches for separate m.

For $m = 2$ we have $3/2 \leq \eta^{RC3}(\overline{C}_{\sum})$. Hence, one can hope that just $RC3$ owns the best worst-case performance evaluation. Unfortunately, besides a slightly improved lower bound on $\eta^{RC3}(\overline{C}_{\sum})$ we have not succeeded in further evaluations.

Example 5.12. Let $m = 2$. The job set consists of 2 subsets J_1, J_2 with cardinalities $174q$ for J_1 and $100q$ for J_2 where q is an integer. The job processing times are equal to $p_{11} = 1, p_{12} = 0, p_{21} = 0, p_{22} = 3$. $\diamond$

Let us start with $S = \emptyset$, i.e. $d = 0$ and $C_{i\sigma(d)} = 0$, $i \in M$. Completion times of any as yet unscheduled job $j \in J \setminus S$, which can be appended to the current partial permutation σ, have the following forms: $C_{1,j} = C_{2,j} = 1$ for $j \in J_1$, and $C_{1,j} = 0$, $C_{2,j} = 3$ for $j \in J_2$. Hence, $\omega_j''' = 3$, for any $j \in J \setminus S$. Consequently, $RC3$ can select $k = 1$ at the first step. This implies $d = 1$, $\sigma = (1)_1$, and $C_{1\sigma(d)} = C_{2\sigma(d)} = 1$. Next, since both machines are available at the same time moment 1, the previous step can be repeated, creating the partial permutation $\sigma = (1)_{174q}$. The remain jobs are uniform and therefore $RC3$ generates the permutation $\pi^{RC3} = (1)_{174q}(2)_{100q}$. It is easy to prove that $n\overline{C}_{\sum}(\pi^{RC'''}) = \sum_{i=1}^{174q} i + \sum_{i=1}^{100q}(174q + 3i) = 47,538q^2 + 237q$. The optimal permutation is $\pi^* = ((2)_1(1)_3)_{58q}(1)_{42q}$, and $n\overline{C}_{\sum}(\pi^*) = \sum_{j=1}^{58q} 4 \cdot 3i + \sum_{i=1}^{42q}(174q + 3i) = 30,138q^2 + 411q$. It means that $\overline{C}_{\sum}(\pi^{RC3})/\overline{C}_{\sum}(\pi^*)$ tends to $\frac{47,538}{30,138} \approx 1.577$ if $q \to \infty$.

The last result of this section refers to algorithm R.

Theorem 5.9. For the algorithm R, we have

$$n - (2 - \frac{4}{n+2}) \leq \eta^R(\overline{C}_{\sum}) \leq \eta^{CDS+API}(\overline{C}_{\sum}) \leq \eta^{CDS}(\overline{C}_{\sum}) \leq n. \quad (5.29)$$

Table 5.1: Data for the Example 5.8.

	J_j			
	1	2	3	4
p_{1j}	1	2	0	0
p_{2j}	0	0	1	2
$card$	$16q$	$2q$	$10q$	$10q$
ω_j^1	2	4	1	2
ω_j^2	1	2	1	2

Proof. Due to Property 5.1, we need to show only a lower bound.

Example 5.13. Let $m = 3$. The job set consists of three subsets J_1, J_2 and J_3 with cardinalities 1, 1 and $n-2$, respectively. The processing times are equal to $p_{31} = 1$, $p_{22} = \delta$, $p_{32} = p_{33} = p_{23} = \epsilon$, $p_{13} = 2\epsilon$, where ϵ is a sufficiently small number such that $\lim_{\epsilon \to 0} \epsilon n^2 = 0$ and $\delta = (1-\epsilon)/n$. All the undefined explicitly processing times are equal to zero. $\diamond$

One can prove that CDS provides the permutation $\pi^{CDS} = (1)_1(2)_1(3)_{n-2}$. This permutation can be improved by API neither in terms of C_{max} nor in $\overline{C}_{\text{sum}}$ criteria. It means that $\pi^{CDS} = \pi^{CDS+API} = \pi^R$. Hence $n\overline{C}_{\text{sum}}(\pi^{CDS}) = \sum_{j=1}^{n}[1+(j-1)\epsilon] = n+\frac{1}{2}\epsilon n(n-1)$. The optimal permutation $\pi^* = (3)_{n-2}(2)_1(1)_1$ yields $n\overline{C}_{\text{sum}}(\pi^*) = \sum_{j=1}^{n-2} 2\epsilon(j+1) + [2\epsilon(n-1)+\delta] + [2\epsilon(n-1)+\delta+1] = 2n\epsilon + 4\epsilon(n-2) + \epsilon(n-2)(n-1) + 2\delta + 1$. Finally, $\overline{C}_{\text{sum}}(\pi^R)/\overline{C}_{\text{sum}}(\pi^*)$ tends to $n - (2 - \frac{4}{n+2})$ if $\epsilon \to 0$, which is close to n. ∎

This surprising theoretical result undermines our notion of the wonderful quality of API. Its good behaviour observed in computer tests follows from the simplicity of API and the commonly known fact that "the best among several solutions is better than any single solution". By employing NPI procedure instead of API we increase the computational complexity of the algorithm, improve the solution performance observed in computer tests; however, we do not improve significantly the worst-case performance. A slight modification of Example 5.13 shows that also $CDS + NPI$ has the worst-case ratio of the order $O(n)$. Indeed, by setting $\delta = 1$ in Example 5.13 we obtain an example such that: (a) $\pi^{CDS} = (1)_1(2)_1(3)_{n-2}$, (b) $n\overline{C}_{\text{sum}}(\pi^{CDS}) = n+\frac{1}{2}\epsilon n(n-1)$, (c) π^{CDS} cannot be improved by NPI, hence $\pi^{CDS+NPI} = \pi^{CDS}$, (c) $\pi^* = (3)_{n-2}(2)_1(1)_1$, (d) $n\overline{C}_{\text{sum}}(\pi^*) = 2n\epsilon + 4\epsilon(n-2) + \epsilon(n-2)(n-1) + 3$, (e) $\overline{C}_{\text{sum}}(\pi^{CDS+NPI})/\overline{C}_{\text{sum}}(\pi^*)$ tends to $n/3$ if $\epsilon \to 0$. Therefore one can say that such a poor value of $\eta^R(\overline{C}_{\text{sum}})$

follows from the application of CDS at the initial stage. Note that both proposed examples can be improved using INS.

5.5.3 The makespan criterion

It follows from Table 5.6 that no approximation algorithm A with $\eta^A(C_{\max}) < \lceil m/2 \rceil$ has been found so far. The only suspected NEH has no such proof made up to now, unfortunately. Therefore, we have examined particularly precisely each new algorithm looking for a desired approach. Although we can immediately prove, by using Properties 5.4–5.5, that the bound $\eta^{T/Yk}(C_{max}) = m_{\max}\,\eta^{Yk}(C_{\max})$ is tight, nevertheless this result implies the existence of a competitive algorithm (see analogy to the Corollary) only in the context of satisfactorily good approximation algorithm for the k-machine problem for $k \geq 2$. Until such algorithm is found, new results referring to the existing algorithms, as for example, the given below evaluation of $\eta^{HC}(C_{\max})$, support us in the search.

Theorem 5.10. For the algorithm $CDS + HC$ we have

$$m/2 \leq \eta^{CDS+HC}(C_{\max}) \leq \lceil m/2 \rceil. \tag{5.30}$$

Proof. The upper bound in (5.30) follows immediately from the obvious inequality $K(\pi^{A+HC}; Z) \leq K(\pi^A; Z)$, and from the evaluation proved in [219] $C_{\max}(\pi^{CDS}; Z)/C_{\max}(\pi^*; Z) \leq \lceil m/2 \rceil$. To complete the proof, we provide a lower bound in (5.30) using the following example.

Example 5.14. This example is designed for $m \geq 7$. The job set consists of $2r$ job subsets. The job processing times and cardinalities of subsets are given in Table 5.2 (even m) and Table 5.5 (odd m). Table 5.2 also contains processing times of auxiliary two-machine flow-shops used by CDS. $\diamond$

Let us consider the case of "even m." According to the data in Table 5.2, CDS can generate the following sequence of permutations: $\pi^1 = (2)_q \ldots (r)_q (r+1)_1 \ldots (m)_1 (1)_q$, $\pi^k = (k+1)_q \ldots (r)_q (m-k+1)_1 \ldots (m)_1 (1)_q \ldots (k)_q (r+1)_1 \ldots (m-k)_1$ for $k = 2, \ldots, r-1$, $\pi^r = (r+1)_1 \ldots (m)_1 (1)_q \ldots (r)_q$, and $\pi^k = (m-k+1)_q \ldots (r)_q (k+1)_1 \ldots (m)_1 (r+1)_1 \ldots (k)_1 (1)_q \ldots (m-k)_q$ for $k = r+1, \ldots, m-1$. To avoid time-consuming calculations of $m-1$ makespans $C_{\max}(\pi^k), k = 1, \ldots, m-1$, required by CDS, observe that the makespan value $C_{\max}(\pi^k)$ cannot be lower than the maximal nondecreasing subsequence of job indexes chosen from π^k. From π^k (see also Table 2) we have $C_{\max}(\pi^1) \geq m-1$, $C_{\max}(\pi^k) \geq \max\{m-2k, r\}$ for $k = 2, \ldots, r-1$, and $C_{\max}(\pi^k) \geq r$ for $k = r, \ldots, m-1$. One can prove that at least $C_{\max}(\pi^{r-1}) = r$, and thus CDS can generate $\pi^{CDS} = \pi^{r-1}$. This schedule is shown in Figure 5.11. For the simplicity of notation, π^{CDS} will

Table 5.2: Job processing times and cardinalities for Example 5.14 for even m; $\epsilon = 1/q$. The lower part contains processing times for the auxiliary two-machine flow shops

	subsets of jobs									
machines	1	2	...	$r-1$	r	$r+1$	$r+2$	...	$m-1$	m
1	ϵ									
2		ϵ				ϵ	ϵ	...	ϵ	ϵ
$\vdots$			$\ddots$							
$r-1$				ϵ						
r					ϵ					
$r+1$						1				
$r+2$							1			
$\vdots$								$\ddots$		
$m-1$									1	
m										1
card	q	q	...	q	q	1	1	...	1	1
a^1	ϵ	0	...	0	0	0	0	...	0	0
b^1	0	0	...	0	0	0	0	...	0	1
a^2	ϵ	ϵ	...	0	0	ϵ	ϵ	...	ϵ	ϵ
b^2	0	0	...	0	0	0	0	...	1	1
$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
a^{r-1}	ϵ	ϵ	...	ϵ	0	ϵ	ϵ	...	ϵ	ϵ
b^{r-1}	0	0	...	0	0	0	1	...	1	1
a^r	ϵ	ϵ	...	ϵ	ϵ	ϵ	ϵ	...	ϵ	ϵ
b^r	0	0	...	0	0	1	1	...	1	1
a^{r+1}	ϵ	ϵ	...	ϵ	ϵ	$1+\epsilon$	ϵ	...	ϵ	ϵ
b^{r+1}	0	0	...	0	ϵ	1	1	...	1	1
$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
a^{m-2}	ϵ	ϵ	...	ϵ	ϵ	$1+\epsilon$	$1+\epsilon$	...	ϵ	ϵ
b^{m-2}	0	0	...	ϵ	ϵ	1	1	...	1	1
a^{m-1}	ϵ	ϵ	...	ϵ	ϵ	$1+\epsilon$	$1+\epsilon$	...	$1+\epsilon$	ϵ
b^{m-1}	0	ϵ	...	ϵ	ϵ	$1+\epsilon$	$1+\epsilon$	...	$1+\epsilon$	$1+\epsilon$

Table 5.3: Block matrix of subtractions $b_j^{m-1} - a_i^{m-1}$ for Example 5.14 (even m)

i/j	1 ... r	$r+1$ ... $m-1$	m
1	$-\epsilon$	$-\epsilon - 1$	ϵ
2 ⋮ r	0	-1	0
$r+1$ ⋮ m	1	0	1

$\boxed{x}$ - denotes a matrix of an appriopriate dimension having all elements equal x $x \in \{-1, 0, 1\}$.

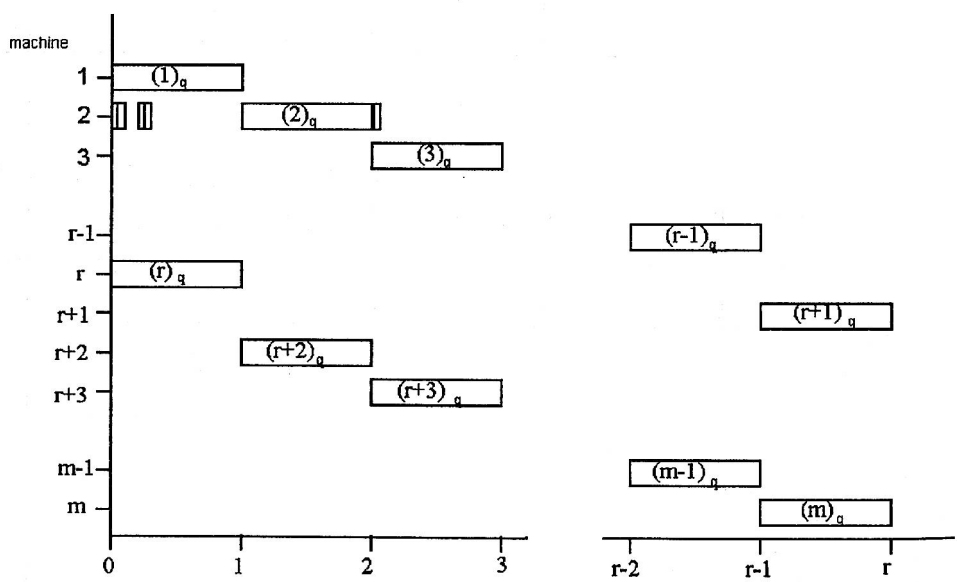

Figure 5.11: Gantt Chart for schedule in Example 5.14

be called π in the sequel. Figure 1 also illustrates the obvious decomposition of π into two disjointed subsequences $\pi = \underline{\pi}\overline{\pi}$ where $\underline{\pi} = (r)_q(r+2)_1 \ldots (m)_1$, $\overline{\pi} = (1)_q \ldots (r-1)_q(r+1)_1$, $C_{\max}(\underline{\pi}) = r$, and $C_{\max}(\overline{\pi}) = r$. Any permutation σ such that at least one of these subsequences is preserved in σ has the makespan $C_{\max}(\sigma)$ not less than $C_{\max}(\pi)$. This property is a direct consequence of the so-called *block properties* [99].

Now we are ready to analyse the performance of $CDS + HC$. Instead of precise study we will only show that HC cannot improve π. Let us begin with the calculation of revised gaps (5.5). To this end, the values $b_j^{m-1} - a_i^{m-1}$ found from Table 5.2 have been formed in the block matrix shown in Table 5.3. Sets L_{ij} are defined for $i = 1, \ldots, m$ as follows: (a) for $j = 1, \ldots, r, L_{ij} = \{j\}$ if $i \neq j-1$, (b) for $j = r+1, \ldots, m-1, L_{ij} = \{2, j\}$ if $3 \neq i \neq j-1, L_{ij} = \{j\}$ if $i = 3, L_{i,j} = \{2\}$ if $i = j-1$, and (c) $L_{im} = \{2\}$ if $i \neq 3$. All the remaining (undefined above) sets are empty. Finally, the revised gaps are given in Table 5.4. The matrix d_{ij}^R has a block structure with some exceptions observed in the first row, last column, sub-diagonal, and the right half of the third row. Elements of this matrix are defined as follows $z = 0.9/(m-2)$, (1) for $j = 1, 2, \ldots, r$ we have $\alpha_j = z(j-1)\epsilon, \alpha'_j = \alpha_j - \epsilon, \gamma_j = \alpha_j + 1$, and (2) for $j = r+1, \ldots, m-1$ we have $\beta_j = z(j-1) + \alpha_2 - 1, \beta'_j = \beta_j - \epsilon, \beta''_j = \beta_j - \alpha_2, \delta_j = \beta_j + 1$. By virtue of the definition and after simple calculations we introduce some dependences, crucial for HC: (a) $\alpha_j \geq 0, \alpha'_j < 0, j = 1, \ldots, r$, (b) $\beta'_j < \beta''_j < \beta_j < 0, j = r+1, \ldots, m-1$, for sufficiently small $\epsilon < (m-2)/9$, (c) all sequences $\alpha_j, \alpha'_j, \beta'_j, \beta''_j, \beta_j$, are increasing, (d) $|\alpha_r| < \left|\alpha'_j\right|, j = 1, \ldots, r-1$, and $|\alpha_r| < \left|\beta'_{r+1}\right|$. The selection of X (and thus g) corresponds to a searching for an element in an appropriate row, while the selection of Y (and h) corresponds to a searching for an element in a column of the matrix d_{ij}^R.

Set initially $a = 1$ and $b = n$. At first, the row r and the column $r+1$ are searched. As a result we obtain $X = \alpha_r$, which refers to job r located nearby position a in the block $(r)_q$, and $Y = \beta'_{r+1}$ which refers to job 1 in the block $(1)_q$. From (a), (b) and (d) we have $X > 0, Y < 0, |X| < |Y|$. It means that HC jumps to Step 5. This interchange operates on jobs 1 and $r-1$ located in the subsequence $\overline{\pi}$ and thus no improvement of the makespan occurs. Therefore job 1 remains on its position, and b decreases. At the second step, row r (without the element $r+1$) and column $r-1$ (without the element $r+1$) are searched. Thus we have $X = \alpha_r > 0$, which refers to job r (as the above), and $Y = \alpha'_{r-1} < 0$, which refers to job 1 in the block $(1)_q$. Similarly we have $X > 0, Y < 0, |X| < |Y|$, and HC jumps to Step 5. The interchange operates on jobs 1 and $r-1$ and for the similar reason cannot improve the makespan. Subsequent q steps have the same character, so finally the sequence $(r-1)_q(r+1)_1$ will be fixed.

Table 5.4: Revised gaps d_{ij}^R for Example 5.14 (even m)

i/j	1	2	$\cdots$	v	$\cdots$	$r-1$	r	$r+1$	$r+2$	$\cdots$	w	$\cdots$	$m-2$	$m-1$	m
1	α'_1	α'_2	$\cdots$	α'_v	$\cdots$	α'_{r-1}	α'_r	β'_{r+1}	β'_{r+2}	$\cdots$	β'_w	$\cdots$	β'_{m-2}	β'_{m-1}	α'_2
2	0	α_2	$\cdots$	α_v	$\cdots$	α_{r-1}	α_r	β_{r+1}	β_{r+2}	$\cdots$	β_w	$\cdots$	β_{m-2}	β_{m-1}	α_2
3	α_1	0		α_v	$\cdots$	α_{r-1}	α_r	β''_{r+1}	β''_{r+2}	$\cdots$	β''_w	$\cdots$	β''_{m-2}	β''_{m-1}	0
4	α_1	α_2		α_v	$\cdots$	α_{r-1}	α_r	β_{r+1}	β_{r+2}	$\cdots$	β_w	$\cdots$	β_{m-2}	β_{m-1}	α_2
$\vdots$	$\vdots$	$\vdots$				$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$		$\vdots$	$\vdots$	$\vdots$
$v+1$	α_1	α_2		0		α_{r-1}	α_r	β_{r+1}	β_{r+2}	$\cdots$	β_w	$\cdots$	β_{m-2}	β_{m-1}	α_2
$\vdots$	$\vdots$	$\vdots$				$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$		$\vdots$	$\vdots$	$\vdots$
$r-1$	α_1	α_2	$\cdots$	α_v		α_{r-1}	α_r	β_{r+1}	β_{r+2}	$\cdots$	β_w	$\cdots$	β_{m-2}	β_{m-1}	α_2
r	α_1	α_2	$\cdots$	α_v		0	α_r	β_{r+1}	β_{r+2}	$\cdots$	β_w	$\cdots$	β_{m-2}	β_{m-1}	α_2
$r+1$	γ_1	γ_2	$\cdots$	γ_v	$\cdots$	γ_{r-1}	1	$\times$	δ_{r+2}	$\cdots$	δ_w	$\cdots$	δ_{m-2}	δ_{m-1}	γ_2
$r+2$	γ_1	γ_2	$\cdots$	γ_v	$\cdots$	γ_{r-1}	γ_r	α_2	$\times$	$\cdots$	δ_w	$\cdots$	δ_{m-2}	δ_{m-1}	γ_2
$r+3$	γ_1	γ_2	$\cdots$	γ_v	$\cdots$	γ_{r-1}	γ_r	δ_{r+1}	α_2		δ_w	$\cdots$	δ_{m-2}	δ_{m-1}	γ_2
$\vdots$	$\vdots$	$\vdots$		$\vdots$		$\vdots$	$\vdots$	$\vdots$		$\ddots$			$\vdots$	$\vdots$	$\vdots$
$w+1$	γ_1	γ_2	$\cdots$	γ_v	$\cdots$	γ_{r-1}	γ_r	δ_{r+1}	δ_{r+2}		α_2		δ_{m-2}	δ_{m-1}	γ_2
$\vdots$	$\vdots$	$\vdots$		$\vdots$		$\vdots$	$\vdots$	$\vdots$	$\vdots$			$\ddots$		$\vdots$	$\vdots$
$m-1$	γ_1	γ_2	$\cdots$	γ_v	$\cdots$	γ_{r-1}	γ_r	δ_{r+1}	δ_{r+2}	$\cdots$	δ_w		α_2	$\times$	γ_2
m	γ_1	γ_2	$\cdots$	γ_v	$\cdots$	γ_{r-1}	γ_r	δ_{r+1}	δ_{r+2}	$\cdots$	δ_w	$\cdots$	δ_{m-2}	α_2	$\times$

$z = 0.9/(m-2), \alpha_j = z(j-1)\epsilon \geq 0, \alpha'_j = \alpha_j - \epsilon < 0, \beta_j = z(j-1) + \alpha_2 - 1 < 0, \beta'_j = \beta_j - \epsilon < 0, \beta''_j = \beta_j - \alpha_2 < 0, \gamma_j = \alpha_j + 1, \delta_j = \beta_j + 1.$

Repeating the latter steps we obtain the sequence $(1)_q \ldots (r-1)_q (r+1)_1 = \overline{\pi}$ fixed. Since all further interchanges preserve $\overline{\pi}$ within each tested permutation, we never improve the makespan. On the other hand the optimal permutation is $\pi^* = (n)_1 (n-1)_1 \ldots (r+1)_1 (r)_q (r-1)_q \ldots (2)_q (1)_q$ and $C_{\max}(\pi^*) = r\epsilon + 1$. It means that $C_{\max}(\pi^{HC})/C_{\max}(\pi^*) \geq r/(r\epsilon + 1)$ which tends to r if $\epsilon \to 0$.

By analogy, to the example from Table 5.5 (odd m) we can conclude that $C_{\max}(\pi^{HC})/C_{\max}(\pi^*)$ tends to $m/2$ if $\epsilon \to 0$. This completes the proof. ∎

5.6 Conclusions

From the worst-case point of view the following conclusions can be drawn.

The worst-case performance ratio for the algorithm producing random permutation when applicable to problems with criterion $C_{\max}$ is equal to m, when applicable to problems with criterion $\overline{C}_{\text{sum}}$ is equal to n, whereas for C_{sum} it can be arbitrarily large, see also Table 5.6.

Table 5.5: Job processing times for Example 5.10 (odd m); $\epsilon = 1/q$.

	subsets of jobs									
machines	1	2	...	$r-1$	r	$r+1$	$r+2$	...	m	$m+1$
1	ϵ									
2		ϵ					ϵ	...	ϵ	ϵ
$\vdots$			$\ddots$							
$r-1$				ϵ						
r					ϵ	1				
$r+1$							1			
$\vdots$								$\ddots$		
$m-1$									1	
m										1
card	$2q$	$2q$	...	$2q$	q	1	2	...	2	2

Surprisingly, for the problems with the criterion C_{sum}, a relatively simple algorithm F is better than quite complex $CDS + HC$. The algorithm Q/X developed "by analogy" to CDS offers performance also "by analogy" $\eta^{Q/X}(C_{\text{sum}}) = \lceil m/2 \rceil \eta^X(C_{\text{sum}})$, where $\eta^X(C_{\text{sum}})$ is the worst-case performance ratio of an algorithm used for solving the two-machine flow-shop problem. In spite of external analogy, the particular auxiliary two-machine problems, solved in the internal work of this algorithm, do not manifest any analogous behaviour. This fact and the presented worst-case instances allow us to formulate another general conclusion: any overlooking of some data (even though they are really incomplete) deteriorates significantly the worst-case performance evaluation. Having in mind the performance of $RC3$, new algorithms for the problem with the criterion C_{sum} should be designed as an extension of ideas taken from this algorithm.

For problems with the criterion $\overline{C}_{\text{sum}}$, there exist approximation algorithms with the worst-case performance ratio equal to or less than m, see Table 5.6. Although evaluations of the performance obtained for the algorithms $RC3$ and RCo are not tight, we have though that for $RC3$ it is closer to $2m/3 + O(1/m)$ than to m. Proof of this fact seems to be hard, since up to now there has been no clear methodology of analysing constructive algorithms with dynamics $(RC3, NEH)$. From among algorithms recommended for this criterion, RCo and $RC3$ are better than $RC1, RC2$ and $CDS + HC$ (this result also has been confirmed by experimental analysis), being as good as F and Q/X. For $m = 2$ the algorithm with the worst-case performance ratio less than 2 is still loking for.

