A Timed State Space-Heuristic Search Framework for Colored Petri Net-based Scheduling of Discrete Event Systems —An Application to Flexible Manufacturing Systems

Olatunde Temitope Baruwa

PhD Thesis

May, 2015

A Timed State Space-Heuristic Search Framework for Colored Petri Net-based Scheduling of Discrete Event Systems

An Application to Flexible Manufacturing Systems

Olatunde Temitope Baruwa

A thesis presented for the degree of

Doctor of Philosophy

Advisor

Dr. Miquel Àngel Piera Eroles

Universitat Autònoma de Barcelona

PHD PROGRAM IN TELECOMMUNICATIONS AND SYSTEMS ENGINEERING
Department of Telecommunications and Systems Engineering
Universitat Autònoma de Barcelona
May, 2015

A Timed State Space-Heuristic Search Framework for Colored Petri Net-based Scheduling of Discrete Event Systems —An Application to Flexible Manufacturing Systems

Author
Olatunde Temitope Baruwa

Advisor
Dr. Miquel Àngel Piera Eroles

Thesis Committee - Members
Dr. Emilio Jimenez
Dr. Elzbieta Roszkowska
Dr. Gabriele Enea

Thesis Committee - Reserve
Dr. Juan Jose Ramos
Prof. Josep Casanovas Garcia

PhD Program in Telecommunications and Systems Engineering
Specialization in Modeling, Simulation, and Optimization of Logistic Systems
Universitat Autònoma de Barcelona

Dr. Miquel Àngel Piera Eroles, Associate professor at the Universitat Autònoma de Barcelona,

CERTIFIES:

That the doctoral thesis entitled **A Timed State Space-Heuristic Search Framework for Colored Petri Net-based Scheduling of Discrete Event Systems —An Application to Flexible Manufacturing Systems** by **Olatunde Temitope Baruwa**, presented in partial fulfillment of the requirements for the degree of Doctor of Philosophy, embodies original work done by him under my supervision.

_____

Dr. Miquel Àngel Piera Eroles

Logistics and Aeronautics Unit
Department of Telecommunications and Systems Engineering
School of Engineering
Univeristat Autònoma de Barcelona
May 2015

Abstract

To gain competitive advantage in the global market, manufacturers have to quickly adapt their systems to respond to fluctuating customer demands under high-quality service factors. The high capital investment in flexible manufacturing systems (FMSs) together with the challenges of the rapidly changing market conditions has made efficient resource utilization become essential. To maximize the benefits of an FMS, appropriate scheduling techniques must be put in place to fully exploit the manufacturing flexibilities. The overall objective of this thesis is to establish a scheduling framework based on timed colored Petri net (TCPN) modeling for optimizing the performance of FMSs through the development of tools and efficient search methods based on the reachability graph (or state space) analysis. Reachability graph analysis is a powerful tool that can be used to automate the decision-making activity in scheduling problems by tracking all the possible behaviors of the modeled system. However, it suffers from the state space explosion problem due to the computational complexity of production scheduling problems in FMSs. This has limited its applicability to small-sized problems.

In the proposed TCPN-based scheduling methodology, the generation of an optimal production schedule involves the construction and traversal of the state space with a search algorithm. Also, a simulator is required for executing the TCPN model. It is quite natural to use graph search algorithms since the underlying analysis method relies on the reachability graph. Graph search strategy is an interdisciplinary technique that spans across the fields of Artificial intelligence (AI), Operations research (OR), and Computer science. This thesis focuses on AI-based heuristic search methods used in simulating only the best scenarios (as a shortest-path search problem). In this method, the exploration of reachability graphs are guided with heuristic functions that rely on the knowledge of the production plans.

The contribution of this thesis is fourfold. The first provides the platform in which the other three contributions are implemented: an automated decision support and special purpose tool called TIMed State space Performance Analysis Tool (TIMSPAT). Because of the complex data structure, TCPN-based scheduling using reachability graph analysis has been merely looked at in the literature. The use of TCPN for scheduling purposes has often been limited to simulation only. Thanks to the common data structure of the heuristic search methods, TIMSPAT is capable of incorporating different search algorithms in a single executable tool. So far, nine algorithms have been implemented, which includes the search algorithms proposed in this thesis and those by other authors.

Second, a memory-efficient approach is developed to alleviate the scalability problem that appears in the state space exploration of FMS scheduling problems. It is aimed at reducing the memory requirements of layered search algorithms that are compelled to store all the generated states in memory to guarantee termination. The approach tackles the research questions: Is it necessary to store all the generated states to guarantee an optimal solution without revisiting

states? and how can we reduce the number of states to be stored so that larger problems can be solved without forgoing optimality? It assumes that the state space graphs of a system with increasing problem size may contain repetitive patterns while the underlying model structure remains as constant as possible. These repetitive patterns transform into structural state space equivalences determined by a new measure called layer detection scope. The challenge is to determine whether or not the FMS behavior follows a regular repetitive pattern for any problem size above a minimum problem size, regardless of a change in the problem size. The proposed solution is based on the notion that the structural behavior captured in the state space of a solvable smaller problem size can be extended to explore a larger size if the two problems share a certain kind of similarity. The repetitive patterns in the graph structures are identified and leveraged to optimize the scheduling problem for larger problem sizes which a priori cannot be solved by closely-related existing approaches. Among the problems solved are the multiple lot size scheduling problem with fixed layout configuration, and FMS problems of similar configurations where the problem size differ by the number of jobs, resources and operations. The approach outperforms previous works when repetitive FMS behavior influences scalability.

The third contribution presents two anytime heuristic search algorithms, developed to overcome the drawbacks of conventional heuristic search algorithms. An anytime algorithm trades off computation time and solution quality. It is capable of finding suboptimal solutions very quickly and continuously improves the solution quality until the solution converges to the optimal solution. This method has been proved successful in AI community. However, they are yet to be explored in the PN research community. The first anytime algorithm adapts and improves an existing anytime algorithm to TCPN-based scheduling, while the second proposes a new algorithm that combines two heuristic search algorithms making them anytime for deadlock-free scheduling. The algorithms are suitable for both off-line and on-line scheduling purposes due to their effectiveness in adapting to different CPU constraints. Also, they can be used in a scheduling/rescheduling mode whenever the system deviates from its original schedule or the system state changes due to disturbance or machine failure.

The overall scheduling of an FMS can be so complex that it cannot be handled in an integrated manner. Several scheduling approaches have treated scheduling problems in an independent manner. The last contribution presents a TCPN-based approach to the simultaneous scheduling of machines and automated guided vehicles (AGVs) with conflict-free routing. Unlike the existing approaches that employ a decomposition framework, the entire scheduling problem is described in a single model. Two simultaneous scheduling models are proposed and evaluated using an event-driven vehicle assignment solution as opposed to the traditional dispatching rules.

Keywords: Timed colored Petri net · Reachability graph · Condensed state space · Scheduling · Flexible manufacturing systems · Heuristic search · Optimization · Memory-efficient · Time-efficient · Anytime heuristic search · Deadlock-free · Simultaneous scheduling · Automated guided vehicles

List of Publications

This thesis is based on a collection of peer-reviewed research articles listed below.

Paper I. Baruwa OT, Piera MA. TIMSPAT —TIMed State Space Performance Analysis Tool for colored Petri net-based scheduling of discrete event systems: An application to flexible manufacturing systems; 2015. Submitted for publication in Computers & Industrial Engineering.

Paper II. Baruwa OT, Piera MA. Anytime heuristic search for scheduling flexible manufacturing systems: a timed colored Petri net approach. The International Journal of Advanced Manufacturing Technology 2014;75(1-4):123–137.

Paper III. Baruwa OT, Piera MA, Guasch A. Deadlock-free scheduling method for flexible manufacturing systems based on timed colored Petri nets and anytime heuristic search. Systems, Man, and Cybernetics: Systems, IEEE Transactions on 2015;45(5):831–846.

Paper IV. Baruwa OT, Piera MA. Identifying FMS repetitive patterns for efficient search-based scheduling algorithm: A colored Petri net approach. Journal of Manufacturing Systems 2015;35(0):120–135.

Paper V. Baruwa OT, Piera MA. A colored Petri net-based hybrid heuristic search approach to simultaneous scheduling of machines and automated guided vehicles; 2015. Revised manuscript submitted for publication in International Journal of Production Research.

In addition to the above, the following articles have been published during the research period, but are not included in this thesis.

VI. Tang J, Piera MA, Baruwa OT. A discrete-event modeling approach for the analysis of TCAS-induced collisions with different pilot response times. Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering 2015;(in press). doi: 10.1177/0954410015577147.

VII. Nosedal J, Baruwa O, Piera MA. Concurrent and distributed systems analysis using colored Petri nets. In: Actas de XXXIV Jornadas de Automática. Terrassa, Spain; 2013, p. 538–544.

VIII. Zuñiga CA, Piera MA, Baruwa OT. Pre-tactical trajectory de-confliction algorithm for air traffic management. In: 1st International Conference on Application and Theory of Automation in Command and Control Systems ATTACS2012. IRIT Press. ISBN 978-2-917490-20-4;2012, p. 233–237.

IX. Piera MA and Baruwa O. A discrete event system model to optimize runway occupancy. In: Proceedings of the 7th EUROCONTROL innovative research workshop and exhibition (INO'08); France; 2008, p 115–122.

X. Baruwa, OT, Piera, MA. Runway capacity optimization: aircraft sequencing in mixed mode operation. In: Proceedings of the 20th European Modeling and Simulation Symposium (EMSS), 2008, p. 579–585.

XI. Baruwa, OT, Piera, MA. A derivative control mechanism for supply chain performance improvement. In: Proceedings of the 22nd European Conference on Modelling and Simulation (ECMS), 2008, p. 270–276.

Acknowledgments

First and foremost, I would like to take this opportunity to express my sincere gratitude to my advisor Dr. Miquel Àngel Piera Eroles for his guidance and support throughout the research period. His brilliant advices and insightful suggestions have greatly helped me to fine tune my research ideas. This thesis would not have been completed without his intellectual and financial support.

I would like to appreciate the editors and anonymous reviewers of my papers (accepted and rejected) whose comments have helped to improve my scientific writing and the technical presentation of the papers. I wish to thank you all for the efforts and the time you have taken to understand the concepts presented in the papers.

I would like to thank the members of the defense committee, Dr. Emilio Jimenez, Dr. Elzbieta Roszkowska, and Dr. Gabriele Enea, for having accepted to serve on my examination board and for the time they have dedicated to reading and evaluating this thesis. Also, many thanks to Dr. Juan Jose Ramos and Prof. Josep Casanovas Garcia for accepting to be on the substitute list.

I am also grateful to my colleagues Jenaro Nosedal, Jun Tang, Sergio Navarro, Hugo Marenco and to those I have worked under as teaching assistants, Roman Buil and Daniel Riera. Jun Tang deserves a special mention for always going an extra mile to help me source for journal papers that are not part of the University's subscription database. I would like to acknowledge the help and support of Monica Gutierrez for designing the tool's website. Also, I appreciate her technical help in hardware purchase and resolution of technical issues. And to all the other members of the Logistics and Aeronautics unit, it has been a pleasure to be part of this great family.

I would like to extend my gratitude to the graduate program in Economics, Finance, and Management of the Pompeu Fabra University who provided me the first opportunity to study abroad through the Master's scholarship. The class lectures served as a stepping stone in getting to know the logistics world that triggered my interest in modeling and optimization.

Most importantly, I would like to thank my wife, Titilola Baruwa for her love, care, understanding and patience during this period. Thank you for being there and supporting me even through the hard times. Her help in proofreading the final version of this thesis is highly appreciated. And to my lovely daughter Elizabeth Oluwadamilola, you are the best thing that has ever happened to me in the past year. Your birth turned things around in my research work and you have been my source of inspiration since then. I would also like to thank my mother, siblings, and in-laws for their prayers and constant encouragement.

Finally, I would like to acknowledge the financial support received from the Universitat Autònoma de Barcelona.

Above all, I am grateful to God for seeing me through the PhD study.

x

Contents

List of Figures

List of Tables

1
Introduction

1.1 Manufacturing Systems: An Overview

The invention of the assembly line by Henry Ford in 1913 [3] marked the second industrial revolution in the manufacturing sector for its reduced labor and increased rate of production over the labor-intensive production system of interchangeable parts developed at the US Armories in the 19th century. It enjoyed success during this period as motor vehicles (automobile) became affordable for the low income earners and thus, expanding its outreach in the domestic market. The transfer line is characterized by high volume production with dedicated equipment best suited for mass production of identical products of a single product type.

As globalization sets in after the Second World War (1960s), market competition became intense as a result of the changes in the buying behavior of customers in advanced economies. This shifted the focus of the sector from a supplier-driven market to a customer-driven market. Product cost was no longer the main concern [4] as manufacturers had to put up with the challenge of rapidly changing market requirements and the delivery of custom-made products of high quality. The frequent changes in the market trend greatly reduced the life cycle of products, with customers craving for individualized products. This led to the production of smaller quantities of different product types. The traditional manufacturing systems of highly specialized transfer lines were too rigid and expensive to handle the production of varying product types.

To respond to these changes, flexibility and automation became critical to the survival of manufacturing companies in the fiercely competitive global market. Manufacturers have to quickly adapt their systems to meet fluctuating customer demands under high quality service factors. Modern day demands characterized by short product life cycle, high product variety and shorter delivery times, brought about a transition from mass production to highly automated and flexible manufacturing systems (FMSs) of mass customization (low to medium volume with product diversity).

Flexible manufacturing started off with the introduction of a numerically controlled (NC) machine center built in MIT in the 1950s. The center was originally intended to machine complex operations, but due to its reprogrammable capability to process various operations like drilling, milling, and boring, more NC machine centers were later developed and deployed for batch production. These machine centers were further equipped in the 1960s by the provision of automatic tool changers, and indexing work tables [3]. Soon after, control systems were integrated. This led to the emergence of computer NC (CNC) that provided several automation benefits including machine control programs, use of memory storage for part programs (a set of instructions that describe how parts are to be produced on machines) and ability to communicate with a central computer. The evolution took a step further when computer controlled (automated) material flow was built into the system with automated material handling, transport and storage systems like robots, automated guided vehicles (AGVs), and automated storage and retrieval systems.

The first fully automated FMS was pioneered in England in the 1960s, invented by Theodore Williams at Molins, Deptford, London. The system named "System 24" was installed to manufacture relatively complex alloy components for tooling in the tobacco industry [3]. Its name was coined from the fact that it can operate without human intervention for 24 hours under computer control. It comprises a group of CNC machines and an automated material handling system, together with a centralized computer control coordinates the flow of jobs between the machines. With the advancement in technology, FMS started gaining recognition worldwide in the 1980s and it has become a mainstream in the industry since then.

With the growing use of smart devices and social network, a fourth industrial revolution called Industry 4.0 had already begun. It is an initiative of the German high-tech strategy as part of the strategic action plan 2020. As a manufacturing strategy, Industry 4.0 (also known as SMART

factory) creates a collaborative cyber-physical system that allows parts, machines, humans, and resources to communicate with each other as in a social network via sensors, mobile devices, actuators, and internet of things [5]. The revolution is meant to shift the focus of productivity from the shop-floor level [6, 7]. The parts and machines within the system will be considered smart objects beyond the physical presence. Manufacturing will become highly flexible, and highly customized and smart products will be manufactured. However, it is still not clear how production management will be affected.

1.1.1 Flexible Manufacturing Systems

Maccarthy and Liu [8] defines an FMS as a production system capable of producing a variety of part types simultaneously at low to medium volumes, which consists of CNC or NC machine tools connected by an automated material handling system. It is operated under a centralized computer-control system. From this definition, an FMS consists of three subsystems: 1. A machining or processing subsystem: the group of CNC machines, 2. A material handling system (MHS): robots, AGVs and conveyors responsible for part movement, and 3. A computer control system.

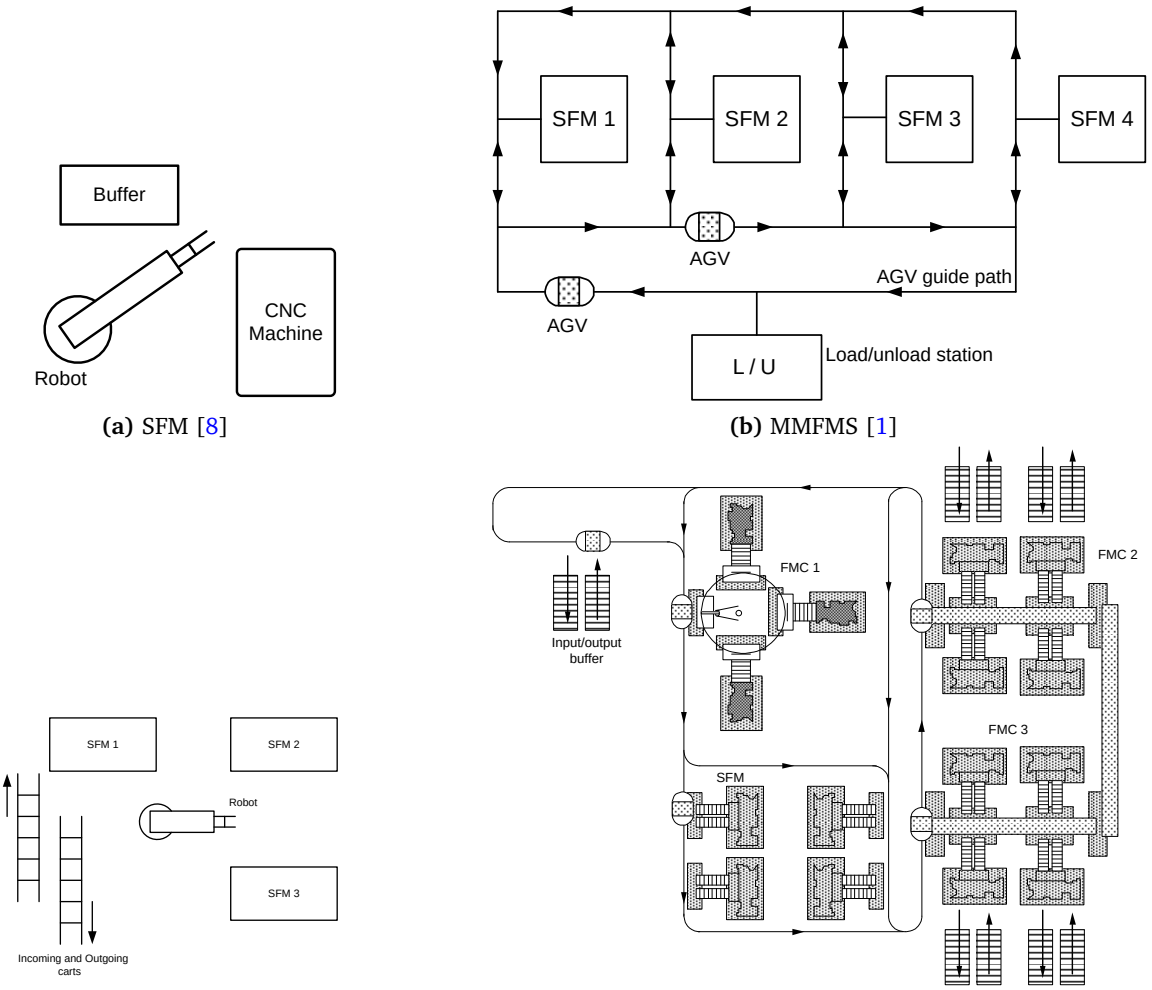
The cost involved in setting up an FMS is usually quite high, and not all companies can afford to install a full-scale FMS. This is true for most small-to-medium manufacturing enterprises (SMEs) like metal casting companies [9] where most of their production processes are labor intensive. [9] asserts that SMEs find it difficult to implement an FMS due to severe resource constraints and limited knowledge of automation methodologies. In these enterprises, the automated production of the entire manufacturing plant may not be possible. Instead, they invest in smaller versions of FMSs called cells [9, 10], to automate part of the manufacturing process rather than the whole system.

Different classification schemes have been proposed by several authors to describe the different types of FMSs based on their size, capacity, number of machines and layout arrangements, process characteristics, the flow pattern of parts through the system, the physical flow, and the number of part types [11–13]. In this thesis, we use the classification system given by Maccarthy and Liu [8]. Unlike the other classifications, they stress the importance of specific FMSs configuration taken into account the operational and control characteristics. The scheme distinguishes the operation of an FMS based on the number of machines and MHSs. Four types of FMSs are identified: 1. Single flexible machine (SFM), 2. Flexible manufacturing cell (FMC), 3. Multi-machine FMS (MMFMS), and 4. Multi-cell FMS (MCFMS). Some examples of these systems are given in Fig. 1.1. For convenience, we recall the definitions of these FMS classes as presented in [8].

Definition 1.1 ([8]). *An SFM is a computer-controlled production unit which consists of a single CNC or NC machine with tool changing capability, a material handling device and a part storage buffer. The material handling device in an SFM could be a robot or a special purpose pallet changing device. When an SFM is used as a component of a larger system, the material handling device may be removed if its function can be performed by the material handling devices of the larger system. Also, some SFMs may not have individual buffer storage depending on the system configuration.*

Definition 1.2 ([8]). *An FMC is a type of FMS consisting of a group of SFMs sharing one common material handling device.*

Definition 1.3 ([8]). *An MMFMS is a type of FMS which consists of a number of SFMs connected by an automated MHS which includes two or more material handling devices or is otherwise capable of visiting and serving two or more machines at a time.*



(a) SFM [8] **(b)** MMFMS [1]

(c) FMC [14] **(d)** MCFMS [15]

Fig. 1.1. Examples of different FMS configurations.

Definition 1.4 ([8])**.** *An MCFMS is a type of FMS which consists of a number of FMCs, and possibly a number of SFMs if necessary, all connected by an automated MHS.*

1.1.2 Manufacturing Flexibility and FMS Problems

Several attempts have been made to define manufacturing flexibility [16–18]. However, no consensus has been reached over a precise definition. For example, Gupta and Goyal [17] define flexibility as "the ability of a manufacturing system to cope with changing circumstances or instability caused by the environment", whereas, Upton [18] defines it from a comprehensive perspective as "the ability to change or react with little penalty in time, effort, cost or performance". Notwithstanding, most of the definitions given in the literature emphasizes on two aspects: the responsiveness to changes or uncertainties, and the speed and ease at which the system respond. From a different viewpoint, Gerwin [16] emphasized that "a basic issue that must be resolved in defining manufacturing flexibility is the level at which it is to be considered".

In a recent research survey on manufacturing flexibility [19], the authors reveal that the def-

initions, types, and dimensions of manufacturing flexibility have been defined in the late 1980s and the early 1990s. These definitions have been adopted and are still being used in the manufacturing domain (context) up to date. Seebacher and Winkler [19] produced a statistical review using citation analysis, that the most widely accepted and significant definition of flexibility in the literature can be traced to the work of Sethi and Sethi [20], followed by Gerwin [21] and Upton [18]. Other works that have similar impact in the early and late 1980s are Browne et al. [11], and Gerwin [16], Gupta and Goyal [17] respectively.