Table 5.6: Lower $\underline{\eta}^A$ and upper $\overline{\eta}^A$ bounds (provided in E) of the worst-case performance ratio η^A of an algorithm A (developed in B) for various scheduling criteria K. (New results are marked by an asterisk (*))

K	A	B	$\underline{\eta}^A$	$\overline{\eta}^A$	E
$C_{\max}$	any		m	m	[96]
	R	[264]	$\lceil m/2 \rceil$	$\lceil m/2 \rceil$	[264]
	CDS	[35]	$\lceil m/2 \rceil$	$\lceil m/2 \rceil$	[219]
	RA	[52]	$m/\sqrt{2}+O(1/m)$	$m/\sqrt{2}+O(1/m)$	[225]
	$RACS, RAES$	[52]	$m/\sqrt{2}+O(1/m)$	$m/\sqrt{2}+O(1/m)$	[226]
	P	[239]	$m/\sqrt{2}+O(1/m)$	$m/\sqrt{2}+O(1/m)$	[226]
	NEH	[212]	$\sqrt{m/2}+O(1/m)$	$(m+1)/2$	[226]
	HR	[148]	$m/\sqrt{2}+O(1/m)$	$m/\sqrt{2}+O(1/m)$	[228]
	G	[124]	$m-1$	$m-1$	[228]
	TG	[183]	$(m+1)/2$	$(m+1)/2$	[183]
	IE, M	[238]	m	m	[183]
	$KS1, KS2$	[164]	m	m	[183]
	$CDS+HC$	[140]	$m/2$	$\lceil m/2 \rceil$	*
	T/Yk	*	$\lceil m/k \rceil \eta^{Yk}(C_{max})$	$\lceil m/k \rceil \eta^{Yk}(C_{max})$	*
$\overline{C}_{\text{sum}}$	any		n	n	[96]
	SPT	[96]	m	m	[96]
	RCo	[253]	$2m/3+1/3$	m	[303]
	$RCo, m=2$	[254]	1.908	2	*
	$RC1, RC2$	[254]	n	n	[303]
	$RC3$	[254]	$2m/3+1/(3m)$	n	[303]
	$RC3, m=2$	[254]	1.577	n	*
	$HK, m=2$	[144]	$2b/(a+b)$	$2b/(a+b)$	[144]
	$KS1, \ldots, KS5$	[164]	$n-1$	n	*
	$R, CDS+API$	[257]	$n-(2-4/(n+2))$	n	*
	$CDS+NPI$	*	$n/3$	n	*
C_{sum}	any		$1+(n-1)(\overline{w}/\underline{w})$	$1+(n-1)(\overline{w}/\underline{w})$	*
	$CDS, G, P, RA,$ $CDS+HC,$ $G+HC, P+HC,$ $RA+HC$	[96]	$1+(n-1)(\overline{w}/\underline{w})$	$1+(n-1)(\overline{w}/\underline{w})$	*
	F	*	m	m	*
	Q/X	*	$\lceil m/2 \rceil \eta^{X}(C_{\text{sum}})$	$\lceil m/2 \rceil \eta^{X}(C_{\text{sum}})$	*
	T/Yk	*	$\lceil m/k \rceil \eta^{Yk}(C_{\text{sum}})$	$\lceil m/k \rceil \eta^{Yk}(C_{\text{sum}})$	*

The simple transfer of the ideas accepted for makespan problems to problems with other scheduling criteria (namely to those based on a sum of penalties) usually produces less efficient algorithms and generally should be avoided. Particularly, these algorithms should not be used to prepare initial solution for improvements algorithms with other than $C_{\max}$ criteria.

Constructive algorithms for the C_{sum} and $\overline{C}_{\text{sum}}$ problems should be designed as an extension of methods developed for the single-machine problems with these criteria. Dynamic priority rules based on a simple job fitness (e.g. between job delays) are usually worse than those based on a weighted combination of between job delays combined with the sum of completion times. Improvements algorithms should contain procedures API (NPI) as well as INS.

For the makespan criterion the algorithm HC is as good as CDS, and is worse than NEH (NEH still remains the champion). For other criteria HC is as good as any algorithm producing random permutation.

Results obtained in [226] for algorithms $RACS$ and $RAES$ as well as results obtained in this paper for HC, $CDS+API$, $CDS+NPI$ and R confirm a general hypothesis that improvement algorithms are as good as auxiliary algorithms used for finding starting solutions.

Clearly, the analysis carried out in this paper for more than ten various algorithms does not complete the worst-case research. The results obtained pinpoint the drawbacks of some existing algorithms, introduce a reference level for comparisons, provides supplementary characteristics of algorithm behaviour and outline directions of the new and better approaches to future research.

Chapter 6

Zero-inventory policy

JIT forces the decrease of in-process inventory by elimination (limitation) either internal/external storage place or storage time. With respect to storage place these constraints usually take the form of "no storage place is allowed", "storage place is limited" by means of the number of stored parts or the size (volume) of stored parts, "total number of items circulated in the system is limited", "number of items of the same type circulated in the system is limited", or "store must contain a limited number of parts" where store's capacity is limited from bottom to top. With respect to storage time they take the form of "parts must go without waiting", "waiting time between successive stages is limited", or "waiting times immediately influence on the production costs" [1]. Two special classes of these constraints are considered as crucial for introducing ZI control, namely no-store (NS) and zero-wait (ZW) policies. Note that pure JIT strategy assumes no in-process storage. In processes where ZW has no obligatory character (because of technology requirements), ZW policy is much more restrictive than ZS.

Although the above constraints are very easily formulated, not many theoretical results are known for these problems and the generality level of the considered problems remains relatively low, see the up-to-date survey of Hall and Sriskandarayah [136]. There are a lot of papers dealing with flow-shop problems with the so-called blocking constraints (NS policy), see survey [136] supplemented by papers [206, 53, 115, 118, 265], limited storage [206, 252, 55, 170] and with no-wait constraints [243, 155, 23]. Slightly more complex problems are considered only in the context of simple priority heuristics, Kuriyan and Reklaitis [177], Sawik [273, 274], hierarchical decomposition Sawik [271, 272], studies of special cases Kubiak [171], or simulation studies. These problems also increasingly preoccupy the attention of many researches because of their applicability in FMS systems, Sawik [275, 276]. Recently, an excellent approach to modelling quite broad class

[1] These constraints also appear frequently in CIM system and FMS.

of FMS problems has been proposed by Toczyłowski [319]. This approach can also be employed for modelling some problems with limited storage constraints. We also refer to our earlier studies Smutnicki [292, 296, 297], Nowicki and Smutnicki [227].

Generally, the methodology of modelling the NS and ZW problems enter the phase of development, particularly for more complex production systems. Even for the simple flow line with single machine per each stage and NS policy there exist several different models, but simultaneously there are no papers with the critical comparison of their applicability. Simultaneously, there are no models which can be applied to more general production structures with NS or limited storage (LS) policy. Additionally, the number of specific properties which can be useful for the solution algorithms still remains poor. Either no numerical results are provided or the reported numerical results are far from making possible a solution with high accuracy instances of greater sizes existed in practice. Particularly significant lack is observed in the very profitable local search methods and exact methods which might minimise the manufacturing cycle time.

In this chapter we provide a common approach to modelling and solving problems without (with limited) in-process storage with the methodology oriented towards the efficient local search methods. This approach will be presented on the base of some case study, a flexible flow-line scheduling problem, which is immediately usable in practice and general enough to show some nontrivial properties. The applied technique of modelling is adjusted to make easy the extending to general problems, namely these with non-uniform machines, machine set-ups, transport time, etc., and those with more general flow-production structure, as for example, job shop or network. Although the delivery performance will not be discussed in this section (since the main stress has been laid on the storage problem), the proposed approach, models and algorithms can be extended in an easy way taking into account, for example, a measure of delivery performance from previous sections. Nevertheless, we should be conscious that the final numerical properties of the algorithm depend on theoretical properties of the problem, which in turn, depend significantly on the chosen measure of delivery performance.

6.1 Introduction

The verbal description of the problem is as follows. There is a set of parts (elements, customer's orders) and a set of processing cells (service centers) each of which has a set of parallel identical machines (service stages), Figure 6.1. A part is associated with a sequence of operations processed in successive cells, and

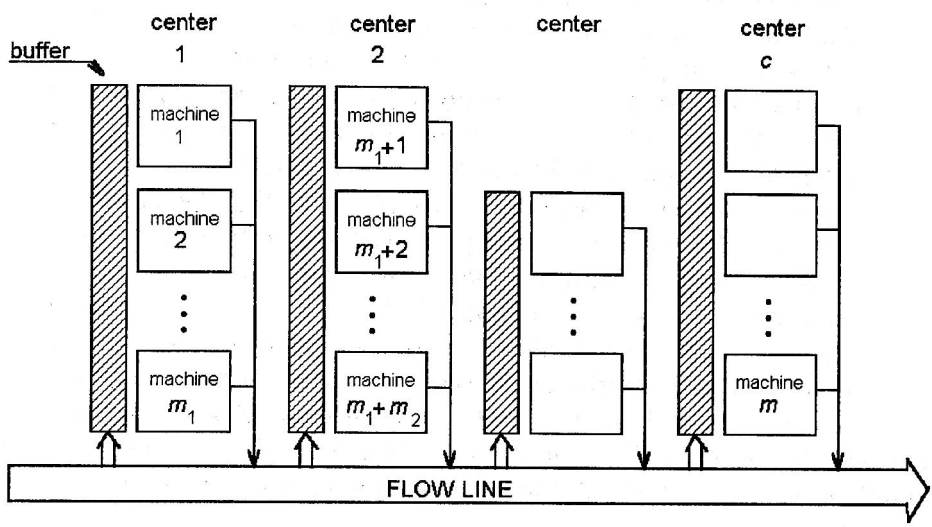

Figure 6.1: Flexible flow line

all parts flow through cells in the same order. In a cell, a part can be processed on any machine. There can be a transport time of a part between cells. Moreover there are buffers with limited capacity that mediate in transferring parts between machines/cells. We want to find a schedule that minimises the makespan, one of the most frequently used criterion. This problem, called hereafter FP, can be considered as a hybrid combination of two classic scheduling problems, namely the flow shop and parallel shop problem.

Architecture of automated manufacturing systems usually does not allow the formation of internal queues of parts (transported on pallets or in containers), or limits the length of these queues due to *buffer* size. With respect to FP, the following problems (with an increasing description complexity) can be considered: (U) system has buffers with infinite capacity or finite however large enough, (W) system works without buffers, (B) system works with buffers of finite capacity. In the system B, buffers can be designed as: buffer to machine (M), buffer to cell (C), common (shared) buffer for all machine/cells (G), dedicated part buffer before machine (PM), dedicated part buffer before cell (PC). Buffers can also be located on the output of the cell (machine), however, due to symmetry, we will not consider this case separately. The case (G) is relative to the buffering performed by automatic-guided-vehicle (AGV) transportation system or a common (single) store of limited capacity. The case (PG) is relative to the Kanban controlled

system. Note that pallets can be interpreted as a specific class of store. If the capacity of a buffer is greater than one, we should also consider the buffer service rule. There can be distinguished at least three rules, which are as follows: (1) first-in-first-out (F) where parts go for processing in the same order in which they enter the buffer, (2) arbitrary order (A) where the next part for processing is selected arbitrarily among those waiting in the buffer, and (3) extended arbitrary (E) where the next part for processing can be selected arbitrarily among these waiting in the buffer extended by those waiting on blocked machines of the preceding stage. In the last case, we permit a part to pass over the buffer, immediately on the next stage. Clearly, the rule applied should be adjusted to the phisical construction of the buffer and should reflect the general service policy used in the manufacturing process. Intuitively, the case (F) is the most restrictive, whereas (E) the least restrictive. From the methodological point of view, the case (E) is significantly harder than (F). Thus each considered problem, besides introduced assumptions, can be characterised by a triple (system type)–(buffer type)–(buffer service rule).

FP creates the basic model for a broad class of problems called in the literature the *flexible flow-line scheduling*. These are pure problem FP or problems obtained directly from FP by introducing a few specific additional assumptions, e.g. some parts can skip one or more machines during the route, buffers have infinite capacity, transport time is negligible, etc. A detailed review of industrial applications, among others in chemical branches, polymer and petroleum industry, computer systems, telecommunication networks, FMS, spaceship processing, etc., one can find in [149], [211].

Although FP have quite simple formulation, it is troublesome from the algorithmic point of view. Its NP-hardness essentially restricts the set of approaches which can be applied to solve the problem. From the literature one can find exact algorithms based on the branch-and-bound ($B\&B$) [284] and mixed-integer programming (MIP), as well as a variety of approximation methods, see the review in [149]. Some of these procedures have been designed for special cases, e.g. for the systems that have only two cells, the ones where only one of the cells has parallel machines, etc.; see the newest paper in this field [43]. Algorithms for FP have also been considered in [273, 337]. Research outcomes show that $B\&B$ algorithms become useless for more than 10 parts. Similarly, the size of MIP models is impractically large even for a small number of parts and cells. Therefore, more attention has been paid recently to the approximation methods. Currently, only a few *constructive* algorithms applicable to FP are known [61, 149, 273, 337]. Surprisingly, up to now there is no *improving* algorithm, although many recent papers have recommended the local search approach as the most promising for very hard optimisation problems [86, 322].

Table 6.1: Data for the Instance.

j	e_j	c_j	p_j	j	e_j	c_j	p_j	j	e_j	c_j	p_j	j	e_j	c_j	p_j
1	1	1	60	4	2	1	30	7	3	1	40	10	4	1	30
2	1	2	60	5	2	2	10	8	3	2	30	11	4	2	40
3	1	3	30	6	2	3	40	9	3	3	40	12	4	3	50
13	5	1	20	16	6	1	30	19	7	1	80				
14	5	2	90	17	6	2	50	20	7	2	40				
15	5	3	70	18	6	3	30	21	7	3	100				

6.2 Basic model – infinite capacity buffers

The system has m machines located in c machine cells, and let $\mathcal{M} = \{1, \ldots, m\}$ be the set of machines, $\mathcal{C} = \{1, \ldots, c\}$ be the set of cells. The cell $l \in \mathcal{C}$ has $m_l \geq 1$ parallel identical machines, and let $\mathcal{M}_l = \{k_l + 1, \ldots, k_l + m_l\} \subseteq \mathcal{M}$ be the set of machines in this cell where $k_l = \sum_{i=1}^{l-1} m_i$. The set of e parts (elements, customer orders) $\mathcal{E} = \{1, 2, \ldots, e\}$ has to be processed in this system. Each part $k \in \mathcal{E}$ corresponds to a set of o_k operations $\mathcal{O}_k = \{l_k + 1, \ldots, l_k + o_k\}$ processed in that order, where $l_k = \sum_{i=1}^{k-1} o_i$, and let $\mathcal{O} = \bigcup_{k=1}^{e} \mathcal{O}_k = \{1, \ldots, o\}$ be the set of all operations that have to be processed. Operation j corresponds to a processing of part e_j[2] at cell c_j, can be performed on any machine from $\mathcal{M}_{c_j}$ and requires $p_j > 0$ time units for processing. Once started, operations cannot be interrupted. Each machine can execute at most one part at a time, each part can be processed on at most one machine at a time, and each part k flows through the system so that $c_{j-1} < c_j$ for $j - 1, j \in \mathcal{O}_k$. A *feasible schedule* is defined by a couple of vectors (S, P), $S = (S_1, \ldots, S_o)$, $P = (P_1, \ldots, P_o)$, such that the above constraints are satisfied, where operation j is started on machine $P_j \in \mathcal{M}_{c_j}$ at time $S_j \geq 0$.

To illustrate the problem, notions and denotations, we will use through this chapter a sequence of Examples, each of them employs the same data given in Instance.

Example 6.15. A feasible schedule for the Instance has been shown in Gantt Chart, Fig. 6.2. For example, operation 2 of part 1 at cell 2 is performed on machine 3, is started at time 130 and completed at time 190, i.e. $P_2 = 3$, $S_2 = 130$. The makespan is 310. $\diamond$

Instance. The system has $m = 3$ machine cells with 2 machines at the first cell, 2 at the second cell, and 3 at the third cell, i.e. $m_1 = m_2 = 2$, $m_3 = 3$. The

[2] $e_j = k$ for any $j \in \mathcal{O}_k$.

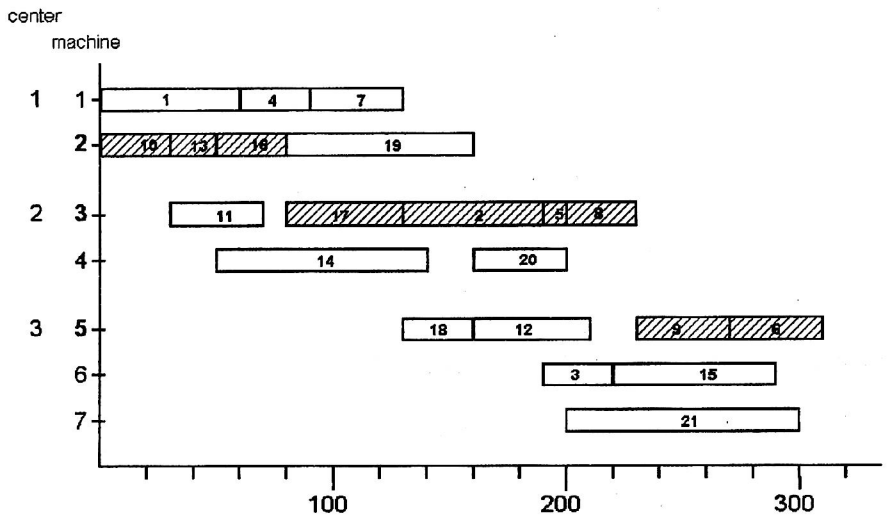

Figure 6.2: Gantt Chart for Example 6.1

set of $n = 7$ parts with processing times given in Table 6.1 has to be processed in this system. •

To provide a formal mathematical model of the problem we introduce some notions. Let $\mathcal{L}_l = \{j \in \mathcal{O} : \ c_j = l\}$ be the set of operations that have to be processed in a cell $l \in \mathcal{C}$. *Machine workload* is a subset of operations assigned to a machine in a cell, however, due to interchangeable machine identity it is not associated with any particular machine. To determine all machine workloads, each set $\mathcal{L}_l$ has to be partitioned into m_l subsets $\mathcal{N}_i \subset \mathcal{L}_l$, $i \in \mathcal{M}_l$, and let $n_i = |\mathcal{N}_i|$, $i \in \mathcal{M}_l$, $l \in \mathcal{C}$. The *machine processing order* is prescribed by a permutation of operations $\pi_i = (\pi_i(1), \ldots, \pi_i(n_i)) \in \mathcal{P}(\mathcal{N}_i)$ where $\pi_i(k)$ denotes the element of $\mathcal{N}_i$ which is in position k in π_i, and $\mathcal{P}(\mathcal{N}_i)$ is the set of all permutations on the set $\mathcal{N}_i$. The *overall processing order* is defined by m-tuple $\pi = (\pi_1, \ldots, \pi_m)$. All such processing orders create the set $\Pi = \{\pi = (\pi_1, \ldots, \pi_m) : \ (\pi_i \in \mathcal{P}(\mathcal{N}_i), \ i \in \mathcal{M}), \ ((\mathcal{N}_i, \ i \in \mathcal{M}_l) \text{ is a partition of the set } \mathcal{L}_l, \ l \in \mathcal{C})\}$.

Next, let us consider relations between π and a feasible schedule. Assume that π is given. For each $j \in \mathcal{O}$ we define *machine* predecessor/successor $\underline{s}_j, \overline{s}_j$ and *technological* predecessor/successor $\underline{t}_j, \overline{t}_j$ as follows: $\underline{s}_{\pi_i(j)} = \pi_i(j-1)$ for $j = 2, \ldots, n_i$ and $\underline{s}_{\pi_i(1)} = o$ (null); $\overline{s}_{\pi_i(j)} = \pi_i(j+1)$ for $j = 1, \ldots, n_i - 1$ and $\overline{s}_{\pi_i(n_i)} = o$; $\underline{t}_j = j - 1$ if $j > 1$ and $e_{j-1} = e_j$, and $\underline{t}_j = o$ otherwise; $\overline{t}_j = j + 1$ if $j < o$ and $e_{j+1} = e_j$, and $\overline{t}_j = o$ otherwise. For this π, starting times S_j and the

completion times C_j of operations have to satisfy the following clear constraints

$$S_j \geq 0,\ j \in \mathcal{O}, \tag{6.1}$$

$$C_{\underline{t}_j} \leq S_j,\ j \in \mathcal{O};\ \underline{t}_j \neq \circ, \tag{6.2}$$

$$C_{\underline{s}_j} \leq S_j,\ j \in \mathcal{O};\ \underline{s}_j \neq \circ, \tag{6.3}$$

$$C_j = S_j + p_j,\ j \in \mathcal{O}. \tag{6.4}$$

These constraints lead to an obvious recursive formula on starting times

$$S_j = \max\{S_{\underline{t}_j} + p_{\underline{t}_j}, S_{\underline{s}_j} + p_{\underline{s}_j}\} \tag{6.5}$$

where $S_\circ = 0 = p_\circ$, which allow us to find them in a time $O(o)$. The appropriate completion times can be found using equation (6.4). The makespan value associated with π will be denoted by $C_{\max}(\pi)$, and thus $C_{\max}(\pi) = \max_{j\in\mathcal{O}} C_j$.

To determine the feasible schedule from the given $\pi \in \Pi$, we have to assign permutations π_i, $i \in \mathcal{M}_l$, to machines $i \in \mathcal{M}_l$ at cell l, $l \in \mathcal{C}$ [3]. Since machines at a cell are identical, the assignment can be carried out in any way. Therefore, one can obtain a feasible schedule (S, P) in the following manner: set $P_{\pi_i(j)} = i$ for $j = 1, \ldots, n_i$, $i \in \mathcal{M}$, and find S_j by using formulae (6.5). It is clear that any π represents $\prod_{l=1}^{c}(m_l)!$ feasible schedules with the same makespan value and various assignments of π_i to machines. Finally, we can state our problem as that of finding $\pi \in \Pi$ which minimises $C_{\max}(\pi)$.

Example 6.16. For the Instance we decide to perform operations in cells in the following manner: $\pi_1 = (1,4,7)$, $\pi_2 = (10,13,16,19)$, $\pi_3 = (11,17,2,5,8)$, $\pi_4 = (14,20)$, $\pi_5 = (18,12,9,6)$, $\pi_6 = (3,15)$, $\pi_7 = (21)$. The overall processing order is $\pi = (\pi_1, \ldots, \pi_7)$. This processing order generates 2!·2!·3!=24 feasible schedules with the same makespan value. One of these schedules is shown in Figure 6.2. Starting times and completion times follow from (6.5) and (6.4). ◇

In the analysis we will refer to an auxiliary graph model associated with a fixed $\pi \in \Pi$. The graph $\mathcal{G}(\pi) = (\mathcal{O}, \mathcal{A}^* \cup \mathcal{A}(\pi))$ has a set of nodes $\mathcal{O}$ and a set of arcs $\mathcal{A}^* \cup \mathcal{A}(\pi)$ where

$$\mathcal{A}^* = \bigcup_{j\in\mathcal{O};\,\underline{t}_j\neq\circ} \{(\underline{t}_j, j)\} \tag{6.6}$$

and

$$\mathcal{A}(\pi) = \bigcup_{j\in\mathcal{O};\,\underline{s}_j\neq\circ} \{(\underline{s}_j, j)\}. \tag{6.7}$$

[3] Machine workload allow us to reduce the number of considered feasible schedules.

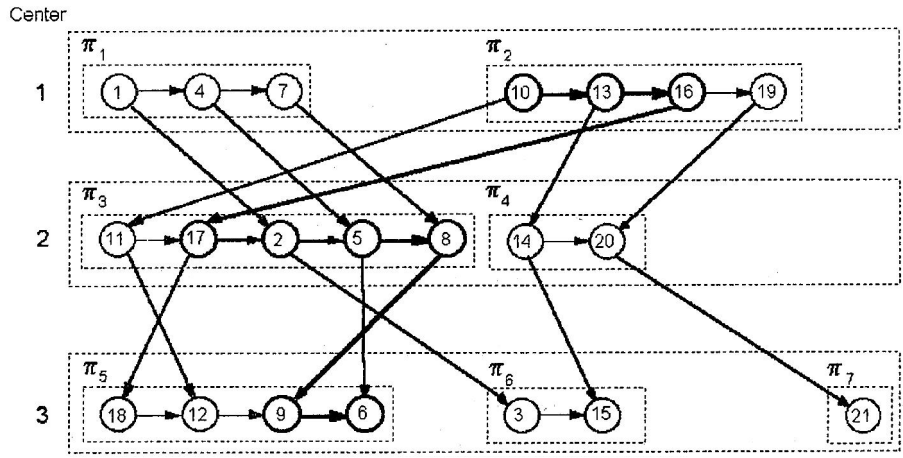

Figure 6.3: Graph $\mathcal{G}(\pi)$ for Example 6.2

Arcs $\mathcal{A}^*$ proceed from constraints (6.2), (6.4), represent the route of parts through the cells and have the weight zero. Arcs $\mathcal{A}(\pi)$ proceed from constraints (6.3), (6.4), represent the machine processing orders and have the weight zero. Each node $j \in \mathcal{O}$ represents the operation j, event "starting" S_j of this operation, and has the weight p_j. Starting time S_j is equal to the length of the longest path to node j (without p_j) in $\mathcal{G}(\pi)$, whereas completion time C_j is equal to the length of the longest path to node j (including p_j) in this graph. The makespan $C_{\max}(\pi)$ equals the length of the longest path (critical path) in $\mathcal{G}(\pi)$.

Let us consider a critical path in $\mathcal{G}(\pi)$ interpreted as a sequence of nodes (operations) $U = (u_1, \ldots, u_w)$, $u_j \in \mathcal{O}$. Each subsequence of successive operations $B_{ab} = (u_a, \ldots, u_b)$ from U such that $P_{u_{a-1}} \neq P_{u_a} = \ldots = P_{u_b} \neq P_{u_{b+1}}$ we call the *block* [4]. We also denote $\mathcal{U} = \{u_1, \ldots, u_w\}$, $\mathcal{B}_{ab} = \{u_a, \ldots, u_b\}$, and for any $j \in \mathcal{O}$ we define x_j so that $\pi_{P_j}(x_j) = j$.

Although notions P_j, x_j, n_i, $\underline{s}_j$, $\overline{s}_j$, U, $\mathcal{U}$, B_{ab}, $\mathcal{B}_{ab}$ depend on π, however, for the sake of simplicity we do not express this fact explicitly. In the sequel, if several overall processing orders will be used simultaneously, these notions will refer to π by default.