According to Sethi and Sethi [20], manufacturing flexibility is defined as the ability to reconfigure manufacturing resources so as to produce effectively different products of acceptable quality. Since the early works on manufacturing flexibility, different taxonomies have been used to classify the various types of flexibility. Browne et al. [11] provide the first eight categories of flexibility types that are then expanded to eleven by [20] classified into 3 categories that offers a strategic and operational dimension rather than an independent variable [22]. The definitions of the flexibility types are given as follows [20]:

1. Basic flexibilities
    - Machine flexibility: a measure of the ease with which the machine can process various operations.
    - Material handling flexibility: a measure of the ease with which different part types can be transported efficiently for proper positioning and processing within the manufacturing facility.
    - Operation flexibility: a measure of the ease with which alternative operation sequences can be used for processing a part type.
2. System flexibilities
    - Volume flexibility: a measure of the system's capability to be operated profitably at different volumes of existing part types.
    - Expansion flexibility: the ease with which the system's capacity and capability can be increased when needed.
    - Routing flexibility: the ability to produce a part by alternate routes through the system.
    - Process flexibility: the set of part types that the system can produce without major setups.
    - Product flexibility: the ease with which new parts can be added or substituted for existing parts.
3. Aggregate flexibilities
    - Program flexibility: ability of a system to run for reasonably long periods without external intervention.
    - Production flexibility: variety of part type that a system can produce without major investment in capital equipment.
    - Market flexibility: the ease with which the system can adapt to a changing market environment.

While the flexibility provided by FMSs gives a significant cutting-edge advantage [23], it does not automatically translate to benefits in the design and operation of FMS. Rather, it poses a challenging problem to the decision making process. Thus, it becomes necessary to address the major problems associated with the use of FMS. According to Stecke [24], there are three issues that must be dealt with at the decision support level: 1. Design problems, 2. Planning problems, and 3. Scheduling and control problems. These problems correspond to the three hierarchical levels of decision making (strategic, tactical and operational) with respect to the problem type and length of decision time involved.

Design problems involves the specification of the manufacturing requirements that includes the determination of the part types to be produced within the FMS (the determination of what part types to be processed and how they will be produced (process plan for each part type) within the FMS), the type of FMS and flexibilities required, type and capacity of the MHS, buffer design and the computer control architecture. These are long-term decisions that are made at the strategic level. Given the design decisions, Stecke [24] refers to FMS planning problems as decisions that have to be made before the start of production, typically at the medium-term (or at the tactical level). They address resource allocation problems like part type selection, machine grouping and loading problems [25].

FMS scheduling problems involves the assignment and sequencing of jobs on machines and the determination of the start and completion times of each job operation, while FMS control problems deal with continuous monitoring of the system and ensuring that production requirements are met and taking appropriate actions in the event of a failure or disturbance such as machine failure, preventive maintenance, arrival of new parts etc. This thesis focuses on FMS scheduling problems and the development of methods and algorithms through a quantitative approach. The high capital investment in FMS together with the challenges of the rapidly changing market conditions has made efficient resource utilization become essential. Appropriate scheduling techniques must be put in place to maximize the FMS benefits so as to fully exploit the manufacturing flexibilities. The thesis covers the flexibilities that are relevant to FMS scheduling problems, basic flexibilities and the routing and volume of system flexibilities.

1.2 FMS Scheduling

Scheduling is a decision making process that plays a vital role in improving the performance of an FMS. A challenging problem inherent to production scheduling is how to maximize the utilization of resources to perform a collection of tasks (jobs) while optimizing a certain performance measure in response to changing customers' demands and tight production requirements in an FMS context. The objective is to determine the optimal schedule based on a selected criterion (makespan, mean flow time, total tardiness etc) such that all technological precedence constraints are met, and production costs are minimized so as to maximize customer satisfaction under high quality factors. FMS scheduling problems are known to be very complex even for simple performance objectives and have been proved to be NP-hard since the computation time to obtain an optimal schedule grows exponentially with the problem size [26–28].

Let us recall some of the basic definitions related to FMS scheduling given by [29]:

- Machine operation: the processing over a continuous time period of a part on a machine
- Transport operation: the movement of parts from one machine to another in an FMS.

- Job: a collection of all the operations needed to complete the processing of a part. A job may contain a single part called single lot or a number of parts called multiple lot. A job or part may be used interchangeably in case of a single lot.
- Dispatching: the process or decision of determining the next operation for a resource when the resource becomes free, and determining the next destination of a part when the current operation has finished.
- Sequencing: the decision determining the order in which operations are performed on machines.
- Machine set up: the process or decision of reassigning tools on a machine in order to perform the next operation(s), from its initial state or working state arising from a previous operation.
- Part routing: the process of determining the machines in which each operation for a part is to be performed, i.e. determining the route or sequence of machines for each part passing through the system.

Scheduling can be performed either off-line or on-line [30]. The off-line approach (also called static scheduling) schedule all operations for the entire planning horizon in which all parts are assumed to be available before the start of activities. On the other hand, the on-line approach (sometimes referred to as dynamic scheduling [30]) attempts during execution or at real-time to schedule operations one at a time (or dynamically) as the system status changes or the scheduling decision is needed. In off-line scheduling, a complete schedule of all operations is required based on the complete knowledge of the system activities, whereas in on-line, the decision depends on if one operation needs to be scheduled (a partial schedule or dispatching) or a complete schedule needs to be produced (a revision or repair of the original schedule) called rescheduling. While the time to solve off-line scheduling is not critical, on-line scheduling is time-constrained such that a limited amount of computation time is given to produce a solution.

A wide range of scheduling problems has been developed. Liu and MacCarthy [29] give a comprehensive framework for FMS scheduling problems classification that takes into account all the significant factors which affects scheduling decisions. Five factors are used to characterize FMS scheduling problems based on system type, capacity constraints, job characteristics, production management environment, and performance measures. Independently of the FMS configuration (Fig. 1.1), there are two subsystems that must be taken into account during the scheduling process: 1. the machining system, and 2. the MHS. However, it is common to treat the subsystems as two independent problems to reduce the scheduling complexity.

In the literature, most works assume that machine is the only resource constraint in the system, called machine scheduling (MS). MS involves the processing of a set of n jobs $J = \{J_1, J_2, \ldots, J_n\}$ on a set of m machines $M = \{M_1, M_2, \ldots, M_m\}$. This is the method adopted for conventional production systems like the job shop [28]. In a typical job shop scheduling problem, transportation times are not taken into account, and MHSs are assumed to be always available whenever they are needed. Likewise, MHS scheduling (for AGVs) has been treated as an independent problem [31]. This creates a gap between the machining and transport systems. However, robots as MHSs with a different behavior to that of the AGV have been included with some machine scheduling problems. This is quite applicable in some FMS types like SFMs and FMCs where robots are used for part transfers. The importance of MHS cannot be ignored in the scheduling process. In [9], the automation of material handling has been shown to increase productivity and reduce labor costs especially in small firms with high labor-intensive production processes. They affirm that "65% of

the total production time is spent in manual handling of the material by human operators". As such, the scheduling of machines and MHS must be handled simultaneously to achieve an overall system performance. FMS scheduling problems are different from the traditional job shop problems [13] due to the consideration of machine and routing flexibility, limited buffer capacity, transportation capacity and time, and reduction of set-up times.

Besides the machines and MHSs, buffer capacity is another constraint that must be considered in FMS scheduling. Most FMS have limitations on the amount of work-in-process that can be held in the system [13]. This may lead to blocking, resource starving, and deadlock. The buffer capacity can vary depending on the system. When a zero-capacity buffer is considered, processed parts have to wait on the machine until the material handling device becomes available to transfer it for the next operation and/or the machine to be used for the next operation is free. These systems are commonly referred to as manufacturing systems with blocking [32]. The problems found under the buffer limitations and MHS constraints are usually denoted as deadlock-free scheduling [14].

Under the production management environment, scheduling decisions may involve processing a single lot size for each part type (single lot size scheduling) or several number of parts for each type (multiple lot size scheduling). In the multiple lot size scheduling, two possibilities exist. The products can be manufactured in a repetitive manner periodically (called cyclic scheduling) or in a non-cyclic manner [33]. From the findings of [33], non-cyclic scheduling is useful for small demand, whereas cyclic scheduling is better for medium-to-large demand when both methods are placed under the same conditions. Cyclic scheduling approaches are viable for high volume production environments with constant demand where the lot size of each part type is relatively large. There have been several arguments supporting the use of non-cyclic scheduling in an FMS environment [34–37]. This is due to the fact that the lot size of part types may be smaller [38], the part set may change frequently [39], parts may have different arrival rates, or the same cycle cannot be repeated for different operations in cluster tools used for semiconductor manufacturing [36]. Also, cyclic scheduling approaches are known to reduce the scheduling complexity of multiple lot size scheduling problems [40] since only the smallest set of the part types to be produced is considered in a cycle.

FMS scheduling has been well studied in the past few decades, and several solution methods have been proposed. As indicated by Lee and DiCesare [41], "a complete and general scheduling method must be able to formulate explicitly and concisely a scheduling problem and provide an efficient and general technique to solve the formulated problem". According to Balogun and Popplewell [42], there are six solution approaches to FMS scheduling problem: 1. Combinatorial optimization, 2. Artificial intelligence, 3. Simulation-based scheduling with dispatching rules, 4. Heuristics-oriented, 5. Multi-criteria decision making, and 6. Hybrid solution methods.

These scheduling approaches are known to provide effective solutions, but some like optimization methods from the OR domain require fitting FMS description in particular mathematical structures which sometimes introduce an oversimplification of the real dynamics. In modeling an FMS scheduling problem, mathematical programming approaches have limited applicability since it is quite difficult to mathematically describe practical constraints of FMS related to MHSs, resource-sharing situations, deadlocks, buffer sizes, routing flexibility, multiple lot sizes, and stochastic times [27].

1.2.1 Petri Nets

To provide a realistic and dynamic evaluation of the interdependencies between the subsystem components, FMS is formalized from a holistic modeling approach in which the system dynamics can be described by events. In such an approach, each operation is characterized by a certain number of preconditions, an estimated duration and a set of postconditions.

As a powerful graphical and mathematical modeling tool, Petri nets (PNs) support these dynamics and have been extensively used to model, simulate and analyze FMS characterized as discrete event systems where there is a high level of concurrency, parallelism, causal dependency, shared resources and synchronization [43]. PN has its origin in Carl Adam Petri's PhD dissertation on "Communication with automata" in 1962. Since its inception, PN has gained recognition in the research community to address several manufacturing problems including its application in a number of different disciplines like communications, robotics, engineering, business and air traffic management. Murata [44], Silva and Valette [45], Silva [46] provide a detailed overview of the applications of PN in flexible manufacturing.

A PN is a directed bipartite graph with two node types called places and transitions where the nodes are connected via directed arcs. A place can contain tokens and is used to describe resources in the system, a transition describes the event (the start or completion), and an arc is used to model the flow of tokens from places to transitions and from transitions to places. In an FMS description, a place can represent a resource or job status, a transition corresponds to an FMS operation whether machining or transportation, while an arc models the flow of product. Graphically, places, transitions and arcs are represented by circles, boxes, and arrows respectively.

PN modeling can be quite large and difficult to manage in terms of the size complexity when the number of resources and parts increases. Here, a high level PN modeling called colored PN (CPN) [47] is employed. CPN captures the high level abstraction of the system by enhancing the modeling power of PN with the use and manipulation of data attributes (values) called colors. It allows a concise representation of the system while maintaining the same modeling power of PN. Furthermore, CPN benefits go beyond the use of colored tokens in reducing the graphical size of the modeled system. For real industrial problems, there are several scheduling policies which rely on key information such as the specification of machine properties depending on the processing task, the due date of each product, the different penalties to meet the deadline for each customer etc. Also, CPN models can easily adapt to changes in the manufacturing environment such that the introduction of a new product, the addition or reduction of system resources and reconfigurations would require only a minimum amount of effort for model maintenance. The untimed CPN provides a qualitative approach to validating and verifying the system. However, it is usually not sufficient to correctly model a scheduling problem for system performance improvement.

Time representation plays a key role in the decision making of scheduling problems. In addition to its capability to describe the logical behavior and structure of discrete event systems in a concise manner with colored tokens, the inclusion of the time concept in CPN called timed CPN (TCPN) makes it possible to evaluate performance and investigate different scheduling strategies. With TCPN, quantitative measures like delays, durations and deadlines can be explicitly described. Time can be specified in the TCPN formalism as deterministic, interval (activity duration specified by an upper and lower bound), or stochastic. In this thesis, only deterministic times are considered.

The consideration of deterministic times is simply based on the fact that the processing times in FMS are often deterministic in nature since operations are computer-controlled [13]. However, this does not mean that stochastic situations cannot be handled. Unlike the job shop where a direct labor is required both during set up and machining operations [48], set ups between consecutive

operations are automated and variations in processing times are quite minimal. In less predictable environments where unplanned events such as machine breakdown and repairs, and dynamic arrival of new parts, can occur, stochastic times are considered appropriate to handle such events. Nonetheless, this is not the only approach to scheduling problems in such uncertain environments [49–51]. Deterministic scheduling methods like scheduling/rescheduling can also be employed whenever a disturbance occurs or the system deviates from its original schedule. Here, scheduling is solved as a static problem each time the system requires a new schedule. The approach takes into account the current state of the system (current shop-floor status) in an off-line scheduling manner [50]. It reacts to stochastic events during execution when the system deviates from the original schedule by revising the existing one and recomputing a new schedule. However, more time may be needed for schedule recomputation.

1.2.2 PN-based Scheduling Methodology

PNs can be executed to simulate the system behavior. The simulation capabilities of PNs allows the use of solution approaches in the operations research (OR), artificial intelligence (AI) and simulation domains. Its combination with these methods provides an integrated approach to optimize FMS scheduling problems. The PN-based scheduling methodology consists of two parts: 1. Modeling the FMS scheduling problem using TPN/TCPN as a representation formalism, and 2. Solving the scheduling problem by combining the execution of the TPN/TCPN model with an appropriate scheduling optimization technique. In this thesis, we use PN-based scheduling to cover both TPN and TCPN, while TPN-based and TCPN-based are used separately to exclusively refer to works using only the TPN and TCPN respectively. Although the focus is on FMS, the methodology can be applied to different classes of scheduling problems.

PN-based scheduling emerged from the works of Hillion et al. [52], Hillion and Proth [53], Carlier and Chretienne [54]. Hillion et al. [52], Hillion and Proth [53] use timed event-graphs, a special class of TPN to evaluate the steady-state performance of job-shop systems under a deterministic and cyclic production process. Carlier and Chretienne [54] study TPN schedules based on the firing sequence of the underlying PN. They propose two methods to compute the earliest schedule depending on the sequence length. A polynomial algorithm is used if the length is finite and an earliest state graph for the infinite case.

Tuncel and Bayhan [27] provide a comprehensive review of research works on the PN-based scheduling methodology. Of the six approaches given by Balogun and Popplewell [42], Tuncel and Bayhan [27] classify the existing PN-based approaches into four groups:

1. PN-based simulation with heuristic dispatching rules.
2. PN with graph search algorithms.
3. PN with mathematical programming approaches.
4. PN with metaheuristics.

Simulation with heuristic dispatching rules is typically used for on-line scheduling so that acceptable solutions can be obtained in a short period of time. PNs are converted into simulation models, and at each simulation step, dispatching rules are used to select the transition to fire from a set of conflicting (simultaneously-enabled) transitions. However, heuristic dispatching rules are considered myopic since they only use local information to make decision on the system's global objective.

Simulation models are known to be capable of analyzing the operational efficiency of the existing system configuration and provide the baseline to new viewpoints and aid the decision making of production managers. The benefits of simulation models have led to increased throughput, lead time improvement, and reduced production and inventory costs. Discrete-event simulation can be conducted to evaluate the performance of TCPN models using the CPN Tools software [55]. Earlier works [56–59] that have used TCPN simulation for scheduling purposes, employ a simplified version of the CPN formalism. A simple CPN can still be quite large in size compared to the advanced CPN formalism [47]. They do not take into account important features like transition guards, conditional expressions and functions. These features are made possible through the use of programming language expressions.

In spite of its advantages, simulation is deemed insufficient to evaluate the different alternatives of a system. For decision making purposes, it is only capable of exploring a limited number of scenarios when applied to improve the performance of an FMS. Due to the inherent flexibility, the performance optimization of FMS scheduling problems using simulation involves a large number of decision variables which requires a large number of simulation runs [60]. Also, it does not appear to be the only technique to obtain solutions in a timely manner, as the development of efficient techniques in the other categories has proven otherwise. In comparison with simulation, the use of mathematical techniques are rather limited since they require a particular class of PNs to model the FMS [27], and their applications are mostly found in cyclic scheduling [61].

In the PN with graph search algorithms category, the scheduling problem is formulated as a search problem through the generation of the reachability graph (or the state space) of the TPN/TCPN model. The reachability graph is used to generate all the possible sequences of transition firings from a given initial state to a desired or defined goal state where all the operations must have been completed and free from deadlock. Each of the firing sequences corresponds to a feasible schedule. As a quantitative analysis tool, decision support activity can be automated by analyzing all the possible alternatives of the system configuration with the aim of selecting the best alternative that minimize a given performance measure.

A basic intuition underlying the use of reachability graph is that the states of the system are represented as nodes and the transformation of these states (i.e. transition firing) that triggers a change in the state of the system, as edges. The graph can be constructed using classical search algorithms like breadth-first search (BFS) and depth-first search (DFS). Reachability graph analysis is a reliable and efficient method to obtain optimal schedules; however, the computational complexity is so high that it is practically impossible to explore the full state space of an industrial-sized system due to the explosion problem. The size of the state space grows exponentially with the size of the system. The number of possible combinations explodes so fast that it outgrows the computational resources within a small amount of time.

AI-based heuristic search methods [26, 62–64] have been proposed to simulate only the best scenarios (as a shortest-path search problem) by generating partial reachability graphs with heuristic functions that rely on the knowledge of the production plans (sequence of operations) captured from the FMS. AI-based heuristic search methods explores the reachability graph systematically [65] in an iterative manner. It starts from an initial state and constructs a partial schedule one at a time, until a complete schedule (goal state) is found. The construction of the complete schedule is obtained from the firing sequence from the given initial state to the goal state.

Tuncel and Bayhan [27] reviewed the works on PN-based scheduling with heuristic search algorithms until 2004. Since then, other heuristic search methods have emerged. From their review, the combination of TPN/TCPN modeling and AI-based heuristic search methods has proved to be an efficient method for solving FMS scheduling problems. However, majority of the works on PN-

based scheduling with AI-based heuristic search algorithms have used the TPN. From now on, the PN-based scheduling with AI-based Heuristic Search methods will be referred to as *PNAIHES*.

It is quite natural to use graph search algorithms for PN-based scheduling since the underlying analysis method relies on the reachability graph. As problem-independent solution methods, metaheuristics have been applied to solve PN-modeled FMS problems [66–68]. They are known to provide good solutions in short computational times but they cannot guarantee optimality. Unlike the graph search application where a stepwise transition firing is used to systematically build the graph, metaheuristics works with the complete firing sequence of transitions (schedule) described as the candidate solution to the scheduling problem. They use a different representation that requires coding and decoding schemes for the PN firing sequence. For instance, the genetic algorithm first encodes a candidate solution called chromosome, with bit strings in which each position in the string is interpreted as a gene. After the generation of new candidate solutions, a decoding scheme is then used to enable a PN simulator check whether the solution is feasible or not. Unlike in PN where a transition in the firing sequence can be directly represented as a gene in a chromosome [66–68], it could be a great challenge for TCPN where a transition can be fired with different set of token colors. A bit-string representation may not be sufficient to describe the firing sequence. Also, an operation described in a TCPN can be a group of more than one transition that must be fired sequentially. Seemingly, the operations may not follow a strict ordering in the firing sequence.

1.3 Motivation and Objectives

In a dynamic FMS environment, production managers are usually confronted with constantly changing scheduling scenarios in their day-to-day activities on the shop floor. Different scenarios may arise as a result of changes in the production mix, product types, due dates, part shortages and unanticipated events like machine failure. The availability of a number of optimization algorithms can allow production managers make better decisions considered acceptable for each scenario. However, not all existing algorithms can be suited to all kinds of scheduling problems that arise on the shop floor. While it is possible to adapt an algorithm to different production scheduling scenarios, it may turn out to be inefficient for some. Putting different algorithms at the disposal of the production managers given the situations they are best suited for, may go a long way to aid their decision making. Therefore, it is important to provide the decision makers with a platform that supports a neutral FMS representation in which the different optimization algorithms could be automatically tested to select the best solution reached or the best algorithm suitable to solve the given problem at hand. However, there is a lack of decision support tools based on PNAIHES that can afford the aforementioned concept.

One of the advantages presented by the PNAIHES approach is that, different search algorithms can be implemented in order to evaluate the best schedule of a particular manufacturing scenario. Several heuristic search methods have been developed for PNAIHES [26, 62–64, 69, 70]. However, it is quite difficult to evaluate and benchmark the efficiency of these algorithms in terms of solution quality and time due to the different computing platforms, programming languages and data structures used.

Although it is easier to perform TCPN-based simulations since there are readily available graphical tools for simulating TCPN models, not much work has been done in its combination with AI-based heuristic search methods. One of the reasons may be due to the difficulty that can be encountered with handling the complex data structure for representation (syntax expressions), storage and transition firing [27]. Apart from simulation limitations, existing tools have some

drawbacks regarding the internal execution of TCPN models. They use the eagerness-to-fire property for transition firings based on the global clock (model time) synchronization [71]. This concept limits the high level of concurrency exhibited in asynchronous systems like FMS, and thus, precludes the generation of firing sequences that would lead to an optimal schedule. The occurrence of an event should not be restricted by time constraints.

Searching for optimal schedules under the PNAIHES approach is quite a challenging task due to the NP-hard nature of FMS scheduling problems. The construction of the reachability graph requires the storage of all the encountered states to prevent duplicate search effort, and to obtain the solution path from the initial state to the goal state. The latter requirement can be avoided by attaching the path information to states as they are generated. On the other hand, the duplicate search effort called duplicate detection is central to the performance of reachability graph algorithms as redundant paths can be eliminated from the search space. It determines whether newly generated states have been previously encountered in order to avoid revisiting an already stored state. This is used to guarantee the termination of the state space exploration by preventing cycles.

However, the search can be very large, requiring a huge amount of memory to solve large-sized problems. This is practically impossible due to the memory limitations. To tackle the intractability, current research trend has focused on developing heuristic techniques by sacrificing optimality in favor of suboptimal solutions that can be obtained efficiently with the reduction of the search space and time. On the other hand, as pointed out in Kwok and Ahmad [72], Sinnen [73], optimal schedule is still an industrial target that can offer significant advantages for the following reasons: 1. In critical applications in which performance is the primary objective, 2. In situations where the same schedule is executed several times, and 3. To serve as a benchmark reference to test the performance and quality of different heuristics. Considering this relevance, the research question to be addressed is: how can we reduce the number of states to be stored so that larger problems can be solved without forgoing optimality?

Despite the fact that it is desirable to obtain the optimal solution for off-line scheduling, the decision making process considered as a non-added-value time [74], can naturally be short in time-constrained manufacturing environments, and for short term and on-line scheduling applications. Obtaining the optimal solution becomes intractable and impractical given the limited amount of time required for decision making. Solutions must be obtained in relatively short computation time especially when unexpected events occur. In these situations, near-optimal schedules are considered to be more appropriate to avoid unnecessary delays in the manufacturing process.

Solving FMS scheduling problems with classical heuristic search methods usually requires a great deal of search effort and time before the first and optimal solution is found. As a result, their applications are limited to small and medium sized problems. Also, the existing PNAIHES algorithms terminate the search as soon as the first solution is found [64, 69, 70]. However, there is no guarantee on the quality of the solution produced and how close it is to the optimal solution.

The overall objective of this thesis is to establish a TCPN-based scheduling framework for modeling and maximizing the performance of FMSs through the development of tools and efficient search methods based on the reachability graph analysis. To achieve this goal, the following objectives were identified:

1. To provide a comprehensive survey of the existing heuristic search methods for the PNAIHES approach in scheduling FMS along with a classification of each search method based on the space-time tradeoff criterion.