Example 6.17. The graph $\mathcal{G}(\pi)$ for the Example 6.2 is given in Figure 6.3. The critical path is $U = (10, 13, 16, 17, 2, 5, 8, 9, 6)$, $w = 9$. There are three blocks on this path $B_{1,3} = (10, 13, 16)$, $B_{4,7} = (17, 2, 5, 8)$, $B_{8,9} = (9, 6)$. $\diamond$

[4] We say $P_{u_{a-1}} \neq P_{u_a}$ if $a = 1$, $P_{u_b} \neq P_{u_{b+1}}$ if $b = w$.

6.3 Related problems

Assuming $c = 1$ we obtain the classic problem of scheduling parts on parallel identical machines to minimise makespan, see Section 3.1 for more comprehensive discussion of this subject.

Assuming $m_l = 1, l \in \mathcal{C}$ (thus $c = m$) we obtain the classic flow-shop problem, for which a lot of exact and approximation algorithms have been developed. The searched schedule can be either *permutation* (processing orders on all machines are identical) or *overall* (no additional assumption is made). Industrial applications of this model are presented in the last chapter of this book. This model is also considered as a fundamental in developing advantageous approximation algorithms for more complex systems, e.g. considered here problem with infinite capacity buffers. A significant progress has been observed recently in this field. First, the worst-case analysis made for the known and some new algorithms have introduced an objective, instant-independent list of ranks, see Table 5.6. Second, due to this list and due to experimental results, there is selected the champion algorithm (NEH) and the most fruitful approach – an *insertion technique*, Nawaz et al. [212]. Next, a skilful combination of insertion technique with tabu search approach has provided fast and easily implementable algorithms with excellent solution performance, which allow us to solve problems up to 10,000 operations on a PC in a reasonable time, Nowicki and Smutnicki [231]. At the end, there are also designed more powerful B&B methods, successfully run on instances up to 2,000 operations, [236, 323]. Assuming all cells non-bottleneck but k-th, one can obtain a problem of scheduling operations on parallel identical machines with ready times and delivery times to minimise maximum makespan, see also Section 3.1.

Several results have been obtained in other special-cases, chiefly for the systems having two cells with one non-bottleneck. These results are summarized in Table 6.2.

6.4 Schedule characterisation

From Section 6.2 it follows that any schedule (S, P) can be represented in a unique way by an overall processing order $\pi \in \Pi$, and any π generates a subset of schedules equivalent to it if the makespan is taken into account. In this section, we discuss some relations to other known in the literature characteristics of the schedule in the flexible flow line, namely the permutation processing order, cell loading sequences and cell input/output sequences. The last one is strictly associated with the *m-processor* notion, a convenient tool of modelling and analysing

Table 6.2: The worst-case performance ratio η^A of an algorithm A (developed in B) for mixed single- and multiple- machine cells and the makespan criterion

A	B	c	case	η^A
LS	[311]	2	$m_1 = 1, m_2 \geq 2$	$3 - 1/m_2$
LS	[311]		$m_1 = \ldots = m_{c-1} = 1, m_c \geq 2$	$c + 1 - 1/m_c$
J	[311]	2	$m_1 = 1, m_2 = 2$	2
J	[311]	2	$m_1 = 1, m_2 \geq 3$, $C_{\max}(\pi^J) \leq \sum_{j \in \mathcal{L}_1} p_j + \max_{j \in \mathcal{L}_2} p_j$	2
J	[311]	2	$m_1 = 1, m_2 \geq 3$, $C_{\max}(\pi^J) > \sum_{j \in \mathcal{L}_1} p_j + \max_{j \in \mathcal{L}_2} p_j$	$1 + (2 - \frac{1}{m_2})(1 - \frac{1}{m_2})$
A	[311]	2	$m_1 = m_2 \geq 2$	$3 - 1/m_2$
A	[311]		$m_1 = \ldots = m_c \geq 2$	$c + 1 - 1/m_c$
B	[311]	2	$m_1 = m_2 \geq 2$	$(7/3 - \frac{2}{3m_2}, 3 - \frac{1}{m_2})$
B'	[311]	2	$m_1 = m_2 \geq 2$	$(2, 3 - 1/m_2)$
CLS	[43]	2		$2 - 1/\max\{m_1, m_2\}$

some complex scheduling problems, Toczyłowski [319]. Nevertheless, the model formulated in Section 6.2 has been chosen consciously after analysing profits and disadvantages of other available approaches.

In the classic flow-shop problem, there exist two clearly distinguished cases: the permutation flow-shop problem in which all machines process the parts in the same order, and the general flow-shop problem in which various machines can process the parts in different orders. The primal case has been introduced chiefly to facilitate algorithm construction in order to satisfy some rare technological requirements and/or to simplify the rule of service part queues. Nevertheless, the latter (significantly harder) case usually provides solutions with the lower makespan value, and from this point of view it seems to be more attractive. Most of the papers have considered only the primal case, treating the latter one marginally, which however is not fully justified.

Among the algorithms formulated for FP one can find a class of methods generating the so-called *permutation processing order*, Ding and Kittichartphayak [61]. Although this notion has already been introduced and used, principal differences between permutation and non-permutation cases of this problem have not been discussed by the authors. As we will show below, these notions can be defined and understood in a various manners.

We call $\pi \in \Pi$ the *weak permutation processing order* if there exists a permutation $\sigma_* \in \mathcal{P}(\mathcal{E})$ such that sequence $(e_{\pi_i(1)}, \ldots, e_{\pi_i(n_i)})$ is a subsequence of σ_* for any $i \in \mathcal{M}$. This definition prohibits the parts from passing over, i.e. if for some

machine a part i precedes a part j then there is no such machine that allows a part j to precede a part i. To check whether a given π is the weak permutation processing order we can proceed as follows. Create the graph $(\mathcal{E}, \mathcal{J}(\pi))$ having the set of nodes $\mathcal{E}$ and the set of arcs $\mathcal{J}(\pi) = \bigcup_{i\in\mathcal{M}} \bigcup_{j=1}^{n_i-1} \{(e_{\pi_i(j)}, e_{\pi_i(j+1)})\}$. If this graph does not contain a cycle then π is the weak permutation processing order. The permutation $\sigma_* = (\sigma_*(1), \ldots, \sigma_*(e)) \in \mathcal{P}(\mathcal{E})$ required in the definition can be found in $O(o)$ time as the *order of nodes* such that "if there exists a path between nodes $\sigma_*(i)$ and $\sigma_*(j)$ then it has to be $i < j$".

Example 6.18. Job processing order from Example 6.2 is not the weak permutation processing order. A weak permutation processing order for the machine workload given in Example 6.2 is e.g. $\pi_1 = (1,4,7)$, $\pi_2 = (10,13,16,19)$, $\pi_3 = (11,17,2,5,8)$, $\pi_4 = (14,20)$, $\pi_5 = (12,18,6,9)$, $\pi_6 = (3,15)$, $\pi_7 = (21)$. This order yields the makespan value of 310. $\diamond$

For each $\sigma \in \mathcal{P}(\mathcal{E})$ there exist $\prod_{l=1}^{c}[((m_l)^e - \sum_{k=1}^{m_l-1}\binom{m_l}{k}k^e)/(m_l)!]$ processing orders $\pi \in \Pi$ such that none of the sets $\mathcal{N}_i$, $i \in \mathcal{M}$, is empty and π is generated by σ. For example, for $c = 3$, $m_1 = m_2 = m_3 = 2$ and $e = 10$ this number exceeds 10^8. Thus, one can say that finding a weak permutation processing order is relatively easy due to large freedom. On the other hand, the arbitrary restriction of the considered processing orders only to those being the permutation in the weak sense can lead to extremely bad makespans. Let us consider the example of the system with 3 cells, single machine at cells 1 and 3, and two machines at cell 2, i.e. $c = 3$, $m_1 = m_3 = 1$, $m_2 = 2$. In this system, we have to process e parts of two types. There are $e-1$ parts of the first type and a single part of the second type. Each part of the first type has processing times equal to $p_1 = p_3 = 1/(n-1)$, $p_2 = \epsilon$. The part of the second type has processing times equal to $p_4 = p_6 = \epsilon$, $p_5 = 1$. It is easy to prove that if $\epsilon \to 0$ the best weak permutation processing order provides the makespan approximately twice longer than the best non-permutation order. One can say that the flexibility of the system has been completely destroyed (or not applied) in this case.

We call permutation $\tau_l \in \mathcal{P}(\mathcal{L}_l)$ the *list sequence of cell l* if $S_{\tau_l(j)} \leq S_{\tau_l(j+1)}$, $j = 1, \ldots, l_l - 1$ where $l_l = |\mathcal{L}_l|$. The *loading sequence of cell l* is determined by the sequence of parts $\rho_l \in \mathcal{P}(\mathcal{E})$, $\rho_l = (e_{\tau_l(1)}, \ldots, e_{\tau_l(l_l)})$. Each τ_l corresponds one-to-one to ρ_l. Note that τ_l does not determine directly and univocally the machine workload at cell l, thus does not determine univocally the overall processing order π. In practice, the final assignment of operations to machines is performed by an auxiliary list schedule algorithm LS (with ceratin rule ML of machine loading) which uses τ_l on its input. Obviously, the obtained results strongly depend on the applied rule ML. This dependence is clearly visible if we admit machine set-up times and/or the use of unrelated machines.

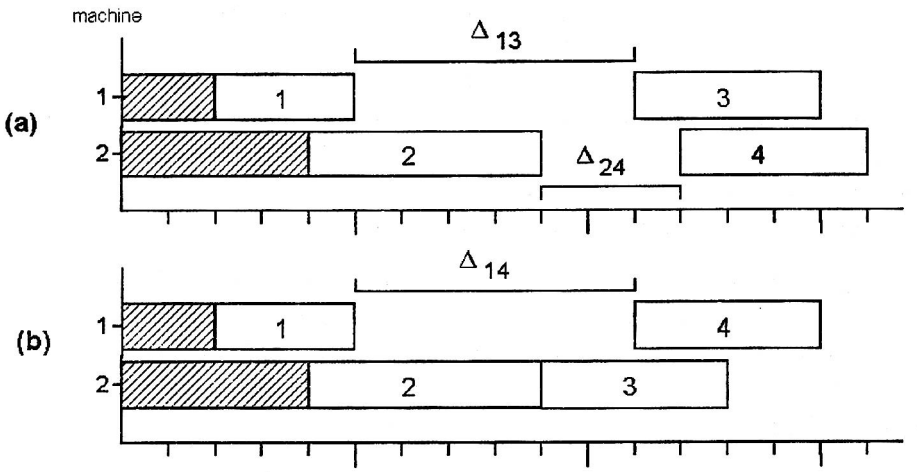

Figure 6.4: Input/output sequence representation with machine set-ups

At the end, let us discuss a representation of the schedule by the input/output cell sequence [319]. Since the proposed tool is more powerful than we need, we restrict our attention only to those aspects that are applicable to the problem stated. This approach is designed for cell consisting of a number of identical parallel machines called m-processor [5], and by definition cannot be extended to the case of unrelated machines. For each cell there are defined two sequences of operations: the *input sequence* $\tau_l \in \mathcal{P}(\mathcal{L}_l)$ and the *output sequence* $\omega_l \in \mathcal{P}(\mathcal{L}_l)$. These sequences introduce only the order of events "starting" and "completing", namely $S_{\tau_l(j)} \leq S_{\tau_l(j+1)}$, $j = 1, \ldots, l_l - 1$, and $C_{\omega_l(j)} \leq C_{\omega_l(j+1)}$, $j = 1, \ldots, l_l - 1$. The approach does not determine explicitly the machine workload, assuming that one can be found univocally. To prevent cell overloading, there are introduced additional constraints, which takes in our case the follwing form

$$C_{\omega_l(j)} + \delta_{\omega_l(j)\tau_l(j+m_l)} \leq S_{\tau_l(j+m_l)} \tag{6.8}$$

where $\delta_{\omega_l(j)\tau_l(j+m_l)}$ is the change over time between the operations $\omega_l(j)$ and $\tau_l(j)$. We will show that the last assumption does not hold if we consider the classic machine set-up times, so the representation of the schedule by input/output sequences cannot be extended even to the case of identical parallel machines with machine set-up times. To this end, let us consider the example shown in Fig. 6.4, see case (a). Two identical parallel machines in a cell process four

[5] It can also be applied to modelling of buffers, pallets, etc.

operations starting from a non-zero primal fill (marked as shadow). We define only some necessary machine set-up times $\Delta_{13} = 2d$, $\Delta_{24} = d$, $\Delta_{23} = 0$, $\Delta_{14} = 2d$, where d is a constant. Assuming $\tau = (1,2,3,4)$ and $\omega = (1,2,3,4)$, by virtue of (6.8) we have two inequalities to prevent cell overload, namely $C_1 + \delta_{13} \leq S_3$ and $C_2 + \delta_{24} \leq S_4$. Concentrate on δ_{13}. There is an unsolved problem how to set this value. By setting $\delta_{13} = \Delta_{13} = 2d$ we obtain the situation shown in Fig. 6.4 (a). Due to setting this value to d or zero, operation 3 shifts to the left on machine 1 and provides tottally incorrect solution. However, the solution from Fig. 6.4 (a) is also incorrect, since the proper solution for this instance is provided in Fig. 6.4 (b). Observe, Fig. 6.4 (b) provides another solution with exactly the same input and output cell sequences, so the same sequences can lead to many various schedules. However, the problem of finding, for the given sequences, the proper schedule has a combinatorial character. In the considered example the difference $S_3 - C_1$ depends on the "history", therefore δ_{13} need to own some "strange" value found as a result of the previously scheduled operations.

We call $\pi \in \Pi$ the *strong permutation processing order* if there exists a permutation $\tau_* \in \mathcal{P}(\mathcal{E})$ such that the sequence $(e_{\tau_1}, \ldots, e_{\tau_{l_l}})$ is a subsequence of τ_* for any $l \in \mathcal{C}$. This definition forces a unique order in all queues of jobs waiting for cells, i.e. if for some queue the part i precedes the part j that there is no queue that the part j precedes the part i. To check whether a given π is the strong permutation processing order we can make as follows. Create the graph $(\mathcal{E}, \mathcal{J}(\pi))$ having the set of nodes $\mathcal{E}$ and the set of arcs $\mathcal{J}(\pi) = \bigcup_{l \in \mathcal{C}} \{(e_i, e_j) : i,j \in \mathcal{L}_l,\ S_i < S_j\}$ If this graph does not contain a cycle then π is the strong permutation processing order. The permutation τ_* required in the definition can be found as the order of nodes in this graph.

If $m_l = 1$, $l \in \mathcal{C}$, then both definitions are equivalent.

6.5 Algorithms for the basic model

In this section we propose and discuss the solution algorithms based on a local search approach, which is currently the best known technique for constructing approximation algorithms of various hard optimisation problems, see Section 2.3. Two of these approaches, namely simulated annealing and tabu search, have been analysed in detail, since they can apply immediately some theoretical properties developed for neighbourhoods used in the search.

6.5.1 Neighbourhoods

Based on conclusions drawn by Nowicki and Smutnicki [230, 231], there are considered only neighbourhoods based on the so-called *insertion moves*. In this case,

such a type of moves is associated with π and can be defined by a triple (j, i, y), which means that operation j is removed from the π_{P_j} and inserted in the position y in the permutation π_i (y becomes a new position of j in π_i extended in such a way). Clearly, it has to be $i \in \mathcal{M}_{c_j}$, $1 \leq y \leq n_i + 1$ if $i \neq P_j$, $1 \leq y \leq n_i$ if $i = P_j$.

The neighbourhood of π consists of the orders π_v generated by the moves v from a given set. Using natural, simple approach one can propose the *insertion neighbourhood* $\{\pi_v : v \in \mathcal{V}(\pi)\}$ generated by the move set

$$\mathcal{V}(\pi) = \bigcup_{j \in \mathcal{O}} \bigcup_{i \in \mathcal{M}_{c_j}} \mathcal{V}_{ji}(\pi) \tag{6.9}$$

where $\mathcal{V}_{ji}(\pi) = \bigcup_{y=1}^{n_i+1}\{(j, i, y)\}$ if $i \neq P_j$ and $\mathcal{V}_{ji}(\pi) = \bigcup_{y=1}^{n_i}\{(j, i, y)\}$ if $i = P_j$ [6]. The set $\mathcal{V}_{ji}(\pi)$ contains moves such that operation j is deleted from the permutation π_{P_j} and inserted in all possible positions in the permutation π_i. One can prove that the number of moves in $\mathcal{V}(\pi)$ equals approximately oe. The neighbourhood $\{\pi_v : v \in \mathcal{V}(\pi)\}$ possesses the connectivity property that guarantees the possibility of finding globally optimal solution, see Section 2.3.

Example 6.19. The move set $\mathcal{V}(\pi)$ for the Example 6.2 is shown in Table 6.3. It contains 140 moves marked by white and black bullets. Moves marked by dots are redundant. $\diamond$

The final quality of the local search algorithm depends among others on the computational complexity of the single neighbourhood search and on the neighbourhood size. This is highly important for those local search methods which tested entirely the neighbourhood. Our previous research shows that such algorithms behave much better if we evaluate descendants directly by the criterion value, [230, 231, 232]. Therefore $\mathcal{V}(\pi)$ is considered as highly time-consuming. Hence we introduce a new notion, i.e. *reduced neighbourhood* which contains "the most promising part" of the original one.

The main idea of the reduction consists in eliminating some moves from $V(\pi)$ for which it is known a priori (without computing the makespan) that they will not immediately improve $C_{\max}(\pi)$. By employing certain graph property "if there exists in $\mathcal{G}(\pi_v)$ a path containing all nodes from $\mathcal{U}$, where $\mathcal{U}$ is found for $\mathcal{G}(\pi)$, we have $C_{\max}(\pi_v) \geq C_{\max}(\pi)$" we propose the following reduced set of moves

$$\mathcal{W}(\pi) = \bigcup_{j \in \mathcal{U}} \bigcup_{i \in \mathcal{M}_{c_j}} \mathcal{W}_{ji}(\pi), \tag{6.10}$$

[6] Set $\mathcal{V}(\pi)$ is examined to eliminate redundant moves, i.e. moves that provide the same solutions.

Table 6.3: Sets $V(\pi)$ and $W(\pi)$ of moves $v = (j, i, y)$ for Example 6.2

j	$i=1$						$i=2$						e_j	c_j	P_j	x_j	block
	$y=1$	2	3	4	5	6	1	2	3	4	5						
1	·	o	o				o	o	o	o	o		1	1	1	1	
4	·	·	o				o	o	o	o	o		2	1	1	2	
7	o	·	·				o	o	o	o	o		3	1	1	3	
10	•	•	•	•			·	o	•	•			4	1	2	1	$B_{1,3}$
13	•	•	•	•			·	·	•	•			5	1	2	2	$B_{1,3}$
16	•	•	•	•			•	·	·	o			6	1	2	3	$B_{1,3}$
19	o	o	o	o			o	o	·	·			7	1	2	4	

j	$i=3$						$i=4$						e_j	c_j	P_j	x_j	block
	$y=1$	2	3	4	5	6	1	2	3	4	5						
11	·	o	o	o	o		o	o	o				4	2	3	1	
17	·	·	•	•	•		•	•	•				6	2	3	2	$B_{4,7}$
2	•	·	·	o	•		•	•	•				1	2	3	3	$B_{4,7}$
5	•	•	·	·	•		•	•	•				2	2	3	4	$B_{4,7}$
8	•	•	•	·	·		•	•	·	•			3	2	3	5	$B_{4,7}$
14	o	o	o	o	o	o	·	o					5	2	4	1	
20	o	o	o	o	o	o	·	·					7	2	4	2	

j	$i=5$						$i=6$					$i=7$		e_j	c_j	P_j	x_j	block
	$y=1$	2	3	4	5	6	1	2	3	4	5	1	2					
18	·	o	o	o			o	o	o			o	o	6	3	5	1	
12	·	·	o	o			o	o	o			o	o	4	3	5	2	
9	o	·	·	•			•	•	•			•	•	3	3	5	3	$B_{8,9}$
6	•	•	·	·			•	•	•			•	•	2	3	5	4	$B_{8,9}$
3	o	o	o	o	o		·	o				o	o	1	3	6	1	
15	o	o	o	o	o		·	·				o	o	5	3	6	2	
21	o	o	o	o	o		o	o	o			·		7	3	7	1	

where $\mathcal{W}_{ji}(\pi) \subseteq \mathcal{V}_{ji}(\pi)$ are defined only for $j \in \mathcal{B}_{ab} \subseteq \mathcal{U}$ in the following manner. If $a = b$, we put $\mathcal{W}_{ji}(\pi) = \emptyset$. If $i \neq P_j$ and $a \neq b$ then $\mathcal{W}_{ji}(\pi) = \mathcal{V}_{ji}(\pi)$. In the remaining cases, i.e. if $i = P_j$ and $a \neq b$, we set: $\mathcal{W}_{ji}(\pi) = \mathcal{V}_{ji}(\pi) \setminus \mathcal{X}_j(x_{u_a} + 1, x_{u_b} - 1)$ if $j \in \mathcal{B}_{ab} \setminus \{u_a, u_b\}$, where

$$\mathcal{X}_j(k, l) = \bigcup_{y=k}^{l} \{(j, P_j, y)\}. \tag{6.11}$$

Next we set $\mathcal{W}_{ji}(\pi) = \mathcal{V}_{ji}(\pi) \setminus \mathcal{X}_j(1, x_j)$ if $j = u_a$; $\mathcal{W}_{ji}(\pi) = \mathcal{V}_{ji}(\pi) \setminus \mathcal{X}_j(x_j, n_i)$ if $j = u_b$. The strongest reduction is possible for $i = P_j$ and $a \neq b$: if $a = 1$ and $j \neq u_b$, we can set $\mathcal{W}_{ji}(\pi) = \mathcal{V}_{ji}(\pi) \setminus \mathcal{X}_j(1, x_{u_b} - 1)$; by symmetry if $b = w$ and $j \neq u_a$, we can set $\mathcal{W}_{ji}(\pi) = \mathcal{V}_{ji}(\pi) \setminus \mathcal{X}_j(x_{u_a} + 1, n_i)$.

Example 6.20. The move set $\mathcal{W}(\pi)$ for the Example 6.2 is shown in Table 6.3. It contains 53 moves marked by black bullets. ◇

Although principally the problem from [232] does not allow incomplete part routes (a part must pass through all cells), however, by direct analogy from [232], one can derive the following theoretical results. First, the proposed reduction of the neighbourhood size is significant.

Property 6.1. $|\mathcal{V}(\pi)| \,/\, |\mathcal{W}(\pi)| \approx o/w$. □

Assuming uniform distribution of machines over cells and uniform distribution of parts over machines, we have $o/w \approx m$. Indeed, a reduction of the neighbourhood size observed in computational experiments was equal (with a small error) to the number of machines in the system. Second, the reduction is advantageous since the skipped part of the neighbourhood is "less interesting" from the immediate improvements point of view.

Property 6.2. For any overall processing order π_v, $v \in \mathcal{V}(\pi) \setminus \mathcal{W}(\pi)$, we have $C_{\max}(\pi_v) \geq C_{\max}(\pi)$. □

Next, the reduced neighbourhood preserves the connectivity property that guarantees the possibility of finding optimal solution.

Property 6.3. For any $\pi^1 \in \Pi$ there exists a finite sequence $\pi^1, \ldots, \pi^r$ such that $\pi^{i+1} \in \{\pi_v : v \in \mathcal{W}(\pi^i)\}$ for $i = 1, \ldots, r-1$, and π^r is an optimal overall processing order. □

The specific structure of the proposed neighbourhood allows us to search it exactly (using makespans) in a very efficient manner.

Property 6.4. Makespans for all π_v, $v \in W(\pi)$ can be found in a time $O(ew)$. □

This last result is rather shocking. Since there are $O(ew)$ neighbours in the reduced neighbourhood, we conclude that each $C_{\max}(\pi_v)$ can be found in a time $O(1)$, instead of the originally required $O(o)$. Note, Property 6.4 can be applied only while calculating *all* neighbours from $\mathcal{W}(\pi)$ but not for a *separate* neighbour. Since the calculation of makespans for all neighbours consumes almost 100% of the algorithm's running time, the application of this property is crucial for the speed of the algorithm. For example, using the Property 6.4 for the instances with $o = 3000$ one can observe in experiments the reduction of a running time by about 1500 times. It is better than we ever can imagine!

6.5.2 Simulated annealing algorithm

In this section we analyse the impact of the results from Section 6.5.1 on the application of SA approach. We consider some variants of SA algorithm comparing them with the tabu search method presented in the next subsection. All examined variants are based on the scheme proposed by Osman and Potts [237].

A neighbourhood (one of those from Section 6.5.1) of the current solution π is searched in a random way using uniform distribution. Two cooling schemes have been examined: (1) logarithmic $T_{i+1} = T_i/(1+\beta T_i)$, (2) logarithm-periodic $T_{i+1} = T_i/(1+\beta T_i)$ if $T_i \geq T^*$ and $T_{i+1} = T_1$ in other cases, where T^* is a threshold value, $i = 1, \ldots, I-1$, and I is the assumed number of iterations. For each scheme we need to set T_1 and β, and for scheme (2) additionally T^*.

Three implementations with different computational complexity were considered:

(V) $\mathcal{V}(\pi)$ is used. For each neighbour chosen at random, the makespan is calculated immediately by using (6.5).

(W) $\mathcal{W}(\pi)$ is used. For each neighbour chosen at random, the makespan is calculated immediately by using (6.5).

(W+) $\mathcal{W}(\pi)$ is used. First, we detect whether π has been visited immediately before. If so, for each neighbour chosen at random, we check whether this neighbour has already been visited. If not, its makespan is calculated by using (6.5) and stored. If so, the stored already makespan is taken. The cost of computations decreases if a solution is repeated more than ew times that has been observed for a low temperature.

The variant (W+) is a new proposition followed from an observation that for the low temperature a solution is usually repeated many times before random scheme select a change. The number of such repetitions is on average significantly higher than the number of neighbours. That is why SA methods works slowly.