2. To identify and explore the open research areas from previously published research findings.

3. To develop a platform that allows the modeling of different scheduling problems irrespective of the FMS description.
4. To develop a tool that is capable of integrating several heuristic search methods for benchmarking and comparison purposes.
5. To tackle the effects of increasing problem size of FMS scheduling problems on the state space explosion.
6. To develop efficient search algorithms suitable for off-line scheduling, on-line scheduling or a hybrid of both schemes (scheduling/rescheduling approach) in order to produce optimal or near-optimal schedules.
7. To cover the different classes of FMS scheduling problems ranging from machine-part scheduling, deadlock-free scheduling to simultaneous scheduling of AGV-served FMS including the conflict-free routing problem.

1.4 Contributions

Although the PN-based scheduling methodology is well studied, there exist open research areas that are yet to be explored. Significant steps have been made to advance the existing body of knowledge on PNAIHES approach by contributing new research findings. The main contributions through the publications are detailed as follows:

1. **TIMed State Space Performance Analysis Tool - TIMSPAT**

   Current existing software lacks the capability to support search optimization based on the PNAIHES approach for TCPN models. While several heuristic search algorithms have been proposed for PNAIHES albeit using only TPN, the practical application of these techniques requires appropriate tool to facilitate the development and analysis of TCPN models for optimization purposes. However, there is currently no tool supporting the optimization of TCPN models for scheduling purposes, and comparing and benchmarking existing algorithms.

   In this light, **Paper I** [75] proposes an automated decision support and special purpose tool called TIMSPAT based on the PNAIHES approach. Thanks to the common data structure of AI-based heuristic search methods, it is capable of incorporating several heuristic search algorithms in a single executable tool. So far, nine algorithms have been implemented, which includes the proposed search algorithms from this thesis and those by other authors: A* [76], breadth-first iterative deepening A* (BFIDA*) [77, 78], hybrid heuristic search algorithms; Beam A* [63], A*-BT [70], dynamic window search [62, 79], BFIDA*-SLDD [80], and anytime algorithms; anytime column adaptive search-TCPN [81, 82], anytime layered search [78], and depth-first branch and bound (DFBnB) [83–85].

2. **Description language for TCPN - TIMSPATLib**

   TIMSPAT includes a description language called TIMSPATLib, for TCPN model development. TIMSPATLib is a C++ syntax library that allows the TCPN structure to be specified in a textual format. The TIMSPATLib syntax combines the standard token expressions of the TCPN formalism with mathematical and function expressions based on C++.

3. **Scaling up classical heuristic search algorithms to larger problem sizes for off-line scheduling application**

   Paper IV [80] proposes a new space-efficient approach to alleviate the scalability problem that appears in the state space exploration of FMS scheduling problems. It addresses the major limiting factor of reachability graph analysis using the concept of duplicate detection. This concept exploits a certain characteristic in the state space behavior of FMS scheduling problems called structural equivalence, to reduce the memory requirements. The equivalence is understood to be a consequence of the repetitive patterns inherent to FMS scheduling problems when the problem size is scaled. Since the state explosion problem is caused by an increase in the problem size and the state space of smaller problem sizes can be explored, it is worth understanding if the state space behavior of smaller problem sizes can be scaled to a larger size that is unsolvable.

   The structural state space equivalence is based on the assumption that a certain structural property should hold for a scalable FMS scheduling problem of any given size N above a certain size N_0. If this assumption holds, one can infer the behavior of a larger problem size from a smaller one (set of instances) by studying the behavior of a few lot of smaller size instances. This concept is then integrated into classical heuristic search methods so that larger problem sizes can be handled without sacrificing optimality.

4. **Space/time-efficient heuristic search algorithms**

   Papers II [81] and **III** [78] present two anytime heuristic search algorithms developed to overcome the drawbacks of existing PNAIHES algorithms. An anytime algorithm trades off computation time and solution quality. It is capable of finding suboptimal solutions very quickly and continuously improves the solution quality until the solution converges to the optimal solution. If given enough computation time, the algorithm will eventually obtain the optimal solution. Also, it is guaranteed to return a solution when interrupted. This method has been proved successful in artificial intelligence community. However, they are yet to be explored in the PN research community.

   Paper II [81] adapts and improves an existing anytime algorithm to TCPN-based scheduling, while **Paper III** [78] propose a new algorithm that combines two heuristic search algorithms making them anytime for deadlock-free scheduling. The proposed algorithms are suitable for both off-line and on-line scheduling purposes due to their effectiveness in adapting to different CPU constraints. Also, they can be used in a scheduling/rescheduling mode whenever a change in the operating conditions of the system requires a change in the previously obtained schedule [86].

5. **Simultaneous scheduling of machines and AGVs with conflict-free routing**

   The overall scheduling can be so complex that it cannot be handled in an integrated manner. The scheduling complexity increases with the integration of AGV scheduling and routing. **Paper V** [87] presents a TCPN-based approach to the simultaneous scheduling of machines and AGVs. Unlike the existing approaches that employ a decomposition framework, the entire scheduling problem is described in a single model, and the algorithm proposed in **Paper III** [78] is used to find optimal or near-optimal solutions to the problem. The paper proposes and evaluates two simultaneous scheduling models using an event-driven vehicle assignment solution as opposed to the traditional dispatching rules.

1.5 Extension of Scope

The primary objective of this thesis is focused on FMS scheduling problems. However, the tool (formerly called RADIUS) and algorithms presented in this research work have been successfully applied in other areas like air traffic management [88–92], and other discrete event systems such as concurrent and distributed systems [93].

1.6 Thesis Organization

This thesis has been structured in such a way that the reader understands the context and can duly follow the evolution of each method and its dependency in other papers. It is separated by publication, and each chapter contains a full text version of a paper, with the exception of **Paper I** [75]. **Paper I** has been expanded into three chapters; Chapters 2, 3, and 9. Chapter 2 gives the background on TCPN modeling and TIMSPAT. Chapter 3 presents the state-of-the-art review on PNAIHES approach, while Chapter 9 provides a benchmarking and comparative study of the nine heuristic search algorithms implemented in TIMSPAT. The benchmark is performed on a case study of a real flexible manufacturing cell of an eyeglass production system. Chapters 4, 5, 6, 7, and 8 present **Papers II, III, IV, V**, and an extension of **Paper V** to simultaneous scheduling of machines and AGVs with conflict-free routing respectively.

2

Background: TCPN Modeling and TIMSPAT

2.1 State of the Art Review on PN-based Tools

There are quite a handful of graphical and command line PN simulators available for modeling and simulating discrete event systems. The PN tool database [94] provides a comprehensive list of the existing tools for PN modeling. The database consists of both general-purpose (mostly) and special-purpose tools. Due to the large number, it is quite difficult to evaluate each tool. Four search criteria are used to filter the tools that closely match the requirements for the performance analysis of TCPN models. The following keywords provided by the database search tool were used: *State spaces*, *Petri nets with time*, *high level Petri nets*, *simple performance analysis*. Of the 85 tools registered in the database, only 32 implement state space analysis. Among these 32, 13 support timed nets, 6 of the 13 support TCPN while 4 implement a kind of performance analysis technique. As the requirements get stronger, the list keeps reducing. Most of the tools are developed as graphical editors (66) that implements token game animation (46). Only four tools passed the filter: CPN Tools [55], INA, JFern, Petruchio. However, none of these tools have directly supported the TCPNAIHES approach.

Although, they do not fulfill the requirements, it is worth mentioning those tools that have been used for manufacturing systems. PNetLab [95] and SimQPN [96] are used for the control and scheduling of manufacturing systems, and queueing systems respectively. Also, PN Toolbox [97, 98] has been used for manufacturing applications. Tools like GPenSIM and PN Toolbox are embedded into third-party commercial software packages like MATLAB. Users are required to learn the MATLAB language to develop models. While PNetLab is standalone, it generates a simulator executable each time a new model needs to be run. Some of the PN simulators (SimQPN, GPenSIM [99]) claim to support CPN without including some of the important features of the advanced CPN formalism [47]. Unlike PN, the complex data structure of CPN makes it difficult to find tools that support model development using CPN. It even gets more difficult with TCPN modeling.

In spite of these drawbacks, CPN Tools stands out as an industrial strength tool that provides both a graphical editing interface and an interactive graphical simulator for constructing and analyzing models. CPN Tools has been used to analyze some FMS scheduling problems in the literature. Aized [100], Aized [101] and He and Wu [102] use CPN Tools to model and analyze the performance of an integrated automated guided vehicle system, multiple cluster tools system with random failures, and the deadlock-free scheduling of cluster tools, respectively. However, the literature reports the usage of the platform for simulation purposes only.

The disadvantages of using simulation have been highlighted in Chapter 1.2.2. Furthermore, it is quite important to identify its weakness with respect to deadlock-free scheduling. In practice, most FMS are deadlock-prone since it is usually difficult to manage the allocation of resources due to inherent resource limitations like no buffer storage (blocking), limited buffer capacity, and material handling systems availability. The concurrent competition for a finite number of resources in a manufacturing environment usually results in deadlock situations, which brings the system to a permanent halt state. The PNAIHES approach removes the overhead to guarantee the liveness (deadlock-freeness) of the PN [41] before scheduling is performed. As such, deadlock control policies are not a necessity to guarantee an optimal deadlock-free schedule [78](**Paper III**). Provided the system is accurately described by the TCPN model (precedence constraints, resource sharing and constraints), the defined goal state guarantees that a deadlock-free schedule is obtained without uncertainty. The schedule generated from the firing sequence from the initial state to the goal state, ensures that deadlock is avoided in the operation of the FMS if the schedule is followed.

On the other hand, deadlock control policies are a requirement for simulation. Since it considers one scenario at a time, it stops evolving when a deadlock is reached. Simulation may fail to return a feasible schedule if most of the scenarios considered lead to deadlock. Also, the downside of the control policy integration with scheduling is that they are system specific and do not guarantee optimality since they impose restrictions on the system evolution. This topic is well treated in [78](**Paper III**). As a result, it is quite difficult to simulate deadlock-prone FMS using CPN Tools. As clearly seen in the CPN Tools deadlock-free scheduling application described in [102], a deadlock control policy is required.

Notwithstanding, CPN Tools has a state space analysis plug-in, but it has several limitations to support the timed state space exploration of TCPN models [71]. It uses the eagerness-to-fire property based on the global clock synchronization that may preclude the generation of firing sequences that would lead to an optimal schedule (Section 2.5). In addition, the absence of efficient search algorithms has limited its applicability. Only basic traversal algorithm like BFS is implemented in the tool, which cannot scale up to industrial-sized problems. Also, CPN Tools offers no support to integrate heuristic search methods.

Due to the limitations of the state space analysis tool, an extensible platform to CPN Tools called ASAP [103] was developed to provide an implementation support for a wide range of advanced state space methods for industrial-sized CPN models. ASAP was integrated into CPN Tools from Version 3.0. Yet, the state space methods developed in ASAP are primarily aimed at untimed CPN models for model checking and verification of behavioral properties. Also, they are limited to memory-efficient search methods without the use of heuristic functions since model checking does not require optimality.

Even with this extension, it is still difficult to integrate one's search algorithm. Both tools (CPN Tools and ASAP) rely heavily on Standard Markup Language (SML), a proprietary functional programming language with a high learning curve. The steep learning curve of functional programming makes it hard to use the platform to develop algorithms without having gotten a full grasp of the language. Also, the syntax for complex mathematical expressions requires SML knowledge. Unlike CPN Tools, the expressions used in TIMSPAT follow a more direct and natural syntax format.