Experimental comparisons, depending upon neighbourhoods and tuning parameters, have been carried out for a wide class of random instances with $o = 40..3000$ and $m = 20..200$, see Nowicki and Smutnicki [232], for the generation scheme. The conclusions obtained in all tested cases are similar, hence we only illustrate them by means of a single (representative) instance. Experimental comparison, depending upon neighbourhoods, is shown in Figure 6.5. A random example having 50 parts, 300 operations, 6 machine cells with 3 machines at each cell (18 machines totally) has been run with three different neighbourhoods based on the neighbourhoods denoted as V, W, and $W+$. In the figure, there are shown the best makespans found during the first 4000 iterations of the run.

Some experiments were carried out for SA algorithm to set the influence of control parameters, the neighbourhood, initial temperature, cooling scheme and limit of iterations on the algorithm quality. For $SA+W$ and $SA+(W+)$ we set experimentally the initial (T_0) and final (T_n) temperatures 5.0 and 1.0, respectively. The limit of iterations is set as 10,000. The logarithmic scheme of

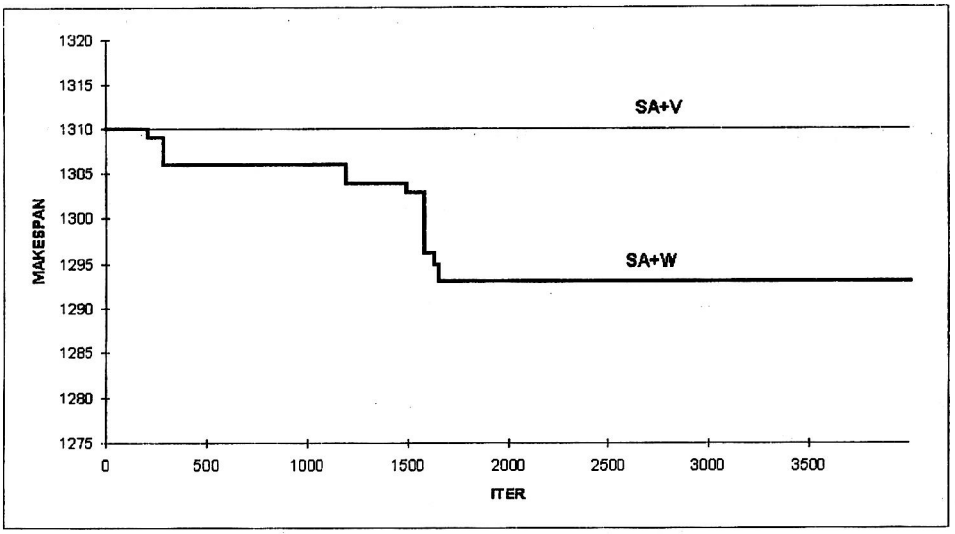

Figure 6.5: Comparison of SA method using different neighbourhoods. The best makespan found during the run (typical run for $e = 50$, $o = 300$, $m = 18$)

cooling has been applied with the parameter β found on from the dependence $\beta = (T_0 - T_n)/(10,000 T_0 T_n)$. Algorithm $SA+V$ is found to be very sensitive to proper selection of cooling scheme, which however has not been observed for $SA+W$ and $SA+(W+)$. Assuming for $SA+V$ slow reduction of the temperature, too distant solutions are accepted in the primal phase of the run, and therefore the algorithm very poorly converges to a good solution. In turn, assuming high reduction of temperature, the behaviour of $SA+V$ tends towards descending search with drift (DSD) methods, which has poor performance. In practice $SA+V$ requires the initial and final temperatures approximately m times lower comparing with these for $SA+W$ and $SA+(W+)$. Very subtle differences were found by varying the cooling scheme for $SA+W$ and $SA+(W+)$, chiefly within the first 1,000 iterations. However, no other dominant cooling scheme has been selected. It is clear that application of the neighbourhood W provides high profits.

6.5.3 Adaptive memory search algorithm

The proposed algorithm AMS uses two memory classes of the search history: a short-term memory represented by the cyclic list T of the length $LengthT$ called the *tabu list* and a long-term memory of *attractive search regions* with unvisited

neighbours of the best solutions already visited. The tabu list stores attributes of the overall processing orders visited defined by a certain pair (or two pairs) of adjacent operations at a cell. The selection of the pair(s) depends on the move performed. The list is initiated by $LengthT$ empty elements. The new element introduced to T replaces the oldest one. The move performed from a processing order π introduces to T one or two new elements (g,h) consistent with π, i.e. $(g,h) \in co\mathcal{A}(\pi)$ where $co\mathcal{A}(\pi)$ is a transitive closure of $\mathcal{A}(\pi)$. More precisely, let (j,i,y) be a performed move. Then, if $i \neq P_j$ we introduce to T the pair $(\underline{s}_j, j)$ if $\underline{s}_j \neq o$ and additionally the pair $(j, \overline{s}_j)$ if $\overline{s}_j \neq o$. If $i = P_j$, we introduce $(\underline{s}_j, j)$ if $x_j > y$ or $(j, \overline{s}_j)$ if $x_j < y$. The move $v = (j,i,y)$ from processing order π has tabu status (cannot be performed) if there exists on the list T a pair (g,h) which is *consistent* with π_v but not with π, i.e.

$$v \in W(\pi) \text{ has tabu status} \Leftrightarrow \exists (g,h) \in T : (g,h)) \notin co\mathcal{A}(\pi) \wedge (g,h) \in co\mathcal{A}(\pi_v). \tag{6.12}$$

This definition is equivalent to the following more practical condition. The move $v = (j,i,y)$, $i \neq P_j$, has the tabu status if at least one pair $(j, \pi_i(k))$, $y \leq k \leq n_i$, or $(\pi_i(k), j)$, $1 \leq k < y$, belongs to T. For $i = P_j$ this move has the tabu status if at least one pair $(\pi_i(k), j)$, $x_j < k \leq y$, if $x_j < y$, or at least one pair $(j, \pi_i(k))$, $y \leq k < x_j$, if $x_j > y$, belongs to T.

Example 6.21. Assuming $T = \emptyset$, the tabu list for the Instance and the processing order π from Example 6.2 is formed as follows. Consider move $v = (13,1,2)$ which deletes operation 13 (located in $\pi_2 = (10,13,16,19)$ in position 2) and inserts it in position 2 in $\pi_1 = (1,4,7)$. Denoting $\beta = \pi_v$, we have $\beta_1 = (1,13,4,7)$, $\beta_2 = (10,16,19)$ and $\beta_i = \pi_i$ for the remaining i. Performing this move we add to T pairs $(10,13)$ and $(13,16)$. Assuming that the first block in β contains operation 13, the following moves from β are forbidden: (i) $(13,2,y)$, $y = 1$, since among pairs $(13, \beta_2(k))$, $1 = y \leq k \leq n_2^{\beta} = 3$, there exists pair $(13,16) \in T$, (ii) $(13,2,y)$, $2 \leq y \leq 4$, since among pairs $(\beta_2(k), 13)$, $1 \leq k < y$ there exists pair $(10,13) \in T$. Therefore operation 13 cannot be inserted in any position in β_2. $\diamond$

Given a job processing order π, its neighbourhood is searched in a specific way based on the *local diversification strategy* with *move filtration*. First, the move set $\mathcal{W}(\pi)$ is split into individual subsets $\mathcal{W}_{ji}(\pi)$, each of them associated with a particular job and machine, $i \in \mathcal{M}_{c_j}$, $j \in \mathcal{U}$. Each such a subset is then searched separately, without checking the tabu status, to find the best move in terms of the makespan value. The moves filtered in this way create the basic move set $\hat{W}(\pi)$ considered for this π. The filtration is to some extent similar to a class of

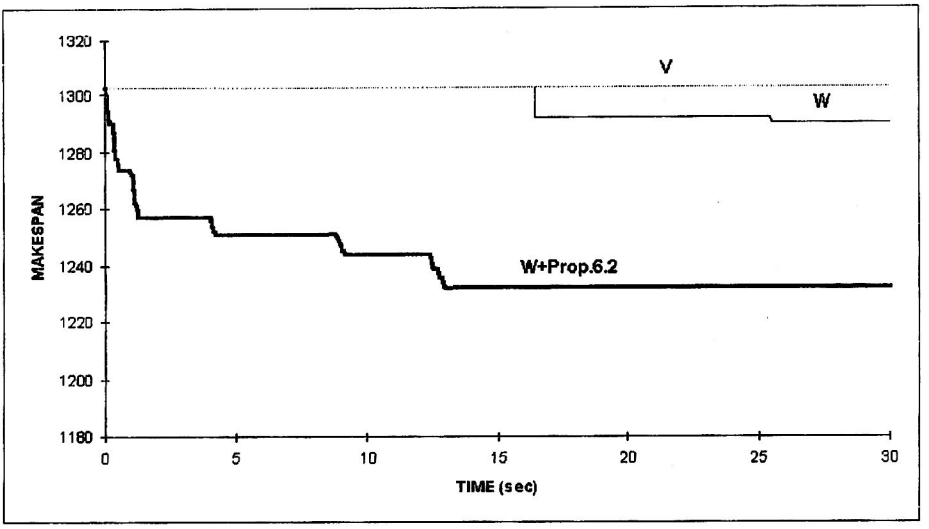

Figure 6.6: Comparison of AMS method using different neighbourhoods. The best makespan found during the run (typical run for $e = 50$, $o = 300$, $m = 18$)

strongly reduced neighbourhoods from Section 3.1 [7].

The long-term memory is associated with the searching trajectory. Starting from the initial solution π^0 we perform successive iterations, generating the sequence of solutions $\pi^0, \pi^1, \ldots$. At the i-th iteration ($i = 0, 1, \ldots$) for the given solution π^i we search the neighbourhood defined by the move set $\hat{\mathcal{W}}(\pi^i)$. *Attractive search regions* with their selected attributes (e.g. the processing order, move set) are stored in the long-term memory and used to guide the search in many diverse directions of the solution space. The details of this technique called *back jumps on the search trajectory* are given by Nowicki and Smutnicki [230, 231, 232].

Experimental comparison of neighbourhoods is shown in Figure 6.6. A random example having 50 parts, 300 operations, 6 machine cells with 3 machines at each cell (18 machines totally) has been run with three different neighbourhoods based on the move sets $\mathcal{V}(\pi)$, $\mathcal{W}(\pi)$, and $\mathcal{W}(\pi)$ with the application of Property 6.4. In the figure, there are shown the best makespans found during the first t seconds of the run. It is clear that application of Properties 6.3–6.4 provides high profits. Such results have been observed for *all instances*, and profits are much more advantageous for greater sizes of instances. The detailed behaviour

[7] The place of insertion of the part j is not random but selected in a specific way.

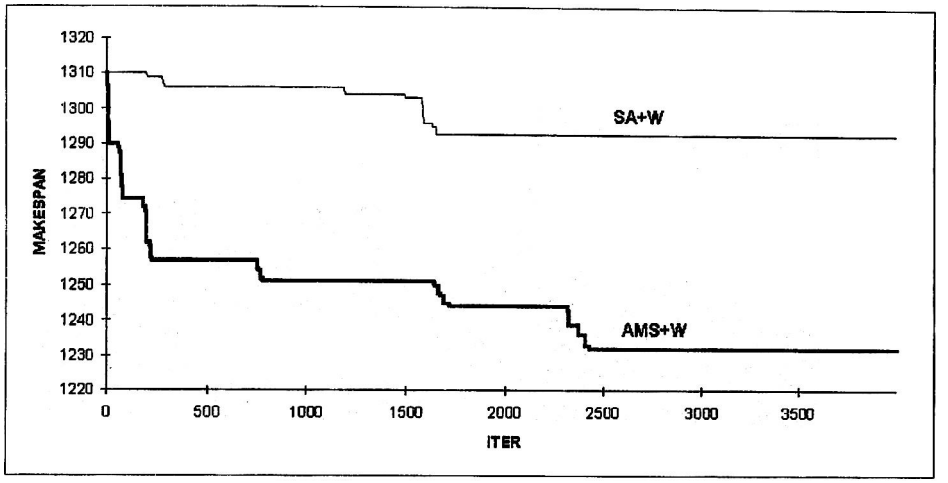

Figure 6.7: Comparison of $SA + W$ versus AMS+W in iterations. The best makespan found during the run (typical run for $e = 50$, $o = 300$, $m = 18$)

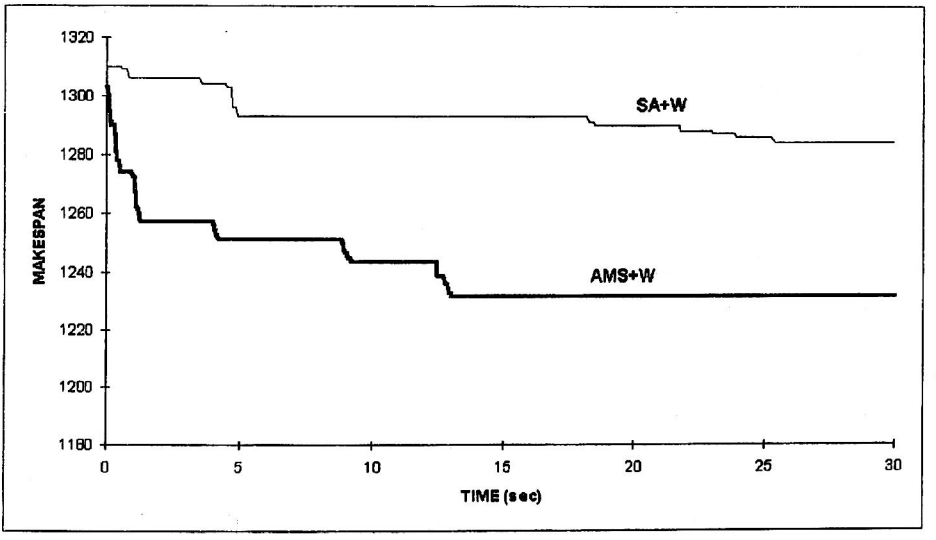

Figure 6.8: Comparison of $SA+W$ versus $AMS+W$ in time. The best makespan found during the run (typical run for $e = 50$, $o = 300$, $m = 18$)

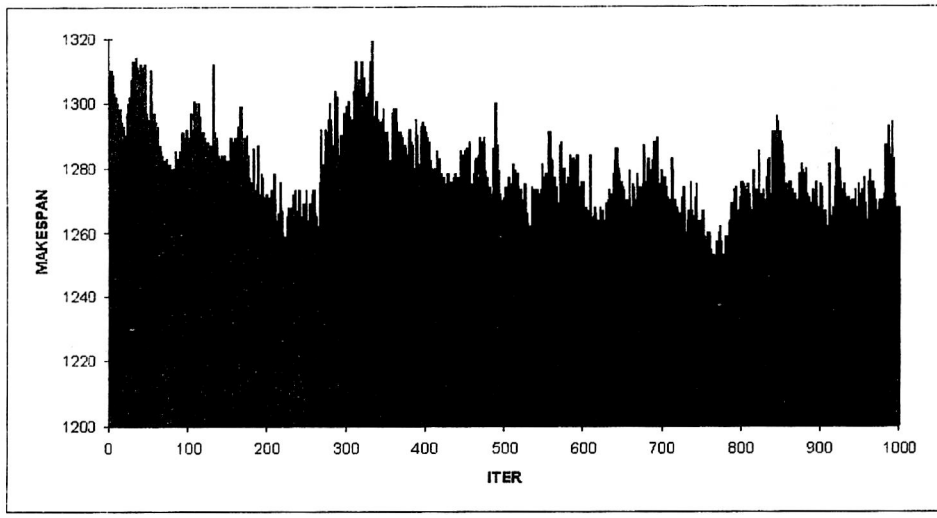

Figure 6.9: AMS trajectory (first 1,000 makespans, $C_{\max}(\pi^i)$, $i = 1, \ldots, 1000$) with the neighbourhood W (typical run for $e = 50$, $o = 300$, $m = 18$)

of AMS algorithm is shown in Figure 6.9. Makespans of visited solutions change themselves significantly, although the best up-to-now makespan quickly decreases (white line). Note that classic descending search method will stop after a few iterations in a local extremum.

Comparison of AMS with SA method shows its clear advantages, Fig. 6.9. AMS faster converges to the best solution than SA and this superiority becomes higher when the size of the instance is greater. For greater instances the cost of calculations of SA is unacceptable large.

Algorithm AMS has more intesively been examined in the paper of Nowicki and Smutnicki, [232], using 1800 random instances of various sizes ($o = 40, \ldots, 1800$, $e = 20, \ldots, 300$, $m = 4, \ldots, 60$) and various schemes of generation (fixed or random number of machines at a center, different parts, groups of similar parts). AMS algorithm provides better makespan than the best known heuristics by 0.2–11.4 within first 1,000 iterations, and by 0.2–13.7% within 10,000 iterations. The main improvement appears within the first 1,000 iterations. The relative improvement of AMS algorithm is on average 5 times greater (11 times extremally) than the relative improvement of the classic descending search method (DS) designed specially for this problem. This superiority increases if the number of centers increases.

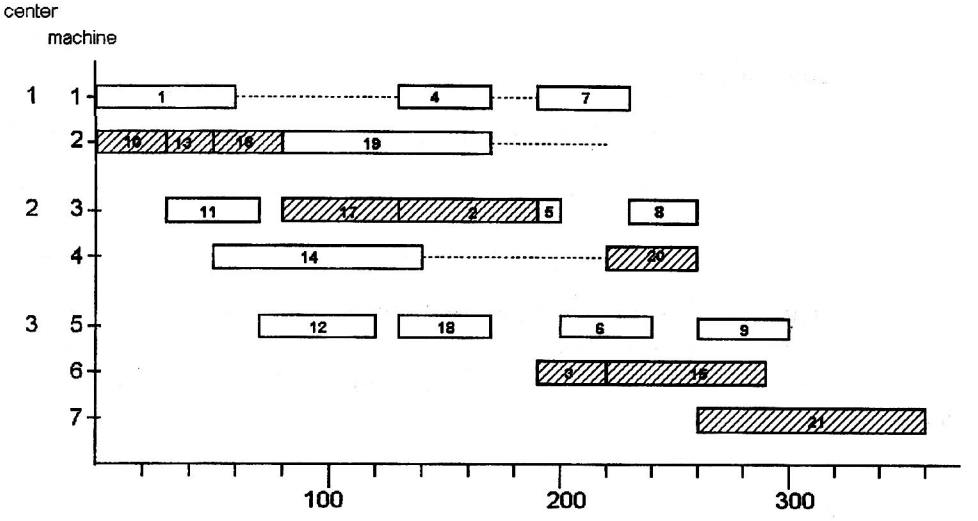

Figure 6.10: Gantt Chart for Example 6.4 (no buffers)

6.6 No buffers

In this section, we deal with the problem where no storage place for parts is allowed. This case has been considered as one of these fundamental for JIT philosophy.

The production system has mathematical formulation from Section 6.2 provided that there are no buffers between cells/machines. It means that each part having a route from some machine i to a downstream machine k can pass immediately from i to k if and only if k is free. Otherwise, it has to remain on i until k becomes available, but prevents another part from being processed on i, i.e. blocks machine i. Let C'_j denote the time in which machine P_j is released after completing operation j. A *feasible schedule* is defined by a triple (S, C', P), where $S = (S_1, \ldots, S_o)$, $C' = (C'_1, \ldots, C'_o)$, $P = (P_1, \ldots, P_o)$, such that above constraints are satisfied, when operation j starts on machine $P_j \in \mathcal{M}_{c_j}$ at time $S_j \geq 0$ and occupies this machine till the time moment C'_j.

Example 6.22. A feasible schedule for the Instance and overall processing order from Example 6.4 is shown in Gantt Chart, Fig. 6.10. For example, operation 14 of part 5 at cell 2 is performed on machine 4, is started at time 50 and completed at time 140, however it blocks machine 4 till time moment 220 since part 5 begins processing at cell 3 (operation 15) at time 220, i.e. $P_{14} = 4$,

$S_{14} = 50$, $C'_{14} = 220$. The makespan is 360. $\diamond$

For a given π the appropriate feasible schedule has to satisfy (6.1)–(6.4) and the following constraints

$$C'_{\underline{t}_j} \leq S_j, \ j \in \mathcal{O}, \ \underline{t}_j \neq o, \tag{6.13}$$

$$C'_{\underline{s}_j} \leq S_j, \ j \in \mathcal{O}; \ \underline{s}_j \neq o, \tag{6.14}$$

$$C_j \leq C'_j, \ j \in \mathcal{O}. \tag{6.15}$$

Although (6.2)–(6.3) and (6.13)–(6.15) are redundant, e.g. (6.2) can be obtained from (6.15) and (6.13), nevertheless we consciously leave them to introduce and employ some special properties of the problem. Since there are no buffers, then we have

$$C'_j = S_{\overline{t}_j} \tag{6.16}$$

if $\overline{t}_j \neq o$ and $C'_j = C_j$ otherwise. Inserting (6.16) to (6.13)–(6.15), and next eliminating some (but not all) redundant and all trivial conditions [8], one can prove that the starting times S_j have to satisfy the already known constraints (6.1)–(6.4) and an additional requirement

$$S_{\overline{t}_{\underline{s}_j}} \leq S_j, \ j \in \mathcal{O}, \ \underline{s}_j \neq o, \ \overline{t}_{\underline{s}_j} \neq o. \tag{6.17}$$

By using (6.1)–(6.4) and (6.17) we propose the following recursive formulae on starting times of operations

$$S_j = \max\{S_{\underline{t}_j} + p_{\underline{t}_j}, S_{\underline{s}_j} + p_{\underline{s}_j}, S_{\overline{t}_{\underline{s}_j}}\} \tag{6.18}$$

where $\overline{t}_o = o$, $S_o = 0 = p_o$. Similarly as (6.5), all S_j can be calculated in $O(o)$ time.

Constraints (6.1)–(6.4), (6.17) as well as starting times S have a convenient interpretation on a graph model. For this purpose we use the graph $\mathcal{G}(\pi) = (\mathcal{O}, \mathcal{A}^* \cup \mathcal{A}(\pi) \cup \mathcal{A}'(\pi))$ with the set of nodes $\mathcal{O}$ and the set of arcs $\mathcal{A}^* \cup \mathcal{A}(\pi) \cup \mathcal{A}'(\pi)$ where $\mathcal{A}^*$ and $\mathcal{A}(\pi)$ are defined by (6.6)–(6.7). The set of arcs

$$\mathcal{A}'(\pi) = \bigcup_{j \in \mathcal{O}; \underline{t}_j \neq o; \overline{s}_{\underline{t}_j} \neq o} \{(j, \overline{s}_{\underline{t}_j})\} \tag{6.19}$$

represents buffering constraints (6.17). The weight of the arc $(j, \overline{s}_{\underline{t}_j}) \in \mathcal{A}'(\pi)$ is minus p_j. Notions S_j, C_j and $C_{\max}(\pi)$ have the same meaning as in Section

[8] The aim is to leave basic constraints (6.1)–(6.4) and support them by a minimal set of conditions necessary to get a feasible solution

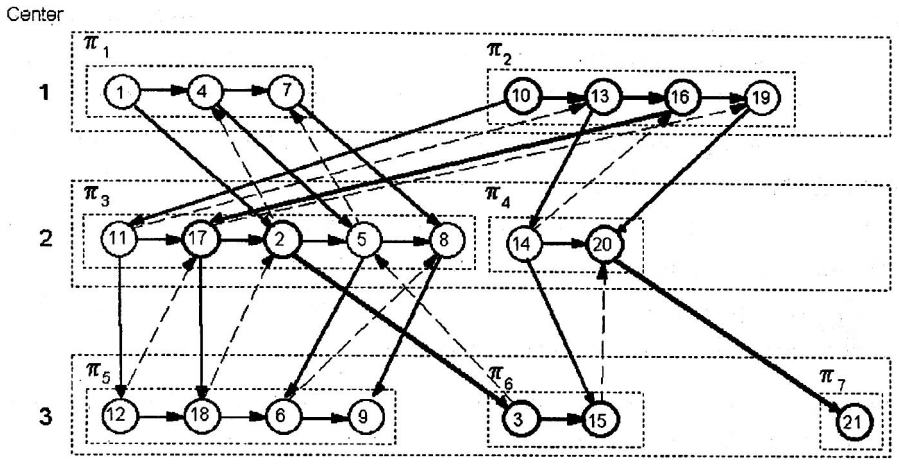

Figure 6.11: Graph $\mathcal{G}(\pi)$ for Example 6.4 (no buffers)

6.2. Notion C'_j does not have immediate interpretation in this graph, its mediate meaning follows from (6.16). Comparing this graph with that from Section 6.2 one can see it as an extension obtained by introducing buffering arcs $\mathcal{A}'(\pi)$.

One can consider this model as rather inconvenient since it contains arcs with negative weights, "artificial" in some sense. Indeed, due to the special structure of $\mathcal{G}(\pi)$, each arc $(i,j) \in \mathcal{A}'(\pi)$ can be replaced by a set of zero-weighted arcs $\{(k,j) : (k,i) \in \mathcal{A}^* \cup \mathcal{A}(\pi)\}$. However, such a modification generally increases the number of arcs in the outcome graph and, moreover, it complicates the properties formulated next. Therefore this possibility has consciously been overlooked.

The relation between processing order π, graph $\mathcal{G}(\pi)$ and feasible schedule is united in the following property.

Property 6.5. For each π such that $\mathcal{G}(\pi)$ has no cycle, there exists a feasible schedule (S, C', P) of the problem without buffers, such that S is determined by (6.18), C' by (6.16), $P_{\pi_i(j)} = i$, $j = 1, \ldots, n_i$, $i \in \mathcal{M}$, and S, C' are as small as possible. □

Proof. If $\mathcal{G}(\pi)$ has a cycle, it must be of a positive length due to special structure of weights assigned to nodes and arcs. Thus, for operations (nodes) from this cycle we cannot find any feasible times of "starting" events which might satisfy all precedence constraints. Hence, no feasible schedule (S, C', P) can be obtained in this case.

Assume that $\mathcal{G}(\pi)$ has no cycle. We will show that for the found schedule (S, C', P) all events appear in correct order. First note that buffering constraints (6.16) are satisfied. Completion times C have to satisfy clear condition (6.4). From (6.18) we immediately obtain (6.1)–(6.3). Since $S_j = S_{\bar{t}_{\underline{t}_j}}$ then from (6.16) we obtain $S_j = C'_{\underline{t}_j}$ that provides (6.13). Next, applying (6.18) and (6.16) we have $S_j \geq S_{\bar{t}_{\underline{s}_j}} = C'_{\underline{s}_j}$ for $\underline{s}_j \neq o$ and $\bar{t}_{\underline{s}_j} \neq o$. For $\underline{s}_j \neq o$ and $\bar{t}_{\underline{s}_j} = o$, we have $S_j \geq C_{\underline{s}_j} = C'_{\underline{s}_j}$. These last two results provide (6.14). Finally, applying (6.18) and (6.16) we have for $\underline{t}_j \neq o$ the inequalities $S_j \geq C_{\underline{t}_j}$ and $S_j = C'_{\underline{t}_j}$, so $C'_{\underline{t}_j} \geq C_{\underline{t}_j}$, that provides (6.15). Easy proof necessary for showing that S, C' are as small as possible we leave to the reader. ∎

Note that the feasible schedule for a given π is determined fundamentally by the single formulae of S, whereas C' has a secondary character since it follows from S. Because the starting times S_j found by (6.18) are as short as possible, $C_{\max}(\pi) = \max_{j \in \mathcal{O}} C_j$ is the minimal makespan that can be obtained for this π. Next, we will consider only *feasible overall processing orders* $\pi \in \Pi^* = \{\alpha \in \Pi : \mathcal{G}(\alpha) \text{ has no cycle } \}$ [9].

Let us define the critical path U and blocks B_{ab} in $\mathcal{G}(\pi)$ in the same way as in Section 6.2. While the existence of multiple critical paths in the problem from Section 6.2 is of little importance (this case rarely exists), now it becomes significant for the distribution of blocks. A path from a node $\underline{s}_j$ to j can pass either by the single arc $(\underline{s}_j, j) \in \mathcal{A}(\pi)$ or by two arcs $(\underline{s}_j, \bar{t}_{\underline{s}_j}) \in \mathcal{A}^*$ and $(\bar{t}_{\underline{s}_j}, j) \in \mathcal{A}'(\pi)$ in this order. In both cases, the lengths of paths are the same, however the distributions of blocks are extremely different. To employ the block elimination properties we realise the policy of "a small number of long blocks", hence the former path will be preferred. Unfortunately, the block notion does not suffice to make an solution algorithm good enough, since Property 6.2, fundamental to the use of move set $\mathcal{W}(\pi)$, does not hold in this case. Nevertheless, by introducing some new notions we can restore all advantageous properties. The *extended critical sequence* $U^* = (u_1^*, \ldots, u_z^*)$ is a sequence obtained from U by inserting an additional element $\underline{s}_{u_k}$ between any successive operations u_{k-1} and u_k such that $(u_{k-1}, u_k) \in \mathcal{A}'(\pi)$. Generally, U^* need not be any path in $\mathcal{G}(\pi)$. Next, we define an *extended block* B_{ab}^* as a maximal subsequence of the successive operations $(u_a^*, \ldots, u_b^*)$ from U^* such that $P_{u_a^*} = \ldots = P_{u_b^*}$.For the convenience of notations we set $\mathcal{U}^* = \{u_1^*, \ldots, u_z^*\}$, $\mathcal{B}_{ab}^* = \{u_a^*, \ldots, u_b^*\}$.

Example 6.23. The graph $\mathcal{G}(\pi)$ for the Example 6.4 is given in Figure 6.11. The critical path is $U = (10, 13, 16, 17, 2, 3, 15, 20, 21)$, $w = 9$, and the extended

[9] The problem of cycle does not exist in the system with infinite capacity buffers.

critical sequence is $U^* = (10,13,16,17,2,3,15,14,20,21)$, $z = 10$. There are five extended blocks in this sequence, i.e. $B^*_{1,3} = (10,13,16)$, $B^*_{4,5} = (17,2)$, $B^*_{6,7} = (3,15)$, $B^*_{8,9} = (14,20)$, $B^*_{10,10} = (21)$. $\diamond$

The main idea of the neighbourhood size reduction presented in Section 6.5.1 consists in eliminating some moves from $\mathcal{V}(\pi)$ for which it is known a priori (without computing the makespan) that they will not immediately improve $C_{\max}(\pi)$. The move v such that there exists in $\mathcal{G}(\pi_v)$ a path containing all nodes from the critical path in $\mathcal{G}(\pi)$ (i.e. nodes from $\mathcal{U}$) has the required property and thus can be eliminated. Instead of an explicit but time-consuming search of all these moves, we propose several simple and fast conditions for identifying most of them. A move can be eliminated if at least one of the following conditions is satisfied:

(1) j does not belong to $\mathcal{U}^*$,

(2) j belongs to extended block B^*_{ab} containing only single element, i.e. $a = b$.

All further cases refer to the situation where $j \in B^*_{ab}$ and $i = P_j$, i.e. move is performed inside the permutation π_{P_j}:

(3) move is made inside the extended block, i.e. $j \in B^*_{ab} \setminus \{u^*_a, u^*_b\}$ and $x_{u^*_a} < y < x_{u^*_b}$,

(4′) first element in extended block is moved to the left, i.e. $j = u^*_a$, $y < x_{u^*_a}$,

(4″) last element in extended block is moved to the right, i.e. $j = u^*_b$, $y > x_{u^*_b}$.

(5′) first element in the first extended block is moved to the right inside this block, i.e. $a = 1$, $j = u^*_a$, $y < x_{u^*_a}$.

(5″) last element in the last extended block is moved to the left inside this block, i.e. $b = w$, $j = u^*_b$, $y > x_{u^*_b}$.

These cases have been justified for the classic block notion [232], however they need an explanation for the extended blocks. Observe that $\mathcal{U} \subseteq \mathcal{U}^*$ and at most u^*_a need not be in $\mathcal{U}$. Moreover it has to be $u^*_1, u^*_z \in \mathcal{U}$ and $u^*_b \in \mathcal{U}$ for any b. If $u^*_a \in \mathcal{U}$, the extended block B^*_{ab} has the same properties as the classic block. Hence cases (1), (2), (5′), (5″) are clear since they must refer to conventional blocks, as well as cases (3), (4′), (4″) if only $u^*_a \in \mathcal{U}$. Let us consider $u^*_a \notin \mathcal{U}$. Any alteration of operations inside the extended block (i.e. operations from $B_{ab} \setminus \{u^*_a, u^*_b\}$) preserves all nodes from $\mathcal{U}$ within a path in the outcoming graph. Similar results can be obtained in cases (4′) and (4″).

Therefore, at the end, we propose the following set of moves for creating the reduced neighbourhood in the local search methods

$$\mathcal{W}^*(\pi) = \bigcup_{j \in \mathcal{U}^*} \bigcup_{i \in \mathcal{M}_{c_j}} \mathcal{W}^*_{ji}(\pi). \tag{6.20}$$

Sets $\mathcal{W}^*_{ji}(\pi) \subseteq \mathcal{V}_{ji}(\pi)$ are defined only for $j \in \mathcal{B}^*_{ab} \subseteq \mathcal{U}^*$ in exactly the same way as $\mathcal{W}_{ji}(\pi)$, see Section 6.2, however with respect to extended block B^*_{ab}.

Example 6.24. The move set $\mathcal{W}^*(\pi)$ for the Instance and processing order from Example 6.4 (no buffers) is shown in Table 6.4. It contains 34 moves marked by black bullets. Moves marked by time symbol are unfeasible. ◇

Let us analyse theoretical properties of the proposed neighbourhood $\{\pi_v : v \in \mathcal{W}^*(\pi)\}$. The proposed reduction of the neighbourhood size strongly depends on the distribution of extended blocks. Although the detailed analysis has been made, final conclusions, assuming near uniform distribution, are similar to those from Property 6.1. So the reduction remains significant. It is still advantageous since the skipped part of the neighbourhood is "less interesting" if we take account of the immediate improvements.

Property 6.6. For any overall processing order π_v, $v \in \mathcal{V}(\pi) \setminus \mathcal{W}^*(\pi)$, we have either $\pi_v \notin \Pi^*$ or $C_{\max}(\pi_v) \geq C_{\max}(\pi)$. □

The proof of this fact can be obtained by analysing the paths in the graph $\mathcal{G}(\pi)$. Since the methodology of producing such proofs has been known, Grabowski et al. [111], therefore it will be omitted.

One can expect that the remaining fundamental properties also hold. Unfortunately, even a simple instance with two parts, four operations, two machines, two cells and having the typical structure of the non-permutation flow-shop without buffers shows the lack of connectivity property. Moreover, this instance clearly shows that the optimal solution cannot be reached without passing the search trajectory through the unfeasible region $\Pi \setminus \Pi^*$. This significantly complicates the form of further theoretical properties. Let us discuss this subject in detail.

We assume that the search trajectory can contain both feasible and unfeasible processing orders. For the unfeasible processing orders $\pi \in \Pi \setminus \Pi^*$ we set the makespan $C_{\max}(\pi) = \infty$ and the move set

$$\mathcal{Z}(\pi) = \bigcup_{j \in \mathcal{O}} \{(j, P_j, x_j - 1), (j, P_j - 1, 1), (j, P_j + 1, 1)\} \tag{6.21}$$

where the move $(j, P_j, 0)$ does not exist and the moves $(j, i, 1)$ does not exist either if $i \notin \mathcal{M}_{c_j}$. Observe, this move set is a generalisation of that minimal with

Table 6.4: Sets $V(\pi)$ and $W^*(\pi)$ of moves $v = (j, i, y)$ for Example 6.4 (no buffers).

	$i=1$						$i=2$									
j	$y=1$	2	3	4	5	6	1	2	3	4	5	e_j	c_j	P_j	x_j	block
1	·	o	o				o	o	o	o	o	1	1	1	1	
4	·	·	o				o	o	o	o	o	2	1	1	2	
7	o	·	·				o	o	o	o	o	3	1	1	3	
10	•	×	×	×			·	o	×	×		4	1	2	1	$B^*_{1,3}$
13	•	•	•	•			·	·	•	×		5	1	2	2	$B^*_{1,3}$
16	•	×	×	×			×	·	·	o		6	1	2	3	$B^*_{1,3}$
19	o	o	o	o			o	o	·	·		7	1	2	4	

	$i=3$						$i=4$									
j	$y=1$	2	3	4	5	6	1	2	3	4	5	e_j	c_j	P_j	x_j	block
11	·	o	o	o	o		o	o	o			4	2	3	1	
17	·	·	•	×	×		×	•	×			6	2	3	2	$B^*_{4,5}$
2	•	·	·	•	•		•	×	×			1	2	3	3	$B^*_{4,5}$
5	o	o	·	·	o		o	o	o			2	2	3	4	
8	o	o	o	·	·		o	o	o			3	2	3	5	
14	×	•	×	×	×	×	·	×				5	2	4	1	$B^*_{8,9}$
20	×	×	•	•	•	•	·	·				7	2	4	2	$B^*_{8,9}$

	$i=5$						$i=6$					$i=7$						
j	$y=1$	2	3	4	5	6	1	2	3	4	5	1	2	e_j	c_j	P_j	x_j	block
18	·	o	o	o			o	o	o			o	o	6	3	5	1	
12	·	·	o	o			o	o	o			o	o	4	3	5	2	
6	o	·	·	o			o	o	o			o	o	3	3	5	3	
9	o	o	·	·			o	o	o			o	o	2	3	5	4	
3	×	×	•	×	×		·	•				•	•	1	3	6	1	$B^*_{6,7}$
15	•	•	•	•	•		·	·				•	×	5	3	6	2	$B^*_{6,7}$
21	•	•	•	•	•		×	×	•			·		7	3	7	1	$B^*_{10,10}$

the connectivity property form Section 3.1. The connectivity property is satisfied for the *generalised move set* $\mathcal{Z}^*(\pi)$ defined as follows: (a) if $\pi \in \Pi^*$ then we set $\mathcal{Z}^*(\pi) = \{v : v \in \mathcal{W}^*(\pi),\ \pi_v \in \Pi^*\}$, (b) if $\pi \in \Pi \setminus \Pi^*$ then we set $\mathcal{Z}^*(\pi) = \mathcal{Z}(\pi)$.

Property 6.7. For any $\pi^1 \in \Pi$ there exists a finite sequence $\pi^1, \ldots, \pi^r$ such that $\pi^{i+1} \in \{\pi_v : v \in \mathcal{Z}^*(\pi^i)\}$ for $i = 1, \ldots, r-1$, and π^r is an optimal overall processing order. □

Instead of the full formal proof, we provide only its outline. To this end we refer to the scheme of the proof of Property 3.10. Consequently, we need to show that for each π^i there exists a move in $\mathcal{Z}^*(\pi^i)$ such that the distance, see the distance measure (3.27), to optimal solution π^* decreases, i.e. $\rho(\pi^i, \pi^*) > \rho(\pi^{i+1}, \pi^*)$. If only $\pi \in \Pi^*$, the required move v belongs to $\mathcal{W}^*(\pi^i)$, and $\pi^{i+1} \in \Pi^*$, then, by modifying the mentioned proof, we can obtain the necessary inequality of the distance measure. If $\pi \in \Pi^*$, the required move v is in $\mathcal{W}^*(\pi^i)$ but $\pi^{i+1} \notin \Pi^*$, we perform the required move and obtain an unfeasible

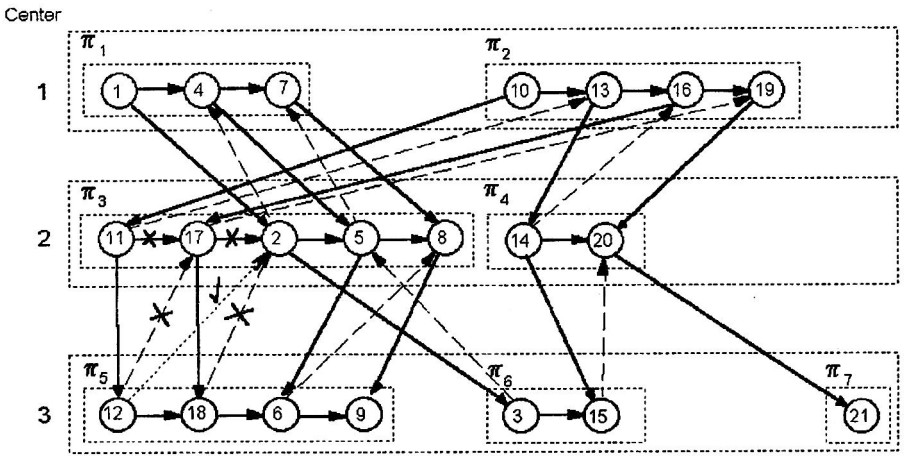

Figure 6.12: Search accelarator. Modification of graph $\mathcal{G}(\pi)$ for Example 6.4 (no buffers) and move $v = (17, 4, 2)$. Phase 1

solution π^{i+1}, but the distance to the optimal solution still decreases. Next, continuing with this unfeasible solution we perform moves however from $\mathcal{Z}(\pi)$, that allow us to reduce the distance to the optimal solution even though we need to pass through unfeasible ones. Finally, we can reach a feasible optimal solution.

This strategy of searching is relative to *strategic oscillation* technique known in AMS approach, Glover [87]. Such an interpretation of the connectivity can be included in the algorithm in an easy way. We modify the basic step of the algorithm as follows: if the current processing order is feasible, we select the move to be performed by searching in the traditional way among feasible neighbouring solutions i.e. $\{(\pi)_v : v \in \mathcal{W}^*(\pi), (\pi)_v \in \Pi^*\}$, see AMS algorithm in the previous Section for details. If no move has been selected but the set $\{(\pi)_v : v \in \mathcal{W}^*(\pi), (\pi)_v \in \Pi^*\}$ is not empty, we add the empty element to tabu and then we repeat the selection. If the set $\{(\pi)_v : v \in \mathcal{W}^*(\pi), (\pi)_v \in \Pi^*\}$ is empty, we select a move from $\mathcal{W}^*(\pi)$ to obtain the unfeasible solution by checking only tabu status but not the makespan. In turn, if the current solution is unfeasible, we select the move to be performed among those from the set $\mathcal{Z}(\pi)$ by checking the tabu status as well as the makespan.

The last subject discussed hereafter refers to the most exciting element of local search methods, i.e. to the search accelerator. Its construction for the problem stated is much more complicated than for other problems (e.g. the one

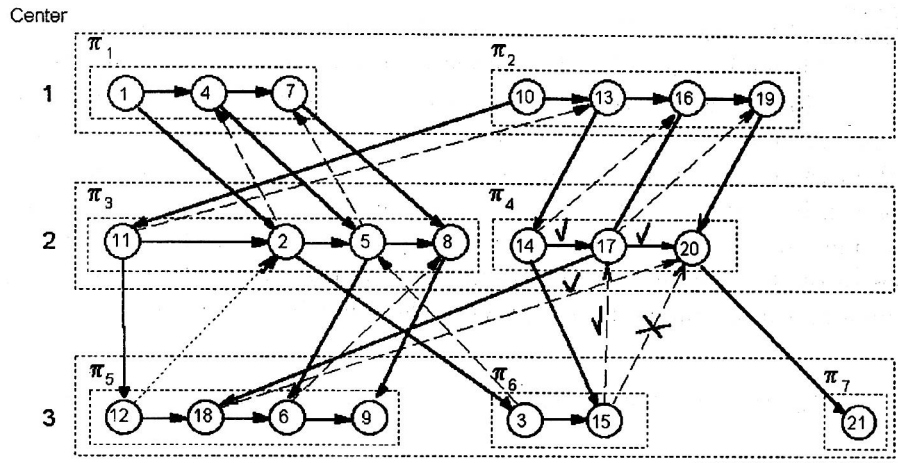

Figure 6.13: Search accelerator. Modification of graph $\mathcal{G}(\pi)$ for Example 6.4 (no buffers) and move $v = (17, 4, 2)$. Phase 2

with single cell or multiple cells with infinite capacity buffers), since a single move introduces significant changes in the graph $\mathcal{G}(\pi)$. Whereas a move changes in those problems few arcs and only those from $\mathcal{A}(\pi)$, for the problem stated the single move modifies usually 12 arcs, both from $\mathcal{A}(\pi)$ and $\mathcal{A}'(\pi)$. In the sequel, we outline the general scheme of the search accelerator for the flow-line problem without buffers. This method allows us to find the makespans for all feasible moves from the set $\bigcup_{i \in \mathcal{M}_{c_j}} \mathcal{W}^*(\pi)$ in a time $O(o)$, thus acceleratating the search $O(e)$ times.

Let $v = (j, i, y)$ be a selected feasible move. Consider graph $\mathcal{G}(\pi)$ and graph $\mathcal{G}(\pi_v)$. The former one can be obtained from the latter by a two-phase modification. First we define a *mediate graph* $\mathcal{H}_j(\pi_v)$ by removing in $\mathcal{G}(\pi)$ the operation j from P_j, but without making the insertion yet. The graph $\mathcal{H}_j(\pi)$ can be obtained from $\mathcal{G}(\pi)$ in a time $O(1)$ by removing arcs $(\underline{s}_j, j)$ and $(j, \overline{s}_j)$ from $\mathcal{A}(\pi)$, removing arcs $(\underline{t}_{\underline{s}_j}, j)$ and $(\overline{t}_j, \overline{s}_j)$ from $\mathcal{A}'(\pi)$, adding the arc $(\underline{s}_j, \overline{s}_j)$ to $\mathcal{A}(\pi)$, and adding the arc $(\underline{t}_{\underline{s}_j}, \overline{s}_j)$ to $\mathcal{A}'(\pi)$. Note, this graph is common for all moves from the set $\bigcup_{i \in \mathcal{M}_{c_j}} \mathcal{W}^*(\pi)$. Clearly, the respective arcs are removed and added if only such exist. The illustration of the phase 1 is shown in Fig. 6.12. It represents the move $v = (17, 4, 2)$ made from the processing order from Example 6.4. In the second phase, the appropriate insertion of the operation j is performed starting from $\mathcal{H}_j(\pi)$. To this end we need to remove two moves $(\pi_i(y-1), \pi_i(y)) \in \mathcal{A}(\pi)$ and

$(\bar{t}_{\pi_i(y-1)}, \pi_i(y)) \in \mathcal{A}'(\pi)$, and next to add the moves $(\pi_i(y-1), j)$ and $(j, \pi_i(y))$ to $\mathcal{A}(\pi)$, and the moves $(\bar{t}_{\pi_i(y-1)}, j)$ and $(\bar{t}_j, \pi_i(y))$ to $\mathcal{A}'(\pi)$. The illustration of phase 2 is shown in Fig. 6.13. The accelerator refers to the values R_s, Q_s, $s \in \mathcal{O}$, calculated for the graph $\mathcal{H}_j(\pi)$. The value R_s equals the length of the longest path passing into the node s (without p_s) in this graph, whereas Q_s equals the length of the longest path passing out of the node s (also without p_s). All these values can be found in a time $O(o)$. Analysing the graph $\mathcal{G}((\pi)_v)$, it is sufficient to consider two separate subcases depending on the fact whether the longest path passes through the node j or not. Hence, the makespan for the move $v = (j, i, y)$ can be found in a time $O(1)$ by applying the following formulae

$$C_{\max}((\pi)_v) = \max\{\max\{R_{\pi_i(y-1)} + p_{\pi_i(y-1)}, R_j, R_{\bar{t}_{\pi_i(y-1)}}\} + p_j$$

$$+ \max\{Q_j, p_{\pi_i(y)} + Q_{\pi_i(y)}\}, C\} \tag{6.22}$$

where C is the length of the longest path in $\mathcal{H}_j(\pi)$. To complete the accelerator, we only mention that for a given $j \in \mathcal{O}$ we can find in a time $O(o)$ all preceding nodes (i.e. there exists a path from a preceding node to j) and all succeeding nodes in $\mathcal{H}(\pi)$. Thus, we can test the feasibility of all moves from $\bigcup_{i \in \mathcal{M}_{c_j}} \mathcal{W}^*(\pi)$ in a time $O(o)$.

Property 6.8. The feasibility of all π_v, $v \in W^*(\pi)$ can be checked in a time $O(ow)$. □

Property 6.9. The makespans for all feasible π_v, $v \in W^*(\pi)$ can be found in a time $O(ow)$. □

Although, the results obtained are not so valuable as the result from Property 6.3, it still remain significant, since the single neighbourhood can be searched in a time $O(e)$, which is faster than the explicite calculation of the makespans for all neighbours.

Experimental test of AMS algorithm has shown clear advantages of the application of the reduced neighbourhood, see Figure 6.14. In this figure there are shown the best makespan values for AMS methods with the neighbourhoods $\mathcal{V}(\pi)$ and $\mathcal{Z}^*(\pi)$ (with accelerator) during the first 150 seconds for an instance with $o = 300$, $m = 18$ (typical run). In practice, unfeasible solutions in the search trajectory have been observed very rarely.

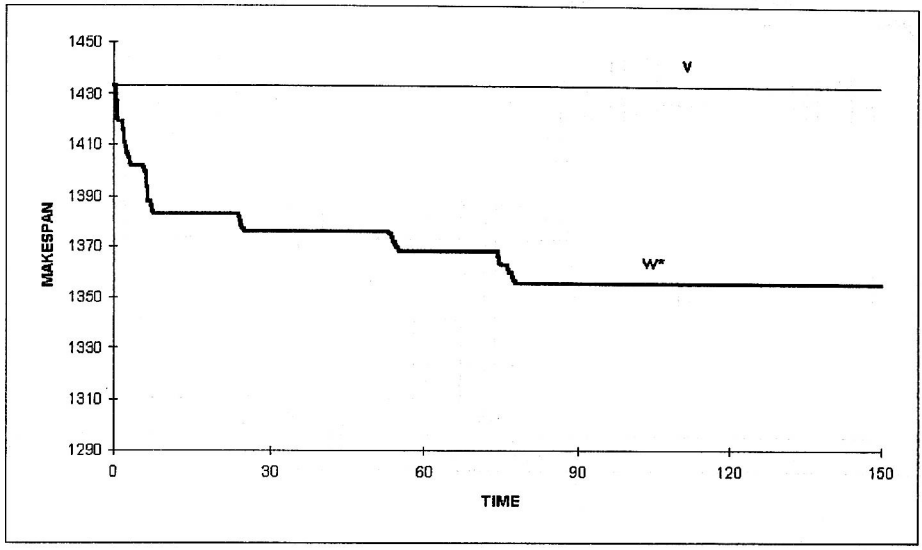

Figure 6.14: Comparison of different neighbourhoods for AMS without buffers. The best makespan found during the run (typical run for $e = 50$, $o = 300$, $m = 18$)

6.7 Machine buffers with limited capacity

To prevent superfluous machine blocking one can accept a limited storage by using buffers of small capacity. The size of individual buffers can also be employed as a tool in controlling flows of parts in JIT systems.

The production system has mathematical formulation in Section 6.2 with an additional constraint that buffer F_i is located before machine i and has capacity $f_i \geq 0$, i.e. it can store at most f_i parts at a time, Figure 6.15. It means that each part completed on some machine and sent to a machine i can be stored in the buffer F_i if it is not completely filled, otherwise this part must remain on the primal machine (blocking this machine) until will be transferred to a buffer. A *feasible schedule* is defined by four vectors (S', S, C', P) such that above constraints are satisfied, and the part e_j, before performing operation j, goes to the buffer F_{P_j} at time S'_j, is taken from the buffer at time S_j, starts processing on machine $P_j \in \mathcal{M}_{c_j}$ at time S_j and occupies this machine till time moment C'_j.