To overcome these shortcomings, TIMSPAT has the following distinguished features from the existing ones:

- It is a dedicated standalone tool for PNAIHES approach based on reachability graph analysis, written in C++.
- It implements an event-driven scheduling solution that overcomes the shortcomings of the global clock synchronization for optimization.
- It offers a localized enabling of transitions. Each transition structure is created as an object and only the places required for enabling the transition are specified.
- Easy-to-write syntax expressions without the need to learn a programming language. Complex mathematical expressions are supported in a plain language format. The syntax combines the standard token expression for CPN and the strength of C++ syntax for mathematical and customized function expressions.
- It allows the implementation of different heuristic search algorithms using a common data structure.
- It offers a portable and standalone modeling library that can be easily integrated with other applications or used by anyone willing to implement its own search algorithm.

- It is not designed for graphical modeling. With a host of graphical editors and simulation tools, an editor is not critical to TIMSPAT development. A specialized performance analysis tool is thus paramount. The CPN structure is read from ASCII files and a syntax analyzer is used for checking the correctness of the model prior to execution.

2.2 TIMSPAT Architecture

The architecture of the TIMSPAT is shown in Fig. 2.1. The TCPN model serves as the input to the tool. Its structure is emulated using ASCII files whose definitions conform to the TIMSPAT modeling library (TIMSPATLib) syntax language. The solutions to the TCPN model are generated by the search algorithms via the state space storage. The main components of TIMSPAT are:

- Syntax checker: It validates the specification of the definition files to ensure that it is consistent with the TIMSPATLib syntax instructions. The checker reports errors encountered in files to the user and the files are only passed to the simulator if they have been certified okay.
- TIMSPATLib: It is a non-graphical modeling library designed for the specification of TCPN models in a textual format provided in ASCII files using easy-to-learn syntax instructions. TIMSPATLib interprets and stores the structure of the model in memory as specified in the definition files.
- Simulator: It performs the discrete event simulation of TCPN models. However, its execution is synchronized with the search algorithm module. The simulator uses the stored TIMSPATLib model information to evaluate the states (or markings) of the system required by the search algorithm. It interfaces with both the TIMSPATLib and the search algorithm module on a continual basis until termination.
- Search algorithm: It is used to construct the state space according to a defined objective function. The search algorithm drives the exploration of the state space toward a near-optimal or optimal solution. Solutions are generated from the state space as a sequence of transition firings when a goal marking is reached. They are generated either continuously (when improved solutions are obtained) or at termination depending on the search algorithm employed. It is mainly dominated by AI-based heuristic search methods.

2.3 TIMSPATLib for TCPN Modeling

A CPN is a directed bipartite graph with two node types called places and transitions where the nodes are connected via directed arcs. It extends the classical PN with the use of a data value called colored token. A place can contain tokens and is used to describe resources in the system while a token consists of one or more colors describing the entity attributes. Each token can carry a weight called cardinality. A transition describes the event (the start or completion) that may occur (or fire) based on the preconditions of input arc expressions and guards. Graphically, places, transitions, arcs, and guards are represented by circles, boxes, arrows, and square brackets.

For performance evaluation purposes, TCPN modeling is considered. A TCPN has a time notion expressed by the introduction of a global clock. The global clock represents the model time and each token carries a time attribute called time stamp. The time stamp describes the earliest time at

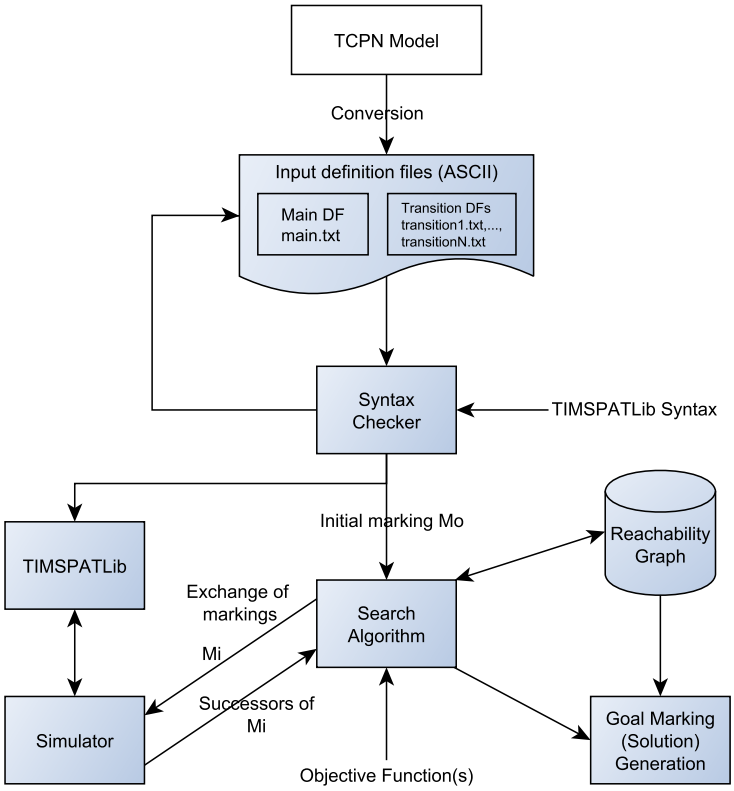


Fig. 2.1. TIMSPAT architecture.

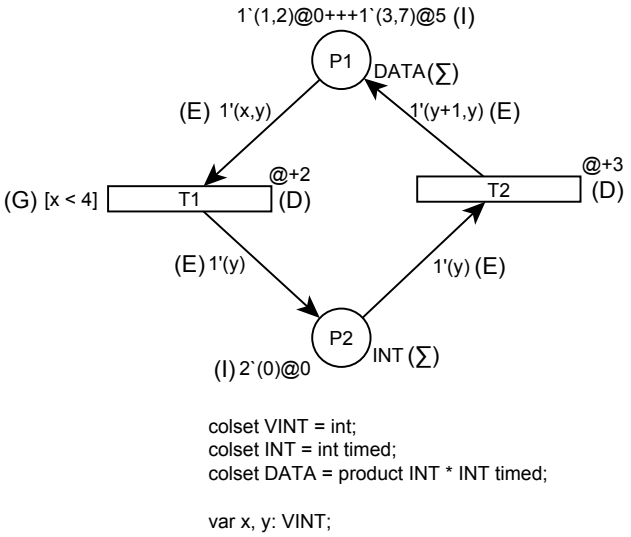


Fig. 2.2. A TCPN model developed in CPN Tools.

which a token becomes available. With TCPN, one can model durations, delays, deadlines, etc. and optimize the performance of a system. We assume the reader is familiar with TCPN formulations and theory [47]. Figure 2.2 gives an example of a simple TCPN using the CPN Tools formalism syntax.

In TIMSPATLib, the specification follows the standard definition of a TCPN [47] with slight modifications on color set labeling, variables and transition delay, including the goal marking syntax.

Definition 2.1. *Formally, a $TCPN_{TIMS}$ for TIMSPAT is defined as $TCPN_{TIMS} = (TCPN, M_0, M_g)$ where $TCPN$ can be defined as a 11-tuple, $TCPN = (P, T, A, \sum, V, C, G, E, I, F, D)$ where:*

1. *P is a finite set of places $\{p_1, p_2, \ldots, p_m\}$.*
2. *T is a finite set of transitions $\{t_1, t_2, \ldots, t_n\}$ such that $P \cap T = \emptyset$.*
3. *A is a finite set of directed arcs $\{a_1, a_2, \ldots, a_k\}$ such that $A \subseteq P \times T \cup T \times P$.*
4. *$\sum$ is a finite set of nonempty types called colored sets that defines the number of token elements (colors) and the operations and functions that can be used in the net inscriptions (i.e. arc and initialization expressions). Each color set is either timed or untimed and an untimed set is either static or otherwise.*
5. *V is a finite set of variables of numeric data types (integer or real).*
6. *$C : P \rightarrow \sum$ is a color set function that assigns a color set to each place. A place p is timed if $C(p)$ is timed; otherwise, p is untimed or untimed static.*
7. *$G : T \rightarrow EXPR_v$ is a guard function that assigns a guard to each transition t such that $Type[G(t)] = true$ or $false$.*
8. *$E : A \rightarrow EXPR_v$ is an arc expression function that assigns an arc expression to each arc a such that $Type[E(a)] = C(p)_{MS}$ if p is untimed or untimed static and $Type[E(a)] = C(p)_{TMS}$ if p is timed, where p is the place connected to the arc and C_{MS} denotes the set of all multisets over C.*
9. *$I : P \rightarrow EXPR_\emptyset$ is an initialization function that assigns an initialization expression to each place p such that $Type[I(p)] = C(p)_{MS}$ if p is untimed or untimed static and $Type[I(p)] = C(p)_{TMS}$ if p is timed.*
10. *$F : P \rightarrow EXPR_\emptyset$ is a finalization function that assigns a finalization expression to each place p such that $F(p) = C(p)_{MS}$.*
11. *$D : T \rightarrow \mathbb{R}_0^+$ is a time-delay function associated with each transition $t \in T$. It describes the set of firing durations (transition delays). $\mathbb{R}_0^+$ denotes the set of all positive real numbers including zero.*

M_0 is the initial timed marking defined by $M_0(p) = I(p)\langle\rangle \ \forall p \in P$
M_g is the final or goal untimed marking defined by $M_g(p) = F(p)\langle\rangle \ \forall p \in P$

As in CPN Tools, $EXPR$ denotes the set of expressions provided by the TIMSPAT library, and $Type[e]$ denote the type of an expression $e \in EXPR$, i.e., the type of the values obtained when evaluating e. The set of free variables in an expression e is denoted $Var[e]$. A free variable is a variable which is not bound in the local environment of the expression [47].

Note that in this definition, variables do not belong to $\sum$ since variable types (integer and real) are handled by default in the library. As such, variable definitions are not needed. Only the color set description for each place is required.

The current state of the system is defined by the distribution of tokens over the places called marking. An untimed marking M_u maps each place into a multiset of tokens $M(p) \in C(p)_{MS}$ [47]. TIMSPATLib adopts the functional token expression of CPN Tools.

Definition 2.2. *A multiset (MS) m over a non-empty set $S = \{s_1, s_2, s_3, \ldots\}$ is a function $m : S \to \mathbb{R}^+$ that maps each element $s \in S$ into a non-negative integer $m(s) \in \mathbb{R}^+$, . It is written as a sum using a single $+$ and $'$:*

$$\sum_{s \in S} m(s)'s = m(s_1)'s_1 + m(s_2)'s_2 + m(s_3)'s_3 + \ldots \tag{2.1}$$

The non-negative integer $m(s)$ is the number of appearances of the element s in the multiset m. $m(s)$ is also called the cardinality (weight) of the token s.

An element s is a member of a multiset m if the number of appearances $m(s)$ of s in m is greater than zero, i.e., if $m(s) > 0$. The size of a multiset $|m|$ is the sum of the number of appearances of the elements in m, the number of tokens in a place p.

The other operations like addition, scalar multiplication, comparison and subtraction are defined in [47]. The set of all multisets over S is denoted as S_{MS}. The empty multiset over a set S, $\emptyset_{MS}$ is defined as $\emptyset_{MS}(s) = 0$, $\forall s \in S$.

The multiset of tokens in an untimed place is constructed using the single sum operator $+$ rather than the double $++$ in CPN Tools. A timed place attaches a time stamp to each token. For scheduling purposes (see Section 2.5), the time stamps of tokens in a timed place are also constructed as a multiset using the single sum operator $+$ and the $@$ symbol. Here, the timed multiset is not used.

The multiset of token time stamps is expressed as:

$$\sum_{s \in S} m(s)@tm[ts] = m(s_1)@tm[ts_1] + m(s_2)@tm[ts_2] + m(s_3)@tm[ts_3] + \ldots \tag{2.2}$$

where $tm[ts]$ is the ordered time stamp list $tm[ts] = [ts_1, ts_2, \ldots, ts_{tm(ts)}]$ and $tm[ts_1] = [ts_1{}^1, ts_2{}^1, \ldots, ts_{tm(ts_1)}{}^1]$. It contains the time values $ts \in TS$ for which $m(s) \neq 0$.The $@$ symbol is omitted if $m(s) = 1$.