Several problems arise depending on physical (mechanical) construction of the buffer. They generate some additional constraints, e.g.: (a) a part must go on the machine only through a buffer, (b) some parts can skip buffer and go immediately on the machine, (c) each buffer has capacity one (or at most one), (d) parts arrive

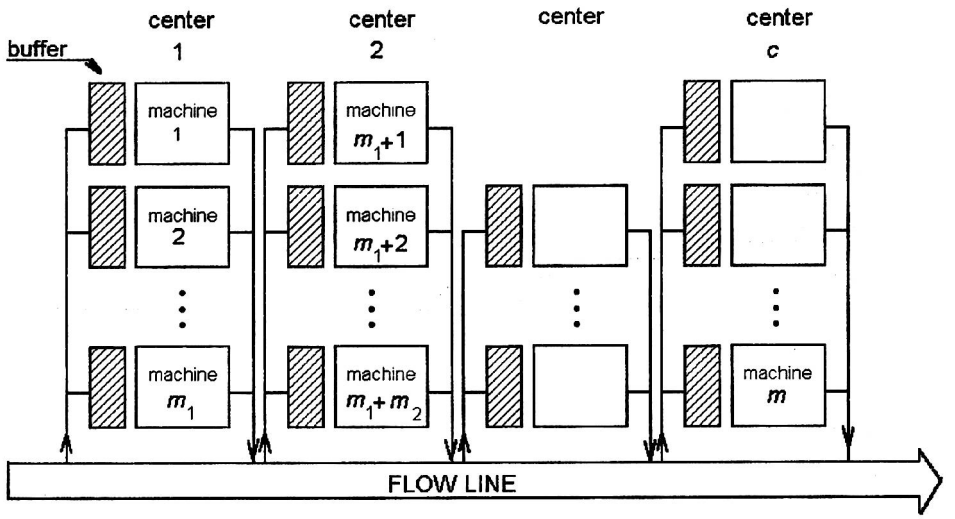

Figure 6.15: Flow line with machine buffers

at the buffer and leave this buffer in the same sequence (FIFO service rule), (e) parts can arrive at the buffer, but the sequence of arrival is different from the leaving sequence (any service rule), (f) passing a part through the buffer needs a time, usually constant for all parts. Clearly, these constraints should be applied in a reasonable way, e.g. either (a) or (b) can be applied, either (d) or (e) can be applied, (c) eliminates (d) and (e), however can also be combined with (a) or (b), etc. Note that (e) requires explicitly a sequence σ_i in which parts arrive at the buffer F_i to prevent competitions where "the winner takes the last free place in the buffer". Clearly, parts must leave buffer F_i in the sequence π_i, which is the processing order on machine i. Taking into account the notions and models that have already been introduced we can propose a few "ad hoc" approaches to solve these problems.

Assuming any of non-zero buffer capacities, FIFO service rule and zero passing time, we can replace each machine buffer F_i by a sequence of f_i fictitious machines with zero processing times for each part. Originally buffered as well as all fictitious machines aquire now the non-buffered status. (Buffer passing time can be modelled next as a time of transport between successive fictitious machines.) In spite of a simplicity of the description, this approach has some serious disadvantages. The most significant is the size of the model measured by the number of nodes and arcs in appropriate graph $\mathcal{G}(\pi)$, since it is crucial both for the time of a single neighbourhood search as well as for a cardinality of the move set. In the

proposed approach, the size of the model significantly depends on buffer capacities, namely the number of nodes and arcs in the graph $\mathcal{G}(\pi)$ is $O(\sum_{i\in\mathcal{M}}(1+f_i))$ times larger than for that from Section 6.2. A next disadvantage is associated with processing times which are assumed to be positive (non-zero). Moreover, to ensure the feasibility of an overall processing order, processing orders on all fictitious machines in a sequence leading to machine i must be conformable with π_i [10]. So, moves cannot be defined separately for fictitious machines, since their workload and processing orders follow from processing orders on real machines. This complicates the definition of a move as well as the properties which would be useful for the application of local search method, particularly in the context of an analogy of Property 6.4. A modification of the proposed approach consists in using, instead of F_i, an auxiliary buffer with capacity $f_i - k$ followed by a sequence of k fictitious machines for some k, $1 \leq k \leq f_i$.

An alternative approach is to replace the machine buffer F_i by a set of f_i parallel fictitious machines with zero processing times. This set is dedicated for processing operations from the set $\mathcal{N}_i$ only. The originally buffered machine i as well as fictitious machines after this transformation attain the non-buffered status. Although this model does not contravene the FIFO buffer service rule, and thus is recommended for buffers without fixed service rule, it has similar disadvantages as the first proposed approach.

It is well-known that the ideal model should be simple, small and convenient as much as possible. Unfortunately, cases listed at the beginning of this subsection imply a few various special small models or a single general model but of bigger size. Obviously, the complexity of models increases with the increasing complexity of problems (and constraints), but the efficiency of algorithms decreases at the same time. That is why we discuss several models. The first one, dedicated to $f_i \in \{0,1\}$, is related to that from Section 6.6 and preserves the original size of the graph. The second, recommended for any f_i and FIFO service rule, is of the size at most twice as big as that of the primal.

6.7.1 Unit machine buffers

In this section we consider the most common case $f_i \in \{0,1\}$ where no passing over of the buffer is allowed if $f_i = 1$ [11]. Thus $\sigma_i = \pi_i$, $i \in \mathcal{M}$. We start from some auxiliary notions. The *b-machine* predecessor/successor $\underline{r}_j$, $\overline{r}_j$ of the operation j is set as $\underline{r}_{\pi_i(j)} = \pi_i(j - f_i)$ for $j = f_i + 1, \ldots, n_i$, and $\underline{r}_{\pi_i(j)} = o$ for $j = 1, \ldots, f_i$; $\overline{r}_{\pi_i(j)} = \pi_i(j + f_i)$ for $j = 1, \ldots, n_i - f_i$, and $\overline{r}_{\pi_i(j)} = o$ for $j = n_i + f_i + 1, \ldots, n_i$. For the considered case we have: if $f_i = 0$ then $\underline{r}_j = j = \overline{r}_j$, if $f_j = 1$ then

[10] This is only a necessary condition, but not sufficient.

[11] If $f_i = 0$, we say that all parts pass over the buffer.

$\underline{r}_j = \underline{s}_j$, $\overline{r}_j = \overline{s}_j$. For a given π, an appropriate feasible schedule (S', C, C', P) has to satisfy (6.1)–(6.4) and (6.13)–(6.15). Invalid constraint (6.16) should to be replaced by

$$C'_j = \max\{C_j, S_{\underline{r}_{\overline{t}_j}}\} \tag{6.23}$$

where $\underline{r}_o = o$ and $S_o = 0$. New constraints on S'_j should be also included

$$S'_j \leq S_j,\ j \in \mathcal{O}, \tag{6.24}$$

$$C'_{\underline{t}_j} = S'_j,\ \underline{t}_j \neq o,\ j \in \mathcal{O}, \tag{6.25}$$

$$S_{\underline{r}_j} \leq S'_j,\ \underline{r}_j \neq o,\ j \in \mathcal{O}. \tag{6.26}$$

From (6.25)–(6.26) and (6.23) we deduce that

$$S'_j = \max\{C_{\underline{t}_j}, S_{\underline{r}_j}\} \tag{6.27}$$

where $C_o = 0$. Inserting (6.23) and (6.27) into the remaining conditions we obtain, after some simple transformations, besides (6.1)–(6.4), an additional requirement

$$S_{\underline{r}_j} \leq S_{\overline{s}_{\underline{t}_j}} \tag{6.28}$$

for $j \in \mathcal{O}$ such that $\underline{r}_j \neq o$, $\underline{t}_j \neq o$, $\overline{s}_{\underline{t}_j} \neq o$. Finally, we can propose the following recursive formulae of starting times of operations

$$S_j = \max\{S_{\underline{t}_j} + p_{\underline{t}_j}, S_{\underline{s}_j} + p_{\underline{s}_j}, S_{\underline{r}_{\overline{t}_{\underline{s}_j}}}\} \tag{6.29}$$

where $\underline{r}_o = o$, $\overline{t}_o = o$, $S_o = 0 = p_o$. Similarly to (6.5), all S_j can be calculated in $O(o)$ time.

Example 6.25. A feasible schedule for the Instance and overall processing order from Example 6.4, assuming that each machine buffer has the capacity one, is shown in Gantt Chart, Fig. 6.16. For example, operation 4 of part 2 at cell 1 is performed on machine 1, it comes to the buffer of this machine at time 0, is started at time 80, and completed at time 110. However, it blocks machine 1 till the time moment 130, since buffer to machine 3 at cell 2 (where this part has to go) is busy. Hence $P_4 = 1$, $S'_4 = 0$, $S_4 = 80$ $C_4 = 110$, $C'_4 = 130$. The makespan is 330. $\diamond$

The appropriate graph model can be formulated as follows: $\mathcal{G}(\pi) = (\mathcal{O}, \mathcal{A}^* \cup \mathcal{A}(\pi) \cup \mathcal{A}'(\pi))$ where $\mathcal{A}^*$ and $\mathcal{A}(\pi)$ are given by (6.6)–(6.7). The set $\mathcal{A}'(\pi)$ of arcs representing buffering constraints need to be re-defined and now takes the form

$$\mathcal{A}'(\pi) = \bigcup_{j \in \mathcal{O};\, \underline{r}_j \neq o;\, \underline{t}_j \neq o;\, \overline{s}_{\underline{t}_j} \neq o} \{(\underline{r}_j, \overline{s}_{\underline{t}_j})\} \tag{6.30}$$

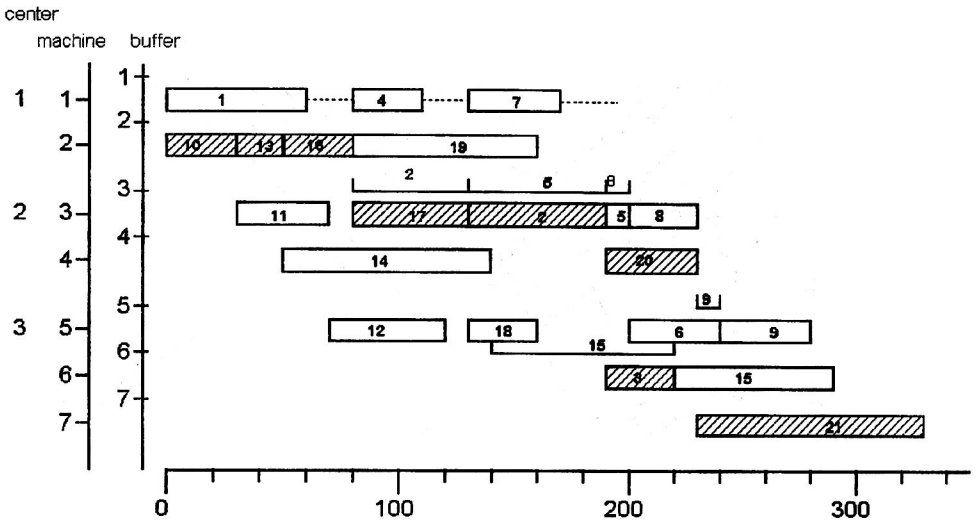

Figure 6.16: Gantt Chart for Example 6.4 (each of the machine buffers has capacity of one)

suitable for (6.28). The arc $(\underline{r}_j, \overline{s}_{\underline{t}_j}) \in \mathcal{A}'(\pi)$ has the weight minus $p_{\underline{r}_j}$. Notions S_j, C_j, and $C_{\max}(\pi)$ have the same meanings as in Section 6.2. A processing order π is united with the feasible schedule (S', S, C', P) by the following property.

Property 6.10. For each π such that $\mathcal{G}(\pi)$ has no cycle, there exists a feasible schedule (S', S, C', P) of the problem such that S is determined by (6.29), C' by (6.23), S' by (6.27), $P_{\pi_i(j)} = i$, $j = 1, \ldots, n_i$, $i \in \mathcal{M}$, and S', C, C' are as small as possible. □

Proof. If $\mathcal{G}(\pi)$ has a cycle, it must be of positive length due to special structure of weights assigned to nodes and arcs. Thus, for operations (nodes) from this cycle we cannot find any feasible times of "starting" events which might satisfy all precedence constraints. Hence, no feasible schedule (S', S, C', P) can be obtained in this case.

Assume that $\mathcal{G}(\pi)$ does not contain a cycle. We will show that events appear in correct sequences, and each buffer i contains not more than f_i parts in each time moment t. First consider events. From (6.29) we immediately get (6.1)–(6.4). Condition (6.15) follows from (6.23). Since obviously $S_{\underline{r}_j} \leq S_j$, then by using (6.2), (6.23) we have $S_j \geq \max\{C_{\underline{t}_j}, S_{\underline{r}_j}\} = C'_{\underline{t}_j}$ which provides (6.13). Inserting (6.23) to (6.27) we obtain (6.25). By using (6.25) and (6.2) we have

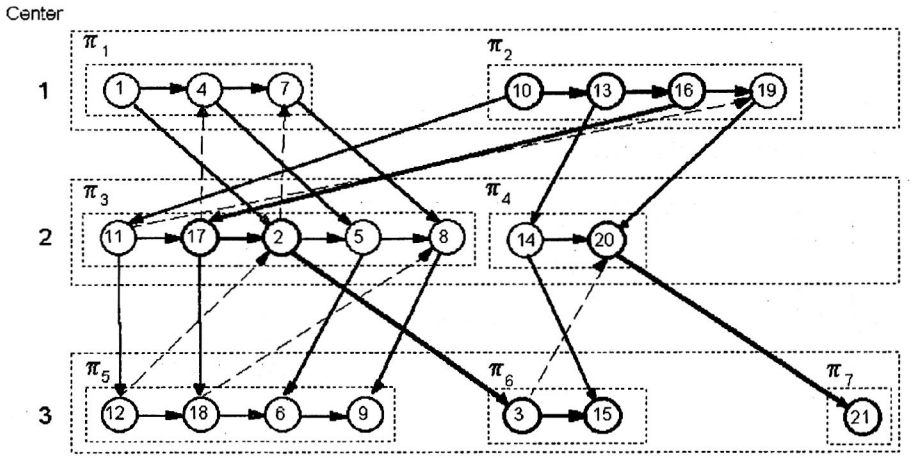

Figure 6.17: Graph $\mathcal{G}(\pi)$ for Example 6.4 (each buffer has capacity of one)

$S'_j = C'_{\underline{t}_j} \leq S_j$, which provides (6.24). Formulae (6.26) is obtained immediately from (6.27). Since $S_{\underline{r}_{\underline{t}_{\underline{s}_j}}} \leq S_j$ by (6.29) and $C_{\underline{s}_j} \leq S_j$ from (6.2), by using (6.23) we have $C'_{\underline{s}_j} = \max\{C_{\underline{s}_j}, S_{\underline{r}_{\underline{t}_{\underline{s}_j}}}\} \leq S_j$, which provides (6.14).

Next analyse the buffer capacities. We will show that for each time moment the number of parts in the buffer F_i is not larger than f_i. Since the buffer changes its state only in discrete time moments, we can check buffer overflow only in S'_j, $j \in \mathcal{O}$. Consider separately two cases. If "$f_i = 0$" then $\underline{r}_j = j$ and from (6.24)–(6.26) we have $S_j = S'_j = C_{\underline{t}_j}$. It means that part e_j between operations $\underline{t}_j$ and t is sent without using buffer F_{P_j}. If "$f_i = 1$" then $\underline{r}_j = \underline{s}_j$ and from (6.26) we have $S_{\underline{s}_j} \leq S'_j$. Hence buffer F_{P_j} contains in S'_j zero parts, so loading this buffer with a new part e_j in S_j we do not make the overflow.

Since (6.29) corresponds to the longest path in a graph, and therefore S_j, $j \in \mathcal{N}$, are as small as possible, C' and S' are as small as possible either. ∎

By virtue of Property 6.10 we conclude that $C_{\max}(\pi) = \max_{j \in \mathcal{O}} C_j$ is the minimal makespan of the schedule (S', S, C', P) obtained from a given π.

Let us define the critical path U and blocks B_{ab} in $\mathcal{G}(\pi)$ in the same way as in Section 6.2. Unfortunately, neither block nor extended block notions are sufficient for introducing the advantageous reduction of the move set, see Properties 6.2 and 6.5. To obtain an analog of these properties we have to introduce some new notions. First, for U we calculate the extended critical sequence U^* and extended

blocks B^*_{ab} in the same way as in Section 6.6. For each extended block B^*_{ab}, we define an *anti-block* B^-_b as follows: $B^-_b = ()$ (empty) if $b = z$ or $(u^*_b, u^*_{b+1}) \in \mathcal{A}^*$, and $B^-_b = (j_1, \ldots, j_{f_i+1})$ such that $j_1 = u^*_b$, $j_{k+1} = \overline{s}_{j_k}$ for $k = 1, \ldots, f_i$, $i = P_{j_1}$, otherwise. By the definition of $\mathcal{A}'(\pi)$ such a sequence always exists and $j_{f_i+1} = \overline{r}_{j_1}$. Denote by $\mathcal{B}^-_b$ the set of elements from the sequence B^-_b, and let $\mathcal{U}^- = \bigcup_{\mathcal{B}^*_{ab} \subseteq \mathcal{U}^*} \mathcal{B}^-_b$.

Example 6.26. The graph $\mathcal{G}(\pi)$ for the Example 6.4 (each of the machine buffer has capacity of one) is given in Figure 6.17. The critical path is U=(10,13,16,17,12,3, 20,21), $w = 8$, and the extended critical sequence is U^*=(10,13,16,17,12,3,14, 20,21), $z = 9$. There are five extended blocks in this sequence $B^*_{1,3} = (10, 13, 16)$, $B^*_{4,5} = (17, 12)$, $B^*_{6,6} = (3)$, $B^*_{7,8} = (14, 20)$, $B^*_{9,9} = (21)$. Antiblocks B^-_3, B^-_5, B^-_8, B^-_9 are empty, whereas $B^-_6 = (3, 15)$. $\diamond$

We propose the following set of moves for tabu search method

$$\mathcal{W}^+(\pi) = \bigcup_{j \in \mathcal{U}^* \cup \mathcal{U}^-} \bigcup_{i \in \mathcal{M}_{c_j}} \mathcal{W}^+_{ji}(\pi) \tag{6.31}$$

Sets $\mathcal{W}^+_{ji}(\pi) \subseteq \mathcal{V}_{ji}(\pi)$ are defined, depending on extended blocks and anti-blocks, only for $j \in \mathcal{B}^*_{ab} \cup \mathcal{B}^-_b \subseteq \mathcal{U}^* \cup \mathcal{U}^-$. If $j \in B^*_{ab}$ such that $|\mathcal{B}^-_b| \leq 1$ we define $\mathcal{W}^+_{ji}(\pi)$ in the way in exactly the same as $\mathcal{W}_{ji}(\pi)$, see Section 6.2, however with respect to extended block B^*_{ab}. If $j \in B^*_{ab} \cup \mathcal{B}^-_b$, $|\mathcal{B}^-_b| > 1$, and $i \neq P_j$, we set $\mathcal{W}^+_{ji}(\pi) = \mathcal{V}_{ji}(\pi)$. In remaining cases, i.e. if $i = P_j$, $a \neq b$, $|\mathcal{B}^-_b| > 1$ we set: $\mathcal{W}^+_{ji}(\pi) = \mathcal{V}_{ji}(\pi)$ if $j = u^*_b$; $\mathcal{W}^+_{ji}(\pi) = \mathcal{V}_{ji}(\pi) \setminus \mathcal{X}_j(x_{u^*_a} + 1, x_{u^*_b} - 1)$ if $j \in \mathcal{B}^*_{ab} \setminus \{u^*_a, u^*_b\}$; $\mathcal{W}^+_{ji}(\pi) = \mathcal{V}_{ji}(\pi) \setminus \mathcal{X}_j(x_{u^*_b} + 1, x_{\overline{r}_{u^*_b}} - 1)$ if $j \in \mathcal{B}^-_b \setminus \{u^*_b, \overline{r}_{u^*_b}\}$; $\mathcal{W}^+_{ji}(\pi) = \mathcal{V}_{ji}(\pi) \setminus \mathcal{X}_j(1, x_j)$ if $j = u^*_a$; $\mathcal{W}^+_{ji}(\pi) = \mathcal{V}_{ji}(\pi) \setminus \mathcal{X}_j(x_j, n_i)$ if $j = \overline{r}_{u^*_b}$.

Among few theoretical properties of the proposed neighbourhood $\{\pi_v : v \in \mathcal{W}^+(\pi)\}$ we deal only with the most important. The reduction of the neighbourhood size is advantageous since the skipped part is "less interesting" if we take account of immediate improvements.

Property 6.11. For any overall processing order π_v, $v \in \mathcal{V}(\pi) \setminus \mathcal{W}^+(\pi)$, we have either $\pi_v \notin \Pi^*$ or $C_{\max}(\pi_v) \geq C_{\max}(\pi)$. $\square$

6.8 Comments and conclusions

The proposed approach can be extended to more general cases which cover most of the practical applications in flexible flow lines scheduling problems and also in scheduling parts in a flexible cell production (work cells). The proposed model

has significantly small dimension than other models which can be proposed for the problem stated. Moreover, it can be modified without essential complication to include at least the following constraints: there is a time of job transport between cells, jobs arrive in various time moments, machines are available after their ready times, jobs must be finished before the given due date, jobs must be finished just in time, etc. Also the non-uniform machines as well as the machine set-up times can be included, see Section 3.3 for details, with all mentioned there consequences.

The remaining components of the proposed algorithm (not discussed here) are presented in detail in [232] and have been selected from among many alternative constructions that were tested and examined by various researches papers [230, 231, 232]. Excellent computational results obtained for all problems from cited papers can be obtained also in all considered cases.

Further research should be carried out to examine the remaining types of buffers already mentioned. Although some works have been devoted to this subject, Smutnicki [307], the proposed models should be improved, particulary for the case of machine buffers greater than two.

Chapter 7

Final conclusions

7.1 Industrial applications

Many industrial applications of the considered models have been given. We mention only a few of them, reported in the papers already published:

chemical industry: problems of scheduling serial multi-product batch plants are modelled by means of classic flow shop model, flow-shop with NS, ZW, or LS (limited storage) policies, mixed NS/LS/ZW policy, cleaning times (set-up times), transfer times, shared buffer, etc, or appropriate models with parallel units per each stage [23, 53, 118, 155, 170, 177, 189, 243, 252, 265, 318, 330, 331],

building industry: the basic problem emerges from a factory in Poland and uses the flow-shop model with mixed NS/ZW policy [114], a more advanced model incorporates also non-bottleneck machines,

printed circuit assembly: the basic model refers to the flow-shop with set-up times [162] applied in a factory in Korea, more advanced model refers to flexible flow line with NS policy in an IBM division [337],

computer systems, telecommunication networks: a list of applications of flexible flow-line models is enumerated in [28, 149],

spaceship processing: this is the most incredible application of the flexible flow-line model with NS policy found in the Kennedy Space Center [149],

polymer, chemical and petrochemical industries: see applications in chemical branches and also [55],

Further applications one can find in packing industry, transport, FMS, see also [149].

7.2 Future research directions

The results presented in this thesis do not exhaust neither theoretical problems arising in JIT systems nor appropriate solution methods. A rich variety of production structures has been observed in practice. As to the subjects, there are at least the following research directions with the immediate applications in manufacturing processes:

(a) NS inventory control policy in a flexible job-shop (FJS) and flexible network (FN),

(b) ZW inventory control policy for different production structures (FFS, FJS, FN),

(c) comparison between NS and ZW for different production structures,

(d) the influence of set-up times and transport times on inventory policy,

(e) minimal WIP policy (with the best delivery performance based on due dates) linked with NS inventory control for different production structures,

(f) minimal cycle time with NS and ZW inventory control,

(g) buffer (store) size selection,

(h) incorporation of more complex buffer structures, e.g. the shared buffer, dedicated part buffers, input/output buffers, buffering performed by transport system,

(i) negotiation of delivery dates.

All these problems can be analysed by using tools and approaches outlined in this thesis.

7.3 Epilogue

The models and methods known from the deterministic scheduling theory can be applied to analyse some selected problems of optimisation and control in JIT manufacturing. Simultaneously, JIT philosophy creates demands, addressed to ST, for new, advanced models of more complex manufacturing structures and efficient solution methods. Several types of local search methods and their application to automated and conventional JIT manufacturing have been considered and examined. Most of them belong the very new generation of fast methods

with very high accuracy, created on the base of SA or TS approach with the reduced neighbourhood and search accelerator. These methods express several important features: they ensure (in a range unknown up to now) a balanced compromise between the solution quality and the computational effort necessary to find a good solution, they allow us to solve industrial instances of a high size in a reasonable time even on a PC, can easily be adjusted to parallel computing. This is a step towards better algorithms of automated manufacturing control as well as semi-learned optimisation algorithms of new generation.

Bibliography

[1] Aarts E.H.L., van Laarhoven P.J.M., Simulated annealing: a pedestrain review of the theory and some applications. In: Pattern Recognition and Applications, Eds. Deviijver P.A. and Kittler J., Springer, 1987, Berlin.

[2] Achuthan N.R., Grabowski J., Sidney J.B., Optimal flow-shop scheduling with earliness and tardiness penalties, *Opsearch*, 1981, 18, 117–138.

[3] Adams, J., Balas E., Zawack D., The Shifting Bottleneck Procedure for Job Shop Scheduling, *Management Science*, 1988, 34, 391–401.

[4] Adamopolous G.I., Pappis C.P, Scheduling jobs with different, job-dependent earliness and tardiness penalties using SLK methos, *European Journal of Operational Research*, 1996, 88, 336–344.

[5] Aderohunmu R., Mobolurin A., Bryson N., Joint Vendor Buyer Policy in Jit Manufacturing, *Journal of Operational Research Society*, 1995, 46, 375–385.

[6] Agrawal A., Harhalakis G., Minis I., Nagi R., Just-in-Time Production of Large Assemblies, *IIE Transactions*, 1996, 28, 653–667.

[7] Al-Turki U.M., Mittenthal J., Raghavachari M., A dominant subset of V-shaped sequences for a class of single-machine sequencing problems, *European Journal of Operational Research*, 1996, 88, 345–347.

[8] Amin S., Using Adaptive Temperature Control for Solving Optimisation Problems, *Baltzer Journals*, 1996, (in print)

[9] Ansari A., Heckle J., JIT purchasing: Impact of freight and inventory costs, *Journal of Purchasing Matl. Management*, 1987, 23, 24–28.