As such, the description of tokens in a place has two parts: the set of tokens and the set of time stamps. For example, using CPN Tools, the tokens in a timed place represented by $2`(4,3)@5, 6 + + + 1`(2,3)@3 + + + 3`(5,5)@0$ are expressed in TIMSPATLib as:

$$2'(4,3) + 1'(2,3) + 3'(5,5);$$
$$5, 6 + 3 + 3@0;$$

A timed marking M is defined as a triple (M_u, TS, ts^*) which consists of the untimed marking M_u, the time set TS, a set of time values called time stamps, $TS = \mathbb{R}_0^+$ and $ts^* \in \mathbb{R}_0^+$, the value of the global clock. The initial timed marking M_0 represents the initial state of the system.

The $TCPN_{TIMS}$ definition given in Def. 2.1 considers timed transitions only, where transitions are associated with a delay $D(t)$ interpreted as the duration of the activity modeled in the event. A transition delay can correspond to machine processing or transportation time in a manufacturing environment. The delay uses the holding duration concept described in [71] for modeling the performance optimization of scheduling problems. In this concept, a timed transition is fired instantaneously but the output tokens will not be available for other transitions until the delay has elapsed. This makes the transition behave as an operation with start and release times. The current version of TIMSPATLib does not allow time delays to be specified for arcs.

The $TCPN_{TIMS}$ definition introduces an important feature called *static* place which can be very useful when evaluating the reachability graph of an FMS.

Definition 2.3. *A static place p_f is an untimed place with a static color set that does not change during the evolution of the system. The token colors are never affected by transition firing. For a place to be considered static in a TCPN, it must have two directed arcs (input and output) such that when connected to any transition in the TCPN, the input and output arc expression is the same i.e.* $\forall A(a_1, a_2)$ $E(a_1) = E(a_2)$ *where* $a_1 \in (p_f \times t)$ *and* $a_2 \in (t \times p_f)$, $t \in T$.

The standard CPN formalism allows one to add as many colors as required by the model for simulating a system, be it static or dynamic information. While this is suitable for simulation purposes, it seems impractical for state space construction particularly in the case of static data. These data are commonly found in the problem definition of most FMS. Examples are: the deterministic processing times, transportation times, AGV routing information, etc.

In the state space exploration of CPN models, all the information required to enable or fire the transitions must be kept in the marking. Unfortunately, static data become redundant since they are propagated from one marking to the other in the state space. The place information is carried over in the marking description and repeated in every reachable state in the state space. As a consequence, the state space can get blown up as quickly as possible, leading to a premature explosion. In TIMSPAT, the static place tokens are stored once in a fixed memory location.

The static place offers a kind of flexibility to users if the information is too large to be written as an *if-then-else* expression. The command *if-then-else* can be used in some cases, however, it cannot completely replace the static place. For example, it is difficult to model alternative routings with *if-then-else* when the firing of a transition is expected to produce more than one successor.

2.3.1 TIMSPATLIb TCPN Structure

The input definition files used to specify the $TCPN_{TIMS}$ structure consist of a main file MDF (main.txt) and a set of N transition files TDF (transition1.txt,...,transitionN.txt). Each transition in the $TCPN_{TIMS}$ is written into a separate file. The MDF specifies the initial marking M_0 (M_u and TS), the color set, the goal marking definition M_g, the information required for heuristic evaluation (shared with the search algorithm module), constants (optional), and functions (optional) used in the transition files. Each TDF list the arc expressions of places (both input and output where applicable) acting on a transition, guard expression and time delay. TIMSPATLib assumes the following for TCPN modeling:

1. Places and transition names are numerically labeled and must be sequential.

2. Color variables are limited to only two numeric types: integer (including large integer) and real data types. As such, neither color variable and token type declaration nor initialization is required by the user. The real type is fixed at a maximum of 3 decimal places using a built-in function $radiusdp()$ to differentiate an integer computation from a real one. The limitation of the types is aimed at minimizing the memory usage of markings in the state space graph. Complex data types like strings, Boolean, list, etc. are not suitable to optimize the marking storage. These variables can be expressed numerically. For instance, if a token color must take the string values "heavy", "medium", and "light", the three values can be represented as 1, 2 and 3 respectively. Same applies to Boolean colors.

3. The construction of a color set need not be explicitly defined, the library only accepts an *n-tuple*, a *product* color set of integer and/or real where $n \geq 2$ or a *simple* integer or real where $n = 1$. It is only necessary to define n, a non-negative integer which describes the

Table 2.1. TIMSPATLIb operators.

Operator	Meaning	Operator	Meaning
$=$	assignment	$>$	greater than
$\&\&$	logical AND	$<$	less than
$\|\|$	logical OR	$+$	addition
$?:$	if then else C++ syntax	$-$	subtraction
$<=$	less or equal	$/$	division
$>=$	greater or equal	$\hat{}$	power
$!=$	not equal	$()$	parenthesis
$==$	equal	$\#$	empty set for place in M_g definition
$*$	multiplication for TDF or any color value or cardinality in M_g definition for MDF	%	guard separator or arc expression multiset operator

number of colors that will reside in a place. List, union and enumeration color sets are not supported for state space analysis.

4. The tokens used for input arc expressions are limited to explicit definition of color variables only. TIMSPATLib does not allow numeric values, conditional expressions or computations to be specified for input arc expressions. Also, the color variables must be unique on all input arcs. Numeric values or equivalent colors intended to be used on input arc expressions can be expressed as guards. This assumption allows quick evaluation of transition bindings [47].

5. Color variables are local to a transition file and can be reused for other token colors of place in the other files without a prior declaration.

2.3.2 TIMSPATLIb Syntax for Operators and Functions

The TIMSPATLIb operators are already well known and used in object-oriented programming languages like C++. The usage of these operators is practically the same, but there exists a minor difference in the usage of some operators. The supported operators are shown in Table 2.1.

Also, TIMSPATLib supports mathematical functions with single and variable number of arguments. The single argument functions include: sin, cos, tan, asin, acos, atan, sinh, cosh, tanh, asinh, acosh, atanh, log2, log10, log, ln, exp, sqrt, sign, rint, abs, floor, and radiusdp. See [89] for the usage of some of the mathematical functions. The list of some variable argument functions are: min, max, sum, and avg. Additional mathematical functions not listed above can be added on request. Like CPN Tools, the term *empty* is used in arc or conditional arc expressions to avoid the addition of tokens to an output place.

2.3.3 TIMSPATLib Syntax for Modeling Objects

The syntax format for tokens in a marking has been explained in Section 2.3. The color set description for a place (in the main file) is described as:

CS colorset name = number of colors, timed/untimed,static;

A place identifier is represented as X, where X is a sequential number starting from 1. Arbitrary place numbers and text-based identifiers are not permitted. For example, an initial marking with three places is written as: $1\ 2'(4,6) + 5'(1,3); 2\ ; 3\ 1'(1,1,2);$ for M_u and $2@0 + 5@0;\ ; 0;$ for TS. There are 7 tokens in place p_1, place p_2 is empty, and p_3 has only one token.

The goal marking definition is preceded by a prefix **EF** followed by the untimed marking description for each place separated by semicolon. For simplicity and computation time considerations, we adopt a single token description for each place by grouping multiple tokens into one. A wild card $*$ is used to identify any color or cardinality value or when the actual value in the goal marking is not known a priori or indeterminate. The symbol # is used to specify an empty place. The main idea behind this format is that not all token colors are relevant in detecting a goal marking. It suffices to specify only the necessary token colors. For example, the goal marking for Fig. 2.2 can be described as:

$$\textbf{EF}\ 2'(*,5); \#;$$

This means that p_1 must have 2 tokens with the second color value of each token equals 5, and p_2 must be empty. An arc expression is preceded by alphanumeric characters. It takes the form:

$$\text{FA}\textbf{XY}\ tc_1\textbf{\%}tc_2\textbf{\%}\ldots\textbf{\%}tc_n;$$

where **X** represents the arc type, **E** for input, **S** for output, and **Y** represents the place identifier, and **%** multiset summation symbol. For example, an input arc expression from a place p_3 with two tokens to a transition is represented as: FAE3 $1'(x,y,z)\%1'(a,b,c);$, while an output arc expression on p_4 is given as: FAS4 $1'(u,v,w);$

A guard expression is an optional one-line expression in which each subset of the guard is joined by the guard separator operator $\%$. It goes by the syntax:

$$\textbf{GU}\ expr_1\ \%\ expr_2\ \%\ \ldots\ \%\ expr_n;$$

Example: **GU** $x == y\ \%\ ((y > x + z)||(y > z))\ \%\ x + y <= z + 8;$

Customized functions are written with the format:

FU function name(variables separated by comma):expression;

Example: **FU** $addif(x,y):\ x > y\ ?\ x + y : x;$. $addif$ is the function name, and the variables used in the expression are x and y. Functions are defined in the main file (main.txt).

A constant is expressed as:

$$\textbf{CT}\ constantname = constantvalue;$$

Example: **CT** $pi = 3.142;$. A transition delay is specified using:

$$\text{FAS}\textbf{D}\ delayexpression;$$

The delay expression can either take a numeric value or a function. Fig. 2.3 shows the equivalent TIMSPATLib syntax for the sample TCPN model in Fig. 2.2. More details on the syntax and examples can be found on the TIMSPAT's website `http://grupsderecerca.uab.cat/timspat/`. An online version of the tool is available for testing by registering on the website.

main.txt
$1\ 1'(1,2)+1'(3,7);\ 2\ 2'(0);$*initial M_u*/
$0+5;\ 2@0;$ /*initial time stamp set*/
CS DATA;INT; /*place color set */
EF $2'(5,*);\ 2'(*);$ /*goal marking*/

%COLOR SET
CS INT=1,timed;
CS DATA=2,timed;

transition1.txt transition2.txt
FAE1 $1'(x,y);$ FAS1 $1'(z+1,z);$

FAS2 $1'(z);$ FAE2 $1'(z);$

GU $z<4\ \%\ y==z;$FASD 3;

FASD 2;

Fig. 2.3. Equivalent syntax expressions in TIMSPATLib for Fig. 2.2.

2.4 Simulator – Execution of a TCPN

The execution of $TCPN_{TIMS}$ is controlled by the simulator module. It involves the enabling and firing of transitions according to the preconditions (guards) and estimated duration. Also, it includes goal marking check.

The simulator uses the TIMSPAT's model structure to evaluate markings using an event-driven approach [71, 81] for the generation of successors (See Section 2.5). Each time it receives a marking from the search algorithm, it checks whether or not the transitions of the TCPN can be enabled given the guard conditions and information from the TIMSPATLib. Once the enabled markings are fired, the simulator determines which marking has reached the goal given the TIMSPATLib M_g syntax. It then sends the reachable markings as successors with a goal marking header to the search algorithm for further evaluation. The interaction between the three TIMSPAT components is based on a continuous evaluation of markings until the search algorithm terminates the exploration. The steps of the simulator algorithm are given as follows:

1. Get a new marking from the search algorithm module.
2. For each transition in the TCPN:
   (a) Preprocessing: check if the number of tokens in each place can be reduced by evaluating the guards with at most two variables.
   (b) If one of the input places is empty or the number of tokens is less than the cardinality or the multiset of tokens in the input arc expression, exit.
   (c) Generate all the possible combinations of tokens (subsets) for all the input places.
   (d) For each token combination subset:
      i. Bind the colors of each token to their variables.

ii. Check to ascertain whether or not it can enable the transition by evaluating the entire guard expression.

iii. If the guard evaluates to true, go to Step 2e, else go to Step 2d.

(e) Fire the transition with the binding:

i. Remove the tokens from the input places of the marking and store their timestamps if color set of the place is timed.

ii. Calculate the enabling time tau_k by taking the maximum of all token time stamps in the enabled token subset.

iii. Calculate the firing time by adding the transition delay d to tau_k, $(tk + d)$.

iv. Generate a new marking by adding new tokens to the output places by evaluating the output arc expressions and attaching the computed time stamp (firing time) to each token in a timed place.

v. If the new marking is a goal marking, mark as goal.