[10] Applegate D., Cook W., A Computational Study of the Job-Shop Scheduling Problem. *ORSA Journal on Computing*, 1991, 3, 149–156.

[11] Ashayeri J., Gelders L.F., Warehouse design optimization, *European Journal of Operational Research*, 1985, 21, 285–294.

[12] Askin R.G., Mitwasi M.G., Goldberg J.B., Determining the Number of Kanbans in Multiitem Just-in-Time Systems, *IIE Transactions*, 1993, 25, 89–98.

[13] Bagchi U., Sullivan R., Chang Y., Minimizing Mean Squared Deviation of Completion Times About a Common Due Date, *Management Science*, 1987, 33, 894–906.

[14] Baker K.R., Introduction to Sequencing and Scheduling, Wiley, New York, 1974.

[15] Baker K.R., A dynamic priority rule for scheduling against due dates, TIMES/ORSA Conference, Houston, 1981.

[16] Baker K.R., Martin J.B., An experimental comparison of solution algorithms for the single-machine tardiness problem, *Naval Research Logistics Quartely*, 1974, 21, 187–199.

[17] Baker K.R., Su Z.S., Sequencing with due dates and early start times to minimize tardiness, *Naval Research Logistics Quartely*, 1974, 21, 171–176.

[18] Baker K.R., Scudder G.D., Sequencing with earliness and tardiness penalties: A review, *Operations Research*, 1990,30, 22–36.

[19] Baker K.R., Su Z.S., Sequencing with due dates and early start times to minimize tardiness, *Naval Research Logistics Quartely*, 1974, 21, 171–176.

[20] Balas E., Vazacopoulous A., Guided Local Search with Shifting Bottleneck for Job Shop Scheduling, Management Science Technical Report MSRR-609(R), Carnegie Mellon University, Pitsburgh, PA 15213, 1995.

[21] Battiti R., Tecchiolli G., The reactive tabu search, *ORSA Journal on Computing*, 1994, 6, 126–140.

[22] Belov I.S., Stolin Y.N., An algorithm in a single path operations scheduling problem, *Mathematical Economics and Functional Analysis (Russian)*, 1974, Nauka, Moscow, 248–257.

[23] Birewar D.B., Grossmann I.E., Incorporating scheduling in the optimal design of multiproduct batch plants, *Computers and Chemical Engineering*, 1989, 13, 141–161.

[24] Bitran G.R., Chang L., A mathematical programming approach to a deterministic Kanban system. *Management Science*, 1987, 33, 427–441.

[25] Błażewicz J., Złożoność obliczeniowa problemó kombinatorycznych, WNT 1988.

[26] Błażewicz J., Ecker K., Schmidt G., Węglarz J., Scheduling in Computer and Manufacturing Systems, Springer-Verlag, 1993.

[27] Błażewicz J., Domschke W., Pesch E., The job shop scheduling problem: Conventional and new solution techniques, *European Journal of Operational Research*, 1996, 93, 1–33.

[28] Brah S.A, Scheduling in a flow shop with multiple processors, Ph.D. Dissertation, University of Houston, TX, 1988.

[29] Bratley P., Florian M., Robillard P., On sequencing with earliest starts and due dates with application to computing bounds for (n/m/G/F) problem, *Naval Research Logistics Quartely*, 1973, 20, 57–67.

[30] Brucker P., Jurish B., Sieviers B., A Fast Branch&Bound Algorithm for the Job-Shop Problem, *Discrete Applied Mathematics*, 1994, 49, 107–127.

[31] Brucker P., Scheduling algorithms, Springer-Verlag, Berlin, 1995.

[32] Buffa E.S., Taubert W.L., Production-Inventory Systems: Planning and Control. Richard D. Irwin, Homewood, 1972.

[33] Buzacott J.A., Automatic transfer lines with buffer stocs, *International Journal of Production Research*, 1967, 5, 183–200.

[34] Cai X., V-shape property for job sequences that minimize the expected completion time variance, *European Journal of Operational Research*, 1996, 91, 118–123.

[35] Campbell H.G., Dudek R.A., Smith M.L., A heuristic algorithm for the n job m machine sequencing problem. *Management Science*, 1970, 16, B630–637.

[36] Carlier J., The one-machine sequencing problem, *European Journal of Operational Research*, 1982, 11, 42–47.

[37] Carlier J., Scheduling jobs with release dates and tails on identical machines to minimize makespan, *European Journal of Operational Research*, 1987, 29, 298–306.

[38] Carlier J., Pinson E., An Algorithm for Solving the Job-Shop problem. *Management Science*, 1989, 35, 164–176.

[39] Carlier J., Pinson E., Jackson's pseudo-preemptive schedule for the $Pm|r_i,-q_i|C_{max}$ scheduling problem, Technical Report, Universite de Technologie de Compiegne, HEUDIASYC, 1996.

[40] Chapman S.N., Just-in-time supplier inventory and empirical implemmentation model, *International Journal of Production Research*, 1989, 27, 1993–2007.

[41] Chaudhury A., Whinston A.B., Towards an adaptive Kanban system, *International Journal of Production Research*, 1990, 28, 437–458.

[42] Chen H.G., Operator Scheduling Approaches in Group Technology Cells – Information Request Analysis, *IEEE Transactions on Systems, Man, and Cybernetics*, 1995, 25, 438–452.

[43] Chen B., Analysis of Classes of Heuristics for Scheduling a Two-Stage Flow Shop with Parallel Machines at One Stage, *Journal of Operational Research Society*, 1995, 46, 234–244.

[44] Chen C.L., Vempati V.S., Aljaber N., An application of genetic algorithm for flowshop problems, *European Journal of Operational Research*, 1995, 80, 389–396.

[45] Cheng T., Optimal Common Due Date With Limited Completion Time Deviation, *Computer and Operations Research*, 1988, 15, 91–96.

[46] Cheng T.C.E. , Gupta M.C., Survey of scheduling research involving due date detemination decisions, *European Journal of Operational Research*, 1989, 38, 156–166

[47] Cheng C.C., Smith S.F., Applying Constraint Satisfaction Techniques to Job Shop Scheduling, Technical Report CMU-RI-TR-95-03, Carnegie Mellon University, Pittsburgh, 1995.

[48] Cheng L.P. , Ding F.Y., Modifying Mixed-Model Assembly-Line Sequencing Methods to Consider Weighted Variations for Just-in-Time Production Systems, *IIE Transactions*, 1996, 28, 919–927.

[49] Choi J.W., Investment in the Reduction of Uncertainties in Just-in-Time Purchasing Systems, *Naval Research Logistics*, 1994, 41, 257–272.

[50] Co H.C. , Jacobson S.H., The Kanban Assignment Problem in Serial Just-in-Time Production Systems, *IIE Transactions*, 1994, 26, 76–85.

[51] Crawford K.M., Blackstone J.H.Jr, Cox J.F., A study of JIT implementation and operating problems, *International Journal of Production Research*, 1988, 26, 1561–1568.

[52] Dannenbring D.G., An evaluation of flow-shop sequencing heuristics. *Management Science*, 1977, 23, 1174–1182.

[53] Das H., Cummings P.T., Le Van M.D., Scheduling of serial multiproduct batch processes via simulated annealing, *Computers and Chemical Engineering*, 1990, 14, 1351–1362.

[54] Davidor Y., A Naturally Occuring Nice & Species Phenomenon: The model and First Results, *Proc. of 4'th Int. Conf. on Genetic Alg.*, 1991, Morgan Kaufmann Publishers, 257–263.

[55] Deal D.E., Yang T., Hallquist S., Job scheduling in petrochemical production: two-stage processing with finite intermediate storage, *Computers and Chemical Engineering*, 1994, 18, 33–344.

[56] Del'Amico M., Trubian M., Applying Tabu Search to the Job-Shop Scheduling Problem, *Annals of Operations Research*, 1993, 41, 213–252.

[57] Deleersnyder J.L., Hodgson T.J., Muller H.M., O'Grady P.J., Kanban clontrolled pull systems: An analytic approach, *Management Science*, 1989, 35, 1079–1091.

[58] Della Croce F., Narayan V., Tadei R., The two-machine total completion time flow shop problem, *European Journal of Operational Research*, 1996, 90, 227–237.

[59] Diaz B.A., Restricted neighborhood in the tabu search for the flowshop problem, *European Journal of Operational Research*, 1992, 62, 27–37.

[60] Ding F.Y., Cheng L.P., A Simple Sequencing Algorithm for Mixed-Model Assembly Lines in Just-in-Time Production Systems, *Operations Research Letters*, 1993, 13, 27–36.

[61] Ding F.Y., Kittichartphayak D., Heuristics for scheduling flexible flow lines, *Computers Industrial Engineering*, 1994, 26, 27–34.

[62] Dorigo M., Maniezzo V., Colorni A., Ant System: Optimization by a Colony of Cooperating Agents, *IEEE Transactions on Systems, Man, and Cybernetics. Part B: Cybernetics*, 1996, 26, 29–41.

[63] Dutta S.K., Cunningham A.A.,Sequencing two-machine flow-shops with finite intermediate storage, *Management Science*, 1975, 21, 989–996.

[64] Eck B., Pinedo M., Good Solution to Job Scheduling Problems Via Tabu Search. Presented at Joint ORSA/TIMS Meeting, Vancouver 1989.

[65] Edosomwan J.A., Marsh C., Streamlining the material flow process for just-ini-time production, *Industrial Engineering*, 1989, 21, 46–50.

[66] El-Rayal T.E., Hollier R.H., A review of plant design techniques, *International Journal of Production Research*, 1970, 8, 263–279.

[67] Ernst R., Pyke D.F., Optimal Base Stock Policies and Truck Capacity in a 2-Echelon System, *Naval Research Logistics*, 1993, 40, 879–903.

[68] Eshelman L.J., Schaffer J.D., Preventing Premature Convergence in Genetic Algorithms by Preventing Incest, *Proc. of 4'th Int. Conf. of Genetic Alg.*, 1991, Morgan Kaufmann Publishers, 115–122.

[69] Fial T., Közelitö algorithmus a haróm gép problémára, *Alkalmazott Matematikai Logopok*, 1977, 3, 389–398.

[70] Fisher H., Thompson G.L., Probabilistic Learning Combinations of Local JobShop Scheduling Rules. [In:] J.F. Muth and G.L. Thompson (Eds.), *Industrial Scheduling*, Prentice Hall, Englewood, Chichester, U.K., 1963.

[71] Flapper S.D.P.,Miltenburg J., Wijngaard J., Embedding JIT into MRP. *International Journal of Production Research*, 1991, 29, 329–341.

[72] Foo Y.P.S., Takefuji Y., Stochastic neural networks for solving job-shop scheduling, *Proc. of IEEE Int. J. Conf. on Neural Networks*, 1988, 275–290.

[73] Foo Y.P.S., Takefuji Y., Integer-linear programming neural networks for job-shop scheduling, *Proc. of IEEE Inter. J. Conf. on Neural Networks*, 1988, 341–348.

[74] Foster G., Horngren C.T., Costs accounting and cost management in a JIT environment, *J. Cost. Mgmt.*, 1988, 4–14

[75] French S., Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop. Horwood, Chichester, UK, 1982.

[76] Funk J.L., A comarison of inventory cost reduction strategies in a JIT manufacturing system. *International Journal of Production Research*, 1989, 27, 1065–1080.

[77] Garey M.R., Johnson D.S., Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H.Freeman and Co, New York, 1979.

[78] Garey M.R., Tarjan R.E., Wilfong G.T., One-Processor Scheduling with Symmetric Earliness and Tardiness Penalties, *Mathematics of Operational Research*, 1988, 13, 330–348.

[79] Gelders L.F., Sambandam N., Four simple heuristics for a flow-shop, *International Journal of Production Research*, 1978, 16, 221–231.

[80] Gilbert J.P., The state of JIT implementation in the U.S.A., *International Journal of Production Research*, 1990, 28, 1099–1109.

[81] Glauber R.J., Time-dependent statistics of the Ishing model, *Journal of Mathematical Phisics*, 1963, 4, 294–299.

[82] Globerson S., Millen R., Determining learning curves in gruop technology settings, *International Journal of Production Research*, 1989, 27, 1653–1664.

[115] Graells M., Espuña A., Puigjaner L., Sequencing intermediate products: A practical solution for multipurpose production scheduling, *Computers and Chemical Engineering*, 1996, 20, 1137–1142.

[116] Graham R.L., Bounds on multiprocessing timing anomalies, *SIAM Journal on Applied Mathematics*, 1969, 17, 263–269.

[117] Graham R.L., Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G., Optimization and approximation in deterministic sequencing and scheduling: a survey, *Annals of Discrete Mathematics*, 1979, 5, 287–326.

[118] Grau R., Espuña A., Puigjaner L., Completion times in multipurpose batch plants with set-up, transfer and clean-up times, *Computers and Chemical Engineering*, 1996, 20, 1143–1148.

[119] Gravel M., Price W.L., Using Kanban in a job-shop environment, *International Journal of Production Research*, 1988, 26, 1105–1118.

[120] Groeflin H., Luss H., Rosenwein M.B., Wahls E.T., Final assembly sequencing for just-in-time manufacturing, *International Journal of Production Research*, 1989, 27, 199–213.

[121] Grout J.R. , Christy D.P., An Inventory Model of Incentives for on-Time Delivery in Just-in-Time Purchasing Contracts, *Naval Research Logistics*, 1993, 40, 863–877.

[122] Gunaskekaran A., Goyal S.K., Martikainen T. , Yli-Olli P., Modeling and analysis of Just-In-Time manufacturing systems, *International Journal of Production Economics*, 1993, 32, 23–37.

[123] Gunasekaran A., Goyal S.K., Martikainen T., Yliolli P., Equipment Selection-Problems in Just-in-Time Manufacturing Systems, *Journal of Operational Research Society*, 1993, 44, 345–353.

[124] Gupta J.N.D., A functional heuristic algorithm for the flow-shop scheduling problem. *Operations Research Quarterly*, 1971, 22, 39–47.

[125] Gupta Y.P., A feasibility study of JIT-purchasing systems implementation in a manufacturing facility, *International Journal on Operational Production Management*, 1987, 10, 31–41.

[126] Gupta Y.P., Bagchi P., Inbound freight consolidation under just-in-time procurements: Application of clearing models, *J.Bus.Logist.*, 1987, 8, 74–94.

[127] Gupta Y.P., Gupta M.C., A system dynamics model for a muti-stage multiline dual-card JIT-kanban system, *International Journal of Production Research*, 1989, 27, 309–352.

[128] Gupta S.K., Kyparisis J., Single machine scheduling research, *OMEGA International Journal of Management Science*, 1987, 15, 207–227.

[129] Hahm J., Yano C.A., The Economic Lot and Delivery Scheduling Problem - The Common Cycle Case, *IIE Transactions*, 1995, 27, 113–125.

[130] Hahn C.K., Pinto P.A., Bragg D.J., Just-in-time production and purchasing, *Journal of Purchasing Matl. Management*, 1983, 19, 2–10.

[131] Hall N.G., Kubiak W., Sethi S.P., Earliness–Tardiness Scheduling Problems, II: Deviation of Completion Times About a Restrictive Common Due Date, *Operations Research*, 1991, 39, 847–856.

[132] Hall L.A., Shmoys D.B., Jackson's rule for single-machine scheduling: Making a good heuristic better, Technical Report, 1990.

[133] Hall R.W., Zero inventories. Down–Jones–Irwin, Homewood, 1983.

[134] Hall R.W., Attaining Manufacturing Excellence. Down–Jones–Irwin, Homewood, 1987.

[135] Hall N.G., Posner M.E., Earliness-Tardiness Scheduling Problems, I: Weighted Deviation of Completion Times About a Common Due Date, *Operations Research*, 1991, 39, 836–846.

[136] Hall N.G., Sriskandarayah C., A survey of machine scheduling problems with blocking and no-wait in process, *Operations Research*, 1996, 44, 510–525.

[137] Hay E.J., The Just-In-Time Breakthrough. Wiley, New York, 1988.

[138] Herring B., Simulation helps tire manufacturer change from push to pull system in controling material flow, *Industrial Engineering*, 1989, 21, 38–40.

[139] Herrmann J.W., Lee C.Y., On scheduling to minimize earliness-tardiness and batch delivery costs with a common due date, *European Journal of Operational Research*, 1993, 70, 272–288.

[140] Ho J.C., Chang Y.L., A new heuristic for the n-job, m-machine flow-shop problem. *European Journal of Operational Research*, 1990, 52, 194–206.

[141] Ho J.C., Flowshop sequencing with mean flowtime objective, *European Journal of Operational Research*, 1995, 81, 571–578.

[142] Hodgson T.J., Wang D., Optimal hybrid push/pull control strategies for a parallel multistage system. Part I. *International Journal of Production Research*, 1991, 29, 1279–1287.

[143] Hoogeveen H., Hurkens C., Lenstra J.K., Vandervelde A., Lower bounds for the multiprocessor flow shop, Presented at 2nd Workshop on Models and Algorithms for Planning and Scheduling, Wernigerode, May 1995.

[144] Hoogeveen J.A., Kawaguchi T., Minimizing total completion time in a two-machine flowshop: analysis of special cases, *Working paper*, Eindhoven University of Technology, 1995.

[145] Hopfield J.J. , Tank D.W., "Neural" computation of decisions in optimization problems, *Biological Cybernetics*, 1985, 52, 141–152.

[146] Huang P.V., Rees L.P., Taylor III B.W., A simulation analysis of the Japaneese Just-in-time technique (with Kanbans) for a multi-line, multi-stage production system. *Decision Science*, 1983, 14, 326–344.

[147] Hubscher R., Glover F., Applying Tabu Search with Influential Diversification to Multiprocessor Scheduling. Technical Report, University of Colorado at Boulder, 1992.

[148] Hundal T.S., Rajgopal J., An extension of Palmers's heuristic for the flow-shop scheduling problem. *International Journal of Production Economics*, 1988, 26, 1119–1124.

[149] Hunsucker J.L., Shah J.R., Comparative performance analysis of priority rules in a constrained flow shop with multiple processors environment. *European Journal of Operational Research*, 1994, 72, 102–114.

[150] Ignall E., Silver E.A., The output of a two stage system with unreliable machines and limited storage, *Industrial Engineering*, 1987, 19, 50–55.

[151] Im J.H., Lee S.M., Implementation of just-in-time systems in US manufacturing firms, *International Journal on Operational Production Management*, 1989, 9, 5–14.

[152] Inman R.R., Bulfin R.L., Sequencing JIT mixed-model assembly lines, *Management Science*, 1991, 37, 901–904.

[153] Ishibuchi H., Misaki S., Tanaka H., Modified simulated annealing algorithms for the flowshop sequencing problems, *European Journal of Operational Research*, 1995, 81, 388–398.

[154] Jackson J.R., Scheduling a production line to minimize maximum tardiness, Research Report, Management Science Research Project, University of California, Los Angeles, 1955.

[155] Jung J.H., Lee H.K., Yang D.R., Lee I.B., Completion times and optimal scheduling for serial multi-product processes with transfer and set-up times in zero-wait policy, *Computers and Chemical Engineering*, 1994, 18, 537–544.

[156] Kachur R.G., Electronic firm combines plant move with switch to JIT manufacturing, *Industrial Engineering*, 1989, 21, 44–48.

[157] Kadłuczka P., Wala K., Wyniki testowania metody szukania o zmiennej głębokości na przykładach permutacyjnych zagadnień optymalizacji, *Zeszyty Naukowe AGH, Ser. Automatyka*, 1991, 59, 127–134.

[158] Kanet J., Minimizing the Average Deviation of Job Completion Times About a Common Due Date, *Naval Research Logistics Quartely*, 1981, 28, 643–651.

[159] Karmarkar U.S., Lot sizes, lead times and in-process inventories, *Management Science*, 1987, 33, 409–418.

[160] Karmarkar U.S., Kekre S., Freeman S., Lotsizing and lead time performance in a manufacturing cell. *Interfaces*, 1985, 15, 1–9.

[161] Kim T.M., Just-in-time manufacturing system: A periodic pull system, *International Journal of Production Research*, 1985, 23, 553–562.

[162] Kim Y.D., Lim H.G., Park M.W., Search heuristics for a flowshop scheduling problem in a printed circuit board assembly process, *European Journal of Operational Research*, 1996, 91, 124–143.

[163] Kimura O., Terada H., Design and analysis of pull system: A method of multi-stage production control, *International Journal of Production Research*, 1981, 19, 241–253.

[164] King J.R., Spachis A.S., Heuristics for Flowshop Scheduling, *International Journal of Production Research*, 1980, 19, 345–357.

[165] Kirkavak N., Dincer C., Performance Evaluation Models for Single-Item Periodic Pull Production Systems, *Journal of Operational Research Society*, 1996, 47, 239–250.

[166] Kirkpatrick S, Gelatt C.D., Vecchi M.P., Optimisation by simulated annealing, *Science*, 1983, 220, 671–680.

[167] Kise H., Ibaraki T., Mine H., Performance analysis of six approximation algorithms for the one-machine maximum lateness scheduling problem with ready times, *Journal of the Operations Research Society of Japan*, 1979, 22, 205–244.

[168] Krajewski L.J., King B.E., Ritzman L.P., Wong D.S., Kanban, MRP and shaping the production environment, *Working Paper Series 83–19*, College of Administrative Science, Ohio State University, 1983.

[169] Krone M.J., Steiglitz K., Heuristic-programming solution of a flow-shop scheduling problem, *Operations Research*, 1974, 22, 629–638.

[170] Ku H.M., Karimi I., Completion time algorithm for serial multiproduct batch processes with shared storage, *Computers and Chemical Engineering*, 1990, 14, 49–69.

[171] Kubiak W., A pseudo-polynomial algorithm for a two-machine no-wait job-shop scheduling problem, *European Journal of Operational Research*, 1989, 43, 267–270.

[172] Kubiak W., Completion time variance minimization on a single machine is difficult, *Operations Research Letters*, 1993, 14, 49–59.

[173] Kubiak W., Sethi S., A note on "Level schedules for mixed-model assembly lines in just-in-time production systems", *Management Science*, 1991, 37, 121–122.

[174] Kubiak W., Minimizing variation of production rates in just-in-time systems: A survey, *European Journal of Operational Research*, 1993, 259–271.

[175] Koulamas C., Single-machine scheduling with time windows and earliness and tardiness penalties, *European Journal of Operational Research*, 1996, 91, 190–202.

[176] Kunde M., Steppat H., First fit decreasing scheduling on uniform multiprocessors, *Discrete Applied Mathematics*, 1985, 10, 165–177.

[177] Kuriyan K., Reklaitis G.V., Scheduling network flowshops so as to minimize makespan, *Computers and Chemical Engineering*, 1989, 13, 187–200.

[178] Kusiak A., Aplication of operational research models and techniques in flexible manufacturing systems, *European Journal of Operational Research*, 1986, 24, 336–345.

[179] Kusiak A., Heragu S., The facility layout problem, *European Journal of Operational Research*, 1987, 29, 229–251.

[180] Van Laarhoven P.J.M., Aarts E.H.L., Lenstra J.K., Job Shop Scheduling by Simulated Annealing. *Operations Research*, 1992, 40, 113–125.

[181] Labetoulle J., Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G., Preemptive scheduling of uniform machines subject to release dates, *Progress in Combinatorial Optimization*, 1994, Academic Press, Florida, 245–261.

[182] Lageweg B.J., Lenstra J.K., Rinnooy Kan A.H.G., Job-Shop Scheduling by Implicit Enumeration, *Management Science*, 1977, 24, 441–450.

[183] Lai T.C., A Note on Heuristics of Flow-Shop Scheduling, *Technical Report*, National Taiwan University, 1995.

[184] Lakshiminarayan I., Lakshanan R., Papineau R., Rochete R., Optimal single-machine scheduling with earliness and tardiness penalties, *Operations Research*, 1978, 26, 1079–1082.

[185] Larson N., Kusiak A., Work-in-Process Space Allocation – A Model and an Industrial Application, *IIE Transactions*, 1995, 27, 497–506.

[186] Lawler E.L., Optimal sequencing of a single machine subject to precedence constraints, *Management Science*, 1973, 19, 544–546.

[187] Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G., Recent Developments in Deterministic Sequencing and Scheduling: A Survey. [In:] M.A.H. Dempster, Lenstra J.K. , Rinnooy Kan A.H.G. (Eds.) *Deterministic and Stochastic Scheduling*, Reichel, Dordrecht, The Netherlands, 1982.

[188] Lawrence S., Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques (Supplement). Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, 1984.

[189] Lee H.K., Jung J.H., Lee I.B., An evolutionary approach to optimal synthesis of multiproduct batch plant, *Computers and Chemical Engineering*, 1996, 20, 1149–1157.

[190] Lee L.C., Parametric appraisal of the JIT systemm, *International Journal of Production Research*, 1987, 25, 1415–1429.

[191] Lenstra J.K.: Sequencing by enumerative methods, Mathematical Centre Tracts 69, Mathematish Centrum, Amsterdam, 1977.

[192] Lenstra J.K., Rinnooy Kan A.H.G., Brucker P., Complexity of machine scheduling problems, *Annals of Discrete Mathematics*, 1977, 1, 343–362.

[193] Levulis R., Group Technology 1978: The State-of-the-Art. K.W. Tunnel Company, Chicago, 1978.

[194] Li A., Co H.C., A dynamic programming model for the kanban assignment problem in a multi-period production system. *International Journal of Production Research*, 1991, 29, 1–16.

[195] McCormick S.T., Pinedooo M.L., Shenker S., Wolf B., Sequencing in an Assembly Line with Blocking to Minimize Cycle Time, *Operations Research*, 1989, 37, 925–935.

[196] MacMahon G.B., Florian M., On scheduling with ready times and due dates to minimize maximum lateness, *Operations Research*, 1975, 23, 475–482.

[197] Mattfeld D.C., Kopfler H., Bierwirth C., Control of Parallel Population Dynamics: Social-Like Behavior of GA-Individuals Solves Large-Scale Job Shop Problems, Technical Report, Department of Economics, University of Bremen, 1995.

[198] Matsuo H.C., Suh C.J., Sullivan R.S., A Controlled Search Simulated Annealing Method for the General Jobshop Scheduling Problem. Working Paper 03-04-88, Department of Management, The University of Texas at Austin, 1988.

[199] Miltenburg J., Level schedules for mixed-model assembly lines in just-in-time production systems, *Management Science*, 1989, 35, 192–207.

[200] Miltenburg J., Wijngaard J., Designing and phasing in just-in-time production systems, *International Journal of Production Research*, 1991, 29, 115–131.

[201] Miltenburg J., On the Equivalence of Jit and MRP as Technologies for Reducing Wastes in Manufacturing, *Naval Research Logistics*, 1993, 40, 905–924.

[202] Miltenburg J., Steiner G., Yeomans S., A dynamic programming algorithm for scheduling mixed-model just-in-time production sytems, *Mathematical and Computer Modelling*, 1990, 13, 57-66.

[203] Miyazaki S., Nishiyama N., Hashimoto F., An adjacent pairwise approach to the mean flow-time scheduling problem, *Journal of the Operations Research Society of Japan*, 1978, 21, 287–299.

[204] Miyazaki S., Nishiyama N., Analysis for minimizing weighted mean flow-time in flow-shop scheduling, *Journal of the Operations Research Society of Japan*, 1980, 23, 118–132.

[205] Miyazaki Y., Ohta H., Nashiyama N., The optimal operating of kanban to minimize the total operation cost, *International Journal of Production Research*, 1990, 26, 1605–1611.

[206] Modi A.K., Karimi I.A., Design of multiproduct batch processes with finite intermediate storage, *Computers and Chemical Engineering*, 1989, 13, 127–139.

[207] Monden Y., How Toyota shortened supply lot production time, waiting time and conveyance time, *Industrial Engineering*, 1981, 22, 22–30.

[208] Monden Y., Toyota Production System. Institute of Industrial Enginners, Atlanta, 1983.

[209] Mosheiov G., V-Shaped Policies for Scheduling Deteriorating Jobs, *Operations Research*, 1991, 979–991.

[210] Mühlenbein H., Gorges-Schleuter I., Die Evolutionsstrategie: Prinzip für parallele Algorithmen, GMD Annual Report, 1988.

[211] Musier R.F.H., Evans L.B., An approximate method for the production scheduling of industrial batch processes with parallel units, *Computers and Chemical Engineering*, 1989, 13, 229–238.

[212] Nawaz M., Enscore Jr. E.E., Ham I., A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *OMEGA International Journal of Management Science*, 1983, 11, 91–95.

[213] Nissanci I., Nicoll A.D., Project planning network is integrated plan for implementing just-in-time, *Industrial Engineering*, 1987, 19, 50–55.

[214] Nori V.S., Sarker B.R., Cyclic Scheduling for a Multiproduct, Single-Facility Production System Operating Under a Just-in-Time Delivery Policy, *Journal of Operational Research Society*, 1996, 47, 930–935.

[215] Nowicki E., Smutnicki C., Problem kolejnoś—ciowy gniazdowy z maszynami równoległymi, *Materiały Konferencji: Problematyka budowy i eksploatacji maszyn i urządzeń w ujęciu systemowym*, Kraków 1986.

[216] Nowicki E., Smutnicki C., On lower bounds on the minimum maximum lateness on one machine subject to release dates, *OPSEARCH*, 1987, 24, 106–110

[217] Nowicki E., Smutnicki C., Resource constrained project scheduling. Basic properties. *Lecture Notes in Economics and Mathematical Systems* 351, Springer-Verlag, 1988, 229–235.

[218] Nowicki E., Smutnicki C., Algorytmy przybliżone rozwiązywania zagadnienia kolejnościowego postaci $1||\sum f_i$, *Zeszyty Naukowe Politechniki Śląskiej, Ser. Automatyka*, 1988, 99, 201–210.

[219] Nowicki E., Smutnicki C., Worst-case analysis of an approximation algorithm for flow-shop scheduling. *Operations Research Letters*, 1989, 8, 171–177.

[220] Nowicki E., Smutnicki C., Zdrzalka S., Algorytmy aproksymacyjne w wybranych zagadnieniach kolejnosciowych przy kryterium minimalizacji sumy kar, *Archiwum Automatyki i Telemechaniki*, 1989, XXXIV, z.3–4, 289–299.

[221] Nowicki E., Smutnicki C., Analiza porownawcza algorytmow aproksymacyjnych dla m-maszynowego problemu przeplywowego, *Materialy 1. Krajowej Konferencji Badan Operacyjnych i Systemowych BOS'88*, Ksiaz 1988, Instytut badan systemowych PAN, W-wa,, 1989, Tom 1, 168–176.

[222] Nowicki E., Smutnicki C., System wspomagania decyzji w harmonogramowaniu zadan, *Zeszyty Naukowe AGH, Ser. Automatyka*, 1989, 49, 237–245.

[223] Nowicki E., Smutnicki C., System wspomagania decyzji dla potrzeb harmonogramowania produkcji, *Zeszyty Naukowe Politechniki Śląskiej, Ser. Automatyka*, 1990, 102.

[224] Nowicki E., Smutnicki C., Analiza najgorszego przypadku algorytmow aproksymacyjnych dla problemu przeplywowego, *Zeszyty Naukowe Politechniki Śląskiej, Ser. Automatyka*, 1990, 100.

[225] Nowicki E., Smutnicki C., Worst-case analysis of Dannenbring's algorithm for flow-shop scheduling. *Operations Research Letters*, 1991, 10, 473–480.

[226] Nowicki E., Smutnicki C., New results in the worst-case analysis for flow-shop scheduling. *Discrete Applied Mathematics*, 1993, 46, 21–41.

[227] Nowicki E., Smutnicki C., A decision support system for resource constrained project scheduling problem, *European Journal of Operational Research*, 1994, 79, 183–195.

[228] Nowicki E., Smutnicki C., A note on worst-case analysis of an aproximation algorithm for flow-shop scheduling, *European Journal of Operational Research*, 1994, 74, 128–134.

[229] Nowicki E., Smutnicki C., An approximation algorithm for single-machine scheduling problem with release times and delivery times, *Discrete Applied Mathematics*, 1994, 48, 69–79.

[230] Nowicki E., Smutnicki C., A fast taboo search algorithm for the job shop. *Management Science*, 1996, 6, 797–813.

[231] Nowicki E., Smutnicki C., A fast taboo search algorithm for the flow shop. *European Journal of Operational Research*, 1996, 91, 160–175.

[232] Nowicki E., Smutnicki C., Flow shop with parallel machines. A tabu search approach. Report ICT PRE 30/95, Technical University of Wrocław, (accepted for publication in speacial issue of EJOR on Tabu Search in 1977).

[233] Occena L.G., Yokota T., Modelling of an automated guided vehicle system (AGVS) in a just-in-time (JIT) enivironment. *International Journal of Production Research*, 1991, 29, 495–511.

[234] Ogbu F.A., Smith D.K., The application of the simulated annealing algorithm to the solution of the $n/m/C_{\max}$ flowshop problem. *Computer and Operations Research*, 1990, 17, 243–253.

[235] O'Neal C.R., The buyer–seller linkage in a just-in-time environment, *Journal of Purchasing Matl. Management*, 1987, 23, 8–13.

[236] OR library. *Internet: http://mscmga.ms.ic.ac.uk/info.html.* Files flowshop1.txt, flowshop2.txt, jobshop1.txt, jobshop2.txt.

[237] Osman I.H., Potts C.N., Simulated Annealing for Permutation Flow Shop Scheduling, *OMEGA International Journal of Management Science*, 1989, 17, 551–557.

[238] Page E.S., An Approach to Scheduling Jobs on Machines, *Journal of Royal Statistics Society*, 1961, B23, 484–492.

[239] Palmer D.S., Sequencing Jobs Through a Multi-Stage Process in the Minimum Total Time - A Quick Method of Obtaining Near Optimum. *Operations Research Quarterly*, 1965, 16, 101–107.

[240] Panwalkker S., Smith M., Seidmann A., Common Due Date Assignment to Minimize Total Penalty for the One Machine Scheduling Problemm, *Operations Research*, 1982, 30, 391–399.

[241] Papadimitriou C.H., Kenellakis P.C., Flowshop Scheduling with Limited Temporary Storage, *Journal of Association for Computing Machinery*, 1980, 27, 533–549.

[242] Parnaby J., A systems approach to the implementation of JIT methodologies in Lucas industries, *International Journal of Production Research*, 1988, 26, 483–492.

[243] Pekny J.F., Miller D.L., Exact solution of the no-wait flowshop scheduling problem with a comparison to heuristic methods, *Computers and Chemical Engineering*, 1991, 15, 741–748.

[244] Perkins J.R., Kumar P.R., Optimal-Control of Pull Manufacturing Systems, *IEEE Transactions on Automatic Control*, 1995, 40, 2040–2051.

[245] Pesh E., Learning in Automated Manufacturing. A Local Search Approach, Springer-Verlag, 1994.

[246] Philipoom P.R., Rees L.P., Taylor B.W., Huang P.Y., An investigation of the factors influencing the number of Kanbans required in the implementation of the JIT technique with Kanbans, *International Journal of Production Research*, 1987, 25, 457–472.

[247] Philipoom P.R., Rees L.P., Taylor B.W., Huang P.Y., A mathematical programming approach for detrmining workcentre lotsizes in just-in-time system with signal Kanbans, *International Journal of Production Research*, 1990, 28, 1–15.

[248] Porteus E.L., Investing in reduced set-ups in the EOQ model, *Management Science*, 1985, 31, 998–1010.

[249] Porteus E.L., Optimal lotsizing, process quality improvement and set-up cost reduction, *Operations Research*, 1986, 34, 137–144.

[250] Potts C.N., Analysis of a heuristic for one-machine sequencing with release dates and delivery times, *Operations Research*, 1980, 28, 1436-1441.

[251] Potts C.N., Analysis of linear programming heuristic for scheduling unrelated parallel machines, *Discrete Applied Mathematics*, 1985, 10, 155–164.

[252] Rajagopalan D., Karimi I.A., Completion times in serial mixed-storage multiproduct processes with transfer and set-up times, *Computers and Chemical Engineering*, 1989, 13, 175–186.

[253] Rajendran C., Chaudhuri D., An efficient heuristic approach to the scheduling of jobs in a flow-shop, *European Journal of Operational Research*, 1991, 61, 318–325.

[254] Rajendran C., Chaudhuri D., A flowshop scheduling algorithm to minimze total flowtime, *Journal of the Operations Research Society of Japan*, 1991, 34, 28–46.

[255] Rajendran C., Heuristic algorithm for scheduling in a flowshop to minimize total flowtime, *International Journal of Production Economics*, 1993, 29, 65–73.

[256] Rajendran C., A heuristic for scheduling in flowshop and flowline-based manufacturing cell with multiple-criteria, *International Journal of Production Research*, 1994, 32, 2541–2558.

[257] Rajendran C., Heuristics for scheduling in flowshop with multiple objectives, *European Journal of Operational Research*, 1995, 82, 540–555.

[258] Ramsay M.L., Brown S., Tabibzadeh K., Push, pull and squeeze shop floor control with simulation, *Industrial Engineering*, 1990, 22, 39–45.

[259] Rand G.K., MRP, JIT and OPT, [In:] Hendry L.G. and Eglese R.W. (Eds.), Operational Research Society 1990. Birmingham, 103–136.

[260] Rees L.P., Huang P.Y., Taylor II B.W., A comparative analysis of an MRP lot-for-lot system and a Kanban system for a multi-stage operation, *International Journal of Production Research*, 1989, 27, 1427–1443

[261] Reeves C., Heuristics for scheduling a single machine subject to unequal job release times, *European Journal of Operational Research*, 1995, 80, 397–403.

[262] Reeves C.R., A genetic algorithm for flowshop sequencing, *Computer and Operations Research*, 1995, 22, 5–13.

[263] Richmond L.E., Blackstone J.H.Jr, Just-in-time in the plastics processing industry, *International Journal of Production Research*, 1988, 26, 27–34.

[264] Röck H., Schmidt G., Machine Aggregation Heuristics in Shop Scheduling. *Methods of Operations Research*, 1982, 45, 303–314.

[265] Rodrigues M.T.M., Gimeno L., Passos C.A.S., Campos T., Reactive scheduling approach for multipurpose chemical batch plants, *Computers and Chemical Engineering*, 1996, 20, 1215–1220.

[266] Ronen D., Perspectives on practical aspects of truck routing and scheduling, *European Journal of Operational Research*, 1988, 35, 137–145.

[267] Santos D.L., Hunsucker J.L., Deal D.E., Global lower bounds for flow shops with multiple processors, *European Journal of Operational Research*, 1995, 80, 112–120.

[268] Sarker B.R., Harris R.D., The effect of imbalance in a just-in-time production system: A simulation study, *International Journal of Production Research*, 1988, 21, 1–18.

[269] Sarker B.R., Fitzsimmons J.A., The performance of push and pull systems: A simulation and comparative study, *International Journal of Production Research*, 1989, 27, 1715–1732.

[270] Sassani F., A simulation study on performance improvement of group technology cells, *International Journal of Production Research*, 1990, 28, 293–300.

[271] Sawik T., Multilevel scheduling of multistage production with limited in process inventory, *Journal of Operational Research Society*, 1987, 38, 651–664.

[272] Sawik T., Hierarchical scheduling flow shop with parallel machines and finite in process buffers, *Archiwum Automatyki i Telemechaniki*, 1988, 33, 403–411.

[273] Sawik T., A scheduling algorithm for flexible flow lines with limited intermediated buffers. *Applied Stochastic Models and Data Analysis*, 1993, 9, 127–138.

[274] Sawik T., Scheduling flexible flow lines with no in-process buffers. *International Journal of Production Research*, 1995, 33, 1359–1370.

[275] Sawik T., Optymalizacja dyskretna w elastycznych systemach produkcyjnych, WNT, 1992.

[276] Sawik T., Planowanie i sterowanie produkcji w elastycznych systemach montażowych, WNT 1996.

[277] Scanzon L., JIT Theory Goes On Line At Sundstrand Data Control, *Industrial Engineering*, 1989, 34, 31–34.

[278] Schonenberger R.G., Some observations on the advantages and implementation issues of just-in-time prodURction systems, *Journal of Opertions Management*, 1982, 3, 1–11.

[279] Shonenberger R.J., Japaneese Manufacturing Techniques: Nine Lessons in Simplicity. The Free Press, New York, 1982.

[280] Schonberger R.J., Gilbert J.P., Just-in-time purchasing: A chalenge for U.S. industry. *Cali. Mgmt. Rev.* , 1983, XXVI, 54–68.

[281] Schrage L., Obtaining optimal solution to resource constrained network scheduling problem, unpublished manuscript, 1971.

[282] Schrorer B.J., Microcomputer analyzes 2-card Kanban system just-in-time small batch production, *Industrial Engineering*, 1984, 16, 54–65.

[283] Sevast'yanov S.V., On an asymptotic approach to some problems in scheduling theory, *Abstracts of papers at 3rd All-Union Conference of Problems of Theoretical Cybernetics (Russian)*, Novosibirsk, 1974, 67–69.

[284] Shaukat A. B., Hunsucker J.L., Branch and bound algorithm for the flow shop with multiple processors. *European Journal of Operational Research*, 1991, 51, 88–99.

[285] Shmoys D.B., C. Stein Wein J., Improved Approximation Algorithms for Shop Scheduling Problems, *Proceedings of the 2nd ACM-SIAM Symposium on Discrete Algorithms*, 1991.

[286] Sidney J.B., Optimal single-machine scheduling with earliness and tardiness penalties, *Operations Research*, 1977, 25, 62–69.

[287] Silver E.A., Peterson R., Decision Systems for Inventory Management and Production Planning. Wiley, New York, 1985.

[288] Simers D., Priest J., Gary J., Just-in-time techniques in process manufacturing reduced lead time, cost; raise productivity, quality. *Industrial Engineering*, 1989, 21, 19–23.

[289] Singh N., Shek K.H., Meloche D., The developement of a Kanban system: A case study. *International Journal on Operational Production Management*, 1990, 10, 28–36.

[290] Sipper D., Shapira R., JIT vs. WIP-a trade-off analysis, *International Journal of Production Research*, 1989, 27, 903–914.

[291] Smutnicki C., Problemy optymalnego szeregowania zadań w dyskretnych systemach produkcyjnych z kryterium minimalno-kosztowym, (praca doktorska), Technical Report 2/81, Technical University of Wrocław, 1981.

[292] Smutnicki C., Podejście blokowe w problemie kolejnościowym taśmowym z ograniczeniami składowania, *Zeszyty Naukowe Politechniki Śląskiej, Ser. Automatyka*, 1986, 84, 223–233.

[293] Smutnicki C., Dolne i górne ograniczenia dla problemu $F|r_j, prec|f_{\max}$, *Zeszyty Naukowe AGH, Ser. Automatyka*, 1987, 42, 169–180.

[294] Smutnicki C., DSS for project scheduling. A reviev of problems, *Lecture Notes in Economics and Mathematical Systems*, 1987, 337, Springer-Verlag, 211–216.

[295] Smutnicki C., Ocena algorytmów szeregowania zadań w systemie taśmowym, *Materiały Konferencji Komputerowe Systemy i Metody Wspomagania Decyzji*, Warszawa 1988, 373–385.

[296] Smutnicki C., Zagadnienia szeregowania z ograniczeniami czasów oczekiwania, *Zeszyty Naukowe Politechniki Śląskiej, Ser. Automatyka*, 1988, 94, 317–327.

[297] Smutnicki C., Algorytmy przyblizone dla m-maszynowego problemu przepływowego z ograniczeniami skladowania, *Materialy 1. Krajowej Konferencji Badań Operacyjnych i Systemowych BOS'88*, Ksiaz 1988, Instytut badan systemowych PAN, W-wa, Tom 1, 176–184.

[298] Smutnicki C., O pewnej klasie algorytmów aproksymacyjnych dla problemów szeregowania, *Zeszyty Naukowe Politechniki Śląskiej, Ser. Automatyka*, 1990, 100, 311–321.

[299] Smutnicki C., Problem szeregowania na jednej maszynie przy żądanych terminach zakończenia zadań, *Zeszyty Naukowe AGH, Ser. Automatyka*, 1991, 145–152.

[300] Smutnicki C., Ogólny gniazdowy problem szeregowania przy żądanych terminach zakończenia operacji, *Zeszyty Naukowe Politechniki Śląskiej, Ser. Automatyka*, 1991, 109, 243–252.

[301] Smutnicki C., Some results of the worst-case analysis for flow-shop scheduling, Technical Report PRE 8/95, (accepted for publication in *European Journal of Operational Research*).

[302] Smutnicki C., Tabu search algorithm for a class of single-machine scheduling problem, *Zeszyty Naukowe Politechniki Śląskiej, Ser. Automatyka*, 1994, 114, 259–266.

[303] Smutnicki C., Results of the worst-case analysis for some scheduling problem, *Proc. of 1995 INRIA/IEEE Symposium on Emerging Technologies and Factory Automation*, Paris, France, 1995, 105–115.

[304] Smutnicki C., System wspomagający rozwiązywanie problemów szeregowania i harmonogramowania, *Elektrotechnika*, 1995, 14, 38–46.

[305] Smutnicki C., Single-machine scheduling with min-max earliness and tardiness penalties, Technical Report 1995, (submitted to *Computing*).

[306] Smutnicki C., Worst-case performance evaluation for some scheduling algorithms, *Zeszyty Naukowe Politechniki Śląskiej, Ser. Automatyka*, 1996, 117, 265–274.

[307] Smutnicki C., Scheduling of flexible flow lines and work centers, *Zeszyty Naukowe Politechniki Śląskiej, Ser. Automatyka*, 1996, 118, 265–274.

[308] Smutnicki C., Approximation algorithms, *Proc. of Symposium of Opeations Research SOR'96*, Braunschewig, Germany, 1996, (in print).

[309] Spachis A.S., King J.R., Job-Shop Scheduling Heuristics with Local Neighborhood Search. *International Journal of Production Research*, 1979, 17, 507–526.

[310] Spence A.M., Porteus E.L., Set-up reduction and increased effective capacity, *Management Science*, 1987, 33, 1291–1301.

[311] Sriskandarajah C., Sethi S.P., Scheduling algorithms for flexible flowshops: Worst and average case performance. *European Journal of Operational Research*, 1989, 43, 43–60.

[312] Storer R.H., David Wu S., Vaccari R., New Search Spaces for Sequencing Problems with Application to Job Shop Scheduling, *Management Science*, 1992, 10, 1495–1509.

[313] Sugimori Y., Kasunoki F.C., Uchikawa S., Toyota production system and Kanban system, materialization of just-in-time and respect-for-human system. *International Journal of Production Research*, 1977, 15, 553–564.

[314] Taillard E., Some efficient heuristic methods for flow shop sequencing. *European Journal of Operational Research*, 1990, 47, 64–74.

[315] Taillard E., Benchmarks for basic scheduling problems, *European Journal of Operational Research*, 1993, 64, 278–285.

[316] Taillard E., Parallel Taboo Search Technique for the Jobshop Scheduling Problem. Working Paper ORWP 89/11 (revised version October 1992), Departement de Mathematiques, Ecole Polytechnique Federale De Lausanne, Lausanne, Switzerland, 1989.

[317] Taheri J., Northern telecom tackles succesfull implementation of cellular manufacturing, *Industrial Engineering*, 1990, 22, 38–43.

[318] Tandon M., Cummings P.T. , LeVan M.D., Flowshop sequencing with non-permutation schedules, *Computers and Chemical Engineering*, 1991, 15, 601–607.

[319] Toczyłowski E., Modelling FMS operational scheduling problems by m-processors, *Elektrotechnika*, 1995, 14, 429–436.

[320] Turner S., Booth D., Comparison of heuristics for flow shop sequencing. *OMEGA International Journal of Management Science*, 1987, 15, 75–78.

[321] Weng X., Ventura J.A., Scheduling about a given due date to minimize mean squared deviation of completion times, *European Journal of Operational Research*, 1996, 88, 328–335.

[322] Vaessens R.J.M., Aarts E.H.L., Lenstra J.K., Job Shop Scheduling by Local Search, *Memorandum COSOR 94-05*, Eidhoven University of Technology, 1994.

[323] Vaessens R.J.M., Generalised Job Shop Scheduling: Complexity and Local Search, Ph.D.Thesis, Ponsen & Looijen BV, Wageningen, The Netherlands, 1995.

[324] Vandervelde A., Minimizing the makespan in a muliprocesor flowshop, Master's thesis, Eindhoven University of Technology, 1994.

[325] Wala K., Gądek-Madeja H., Zagadnienia syntezy przybliżonych algorytmów sterowania dyskretnymi procesami przemysłowymi, *Zeszyty Naukowe AGH, Ser. Automatyka*, 1989, 49, 201–215.

[326] Wala K., Metody lokalnej optymalizacji w zagadnieniu harmonogramowania, *Zeszyty Naukowe Politechniki Śląskiej, Ser. Automatyka*, 1988, 95, 169–177.

[327] Wala K., Chmiel W., Operatory genetyczne dla permutacyjnych zagadnień optymalizacji, *Zeszyty Naukowe Politechniki Śląskiej, Ser. Automatyka*, 1994, 114, 272–287.

[328] Wala K., Chmiel W., Nowy schemat procesu genetycznego poszukiwania na przykładzie zagadnienia harmonogramowania z maszynami równoległymi, *Zeszyty Naukowe Politechniki Śląskiej, Ser. Automatyka*, 1995, 116, 67–77.

[329] Wala K., Chmiel W., Investigation of cross-over genetic operators for permutation optimization problems, *Elektrotechnika*, 1995, 14, 451–458.

[330] Wellsons M.C., Reklaitis G.V., Optimal schedule generation for a single-product production line-I. Problem formulation., *Computers and Chemical Engineering*, 1989, 13, 201–212.

[331] Wellsons M.C., Reklaitis G.V., Optimal schedule generation for a single-product production line-II. Identification of dominant unique path sequences, *Computers and Chemical Engineering*, 1989, 13, 213–227.

[332] Wemmerlöw U., Hyer N.L., Procudures for the part family/machine group identification problem in cellular manufacture, *Journal of Opertions Management*, 1986, 6, 125–147.

[333] Wemmerlöw U., Hyer N.L., Research issues in cellular manufacturing *International Journal of Production Research*, 1987, 25, 413–431.

[334] Werner F., On the Heuristic Solution of the Permutation Flow Shop Problem, *Computers and Operations Research*, 1993, 20, 707–722.

[335] Werner F., Winkler A., Insertion Techniques for the Heuristic Solution of the Job Shop Problem. Report, Technische Universität "Otto von Guericke", Magdeburg, 1992.

[336] Widmer M., Hertz A., A new heuristic method for the flow shop sequencing problem, *European Journal of Operational Research*, 1989, 41, 186–193.

[337] Wittrock R.J., An adaptable scheduling algorithm for flexible flow lines. *Operations Research*, 1988, 36, 445–453.

[338] Woodruff D.L., Zemel E., Hashing vectors for tabu search, *Annals of Operations Research*, 1993, 41, 123–137.

[339] Woodruff D.L., Proposals for Chunking and Tabu Search, Technical Raport, Graduate School of Management, Davis, 1994.

[340] Yeomans J.S., A Simple Sequencing Algorithm for Mixed-Model Assembly Lines in Just-in-Time Production Systems – Comment, *Operations Research Letters*, 1995, 16, 299–301.

[341] Zangwill W.I., From EOQ towards ZI. *Management Science*, 1987, 33, 1209–1223.

[342] Zangwill W.I., Mathematical Programming - A Unified Approach. Prentice-Hall, Englewood, Cliffs, NJ, 1969.

[343] Zhou D., Cherkassky V., Baldwin T.R., Olson D.E., A neural network approach to job-shop scheduling, *IEEE Transactions on Neural Networks*, 1991, 2, 175–179.

Wydawnictwa Politechniki Wrocławskiej
są do nabycia w następujących księgarniach:
„Politechnika"
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław
budynek A-1 PWr., tel. (0-71) 320 25 34,
„Tech"
plac Grunwaldzki 13, 50-377 Wrocław
budynek D-1 PWr., tel. (0-71) 320 32 52.
Prowadzimy sprzedaż wysyłkową

ISSN 0324-9786