3. Send the computed successors of each transition if applicable to the HS module.

The enabling of transitions in a CPN is usually quite expensive [104]. It has been a subject of much research in [47, 105–108]. The simulator must first compute the set of all possible bindings B for a transition t, denoted as $B(t)$ [47]. A binding b of a transition t, $b \in B(t)$ assigns a value $b(v)$ to each variable v of the transition t. It binds the tokens in the input places of transitions to the input arc expressions and guards. The variables of a transition t, denoted as $Var(t) \in V$, consist of the free variables specified in the guard and in any of the arc expressions of any arcs connected to the transition t. For example, the variables of transition t_1 in Fig. 2.2 is $Var(t_1) = \{x, y\}$. A transition is enabled if the input places contain the multiset of tokens specified and the guard of the binding $G(t)\langle b\rangle$ is true.

It is easier to compute bindings for simulation purposes since only a subset is required and in a situation where the model time is used to drive the state space exploration for TCPN models as in CPN Tools (See Section 2.5). To reduce the combinatorial effects of transition bindings in TIMSPATLib, we adopt the following rules: 1. The token multiset of input arc expressions must be limited to two for input places with a large number of tokens, 2. Before evaluating the bindings, the simulator first removes ineligible tokens from the input places with guards having at most two variables. This is done to reduce the number of tokens to use in the combinatorial process, and 3. When a static place is used, there must be sufficient guard conditions to trim down the number of tokens in the place. If this is not possible, an *if-then-else* operator is recommended.

An enabled transition may fire. Firing means that the tokens are removed from the input places and added to the output places of the firing transitions. In a TCPN, a transition t is time-enabled at time τ_k in a marking M denoted by $M[t\rangle_{\tau_k}$ if all the tokens to be consumed from the input places have a time stamp not later than time τ_k [109]. If a transition t fires at time τ_k, it changes M to a new marking M' denoted by $M[t\rangle_{\tau_k}M'$. M' is said to be reachable from M. In $TCPN_{TIMS}$, a transition delay applies to all output tokens created at transition firing. The time stamp of a token is defined at its generation time. Firing a transition t at time τ_k with a delay d, time stamps the output tokens with the time value $\tau_k + d$.

2.5 Timed State Space Exploration

The performance analysis of TCPN involves the generation of a timed state space (TSS) and the traversal of the state space with a search algorithm. A TSS can be defined as a reachability set $R(TCPN, M_0)$ that comprises the set of all possible markings reachable from M_0 which minimizes a given objective function. The TSS is represented as a directed graph $TSS = (N, E, M_0)$ where N is the set of nodes and E is the set of directed edges $E = \{(M, t, M')_{\tau_k} \in N \times (T \times \mathbb{R}) \times N | M[t\rangle_{\tau_k} M'\}$. A node contains a reachable marking M including the parent identifier and any other information required by the search algorithm. A marking $M' \in V$ is a successor of (or reachable from) marking $M \in V$ if $(M, t, M')_{\tau_k} \in E$. The edges represent the transition bindings used to generate the successor marking. Expanding a marking involves the computation of its successors. A visited marking M is a marking that has been expanded. A path between two markings M_0 and M_n is a sequence of markings $\sigma = M_0[t_0\rangle, M_1[t_1\rangle, \ldots, M_{n-1}[t_{n-1}\rangle, M_n$ connected by a sequence of edges with enabling times such that $\forall i \in [0, n-1]$, $(M_i, t_i, M_{i+1}) \in E$.

Lakos and Petrucci [110] identify two different approaches to TSS generation based on their firing rules: the conservative and the optimistic approach. The conservative approach called the reduced earliest time state space (RSS), the TSS generation method used by CPN Tools, uses the eagerness-to-fire property based on the global clock synchronization such that a transition is only allowed to fire if $\tau_k \leq \mathfrak{r}^*$. As a consequence, the firing of a transition t' enabled at $\tau'_k > \mathfrak{r}^*$ for a particular operation is delayed until the global clock advances to τ'_k or to a much later time than τ'_k depending on the prior firing sequences. When used for the optimization of inherent asynchronous systems like FMS which exhibit a high level of concurrency and parallelism [71], this property may preclude the generation of firing sequences that would lead to an optimal schedule. Since there are activities that can be performed concurrently, delaying the execution of the operation to a later time can have a negative impact on the overall system performance. RSS is defined as a tuple $RSS = (N, E, M_0)$ where $E = \{(M, t, M')_{\tau_k} \in N \times (T \times \mathbb{R}) \times N | M[t\rangle_{\tau_k} M', \nexists t', \tau_{k'} < \tau_k : M[t'\rangle_{\tau_{k'}}\}$

The optimistic approach called the earliest time state space (ESS) allows the firing of transitions as soon as they are enabled i.e. it includes the transition firings with $\tau_k > \mathfrak{r}^*$ in addition to those of the RSS without the global clock constraint. As a result, the firing of transitions no longer depends on the behavior of the global clock, hence, leading to an event-driven approach. ESS is a tuple $ESS = (N, E, M_0)$ where $E = \{(M, t, M')_{\tau_k} \in N \times (T \times \mathbb{R}) \times N | M[t\rangle_{\tau_k} M', \nexists \tau_{k'} < \tau_k : M[t\rangle_{\tau_{k'}}\}$. Piera and Music [71] investigated the use of the two approaches for FMS scheduling, highlighting the shortcomings of RSS for optimization.

ESS can be explored either in a classical manner, by evaluating both the untimed marking and time stamp together as one set for comparison in duplicate detection [80](**Paper IV**) or in a compact form, as a condensed state space (CSS) [78, 81, 111, 112] (**Papers II** and **III**). The CSS combine several markings into a single class using the notion of untimed marking equivalence. During the search process, it excludes the time stamps from duplicate marking detection to avoid exploring a large state space containing several markings with the same untimed marking but different time stamp set. Here, the time stamps are used for calculating the firing times of transitions and the creation times of new tokens, and to evaluate performance with respect to the objective function. TIMSPAT implements both the ESS and its condensed version (CESS) in the heuristic search algorithms. The CESS justifies the separation of the timed marking into untimed marking and time stamp set in the descriptor used in Section 2.3.

As an example, consider a system that consists of two jobs and two machines [113]. The jobs' requirements are given in Table 2.2. Operation $M_1(6)$ means the first operation of job J_1 must be performed on machine M_1 during 6 time units. The TCPN model is described in Fig. 2.4. Fig. 2.5

Table 2.2. Routing and processing times of jobs.

Job/Operation	1	2
J_1	$M_1(6)$	$M_2(7)$
J_2	$M_1(5)$	$M_2(10)$

shows the ESS graph of the instance constructed using BFS. For legibility, the @ symbol is used to represent the time stamps of the tokens in the graph. The graph has a total of 15 nodes (M_0 to M_{14}). Each edge shows the transition binding and the enabling time. The goal markings (M_{11} to M_{14}) described by **EF** $2'(*,0);2'(*)$; are found at the same level of the graph. The optimal solution is in bold. In contrast to the ESS, Fig. 2.6 gives the condensed version of the graph. Markings M_4 and M_5 in the ESS are collapsed into one as M_4 in the CESS. The same goes for M_6 to M_8 in Fig. 2.6. For large state space graphs, it is impractical to keep the time stamp set of all equivalent untimed markings in a class. An additional measure is required to select the most promising time stamp set to be used for exploration. The CSS procedure $CSS(g(M_{stored}), g(M'))$ described in [80, 81] (**Papers II** and **III**) is used in the CESS graph.

2.5.1 Heuristic Search Algorithms for TSS

A classical AI heuristic search algorithm like A* [65] can be used to construct the TSS of a TCPN. A* is a best-first search that searches through the TSS by systematically expanding the most promising marking one at a time, in order to find the shortest path from M_0 to M_g. It guarantees that the first solution obtained is optimal when all the markings with cost less than the optimal goal marking cost have been expanded. The search is guided by an evaluation function $f(M) = g(M) + h(M)$ that determines the cost of each marking in the search space. Cost function $g(M)$ is the actual cost to reach a marking M from M_0 and $h(M)$ is a heuristic function that estimates the remaining cost to reach M_g from M. A* guarantees that the search always finds an optimal solution if $h(M)$ is admissible i.e. it is a lower bound that does not overestimate the cost to goal, $h(M) \leq \mathrm{h}^*(M), \forall M$ where $\mathrm{h}^*(M)$ is the cost of the optimal path from M to M_g.

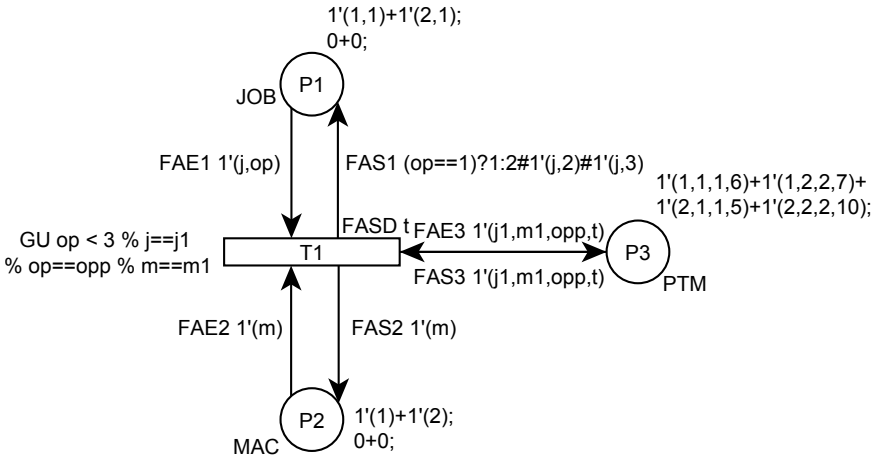


Fig. 2.4. The TCPN model of a 2×2 job shop instance.

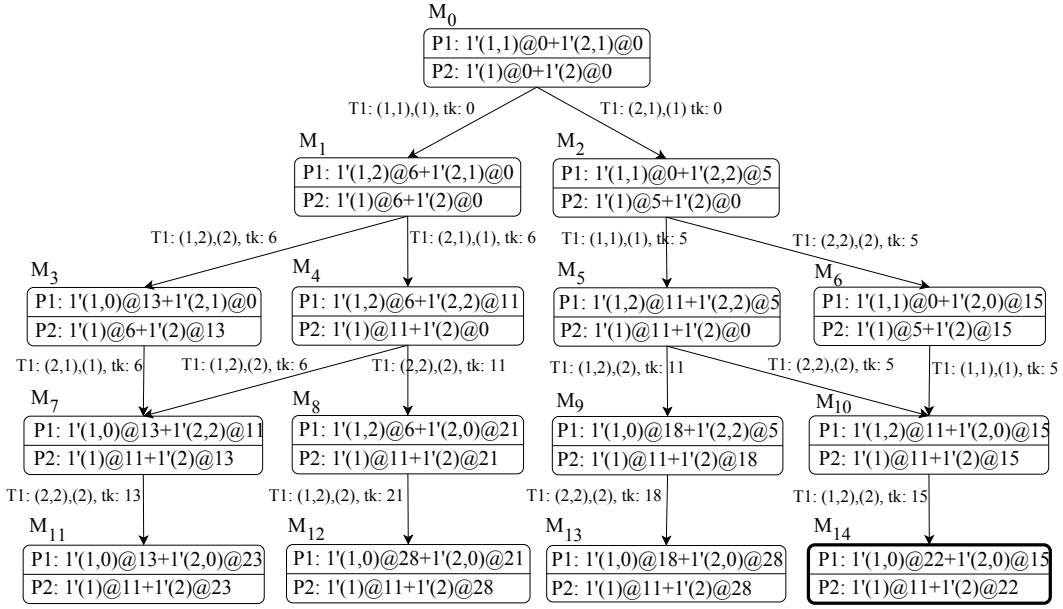


Fig. 2.5. The ESS graph of the 2×2 job shop instance.

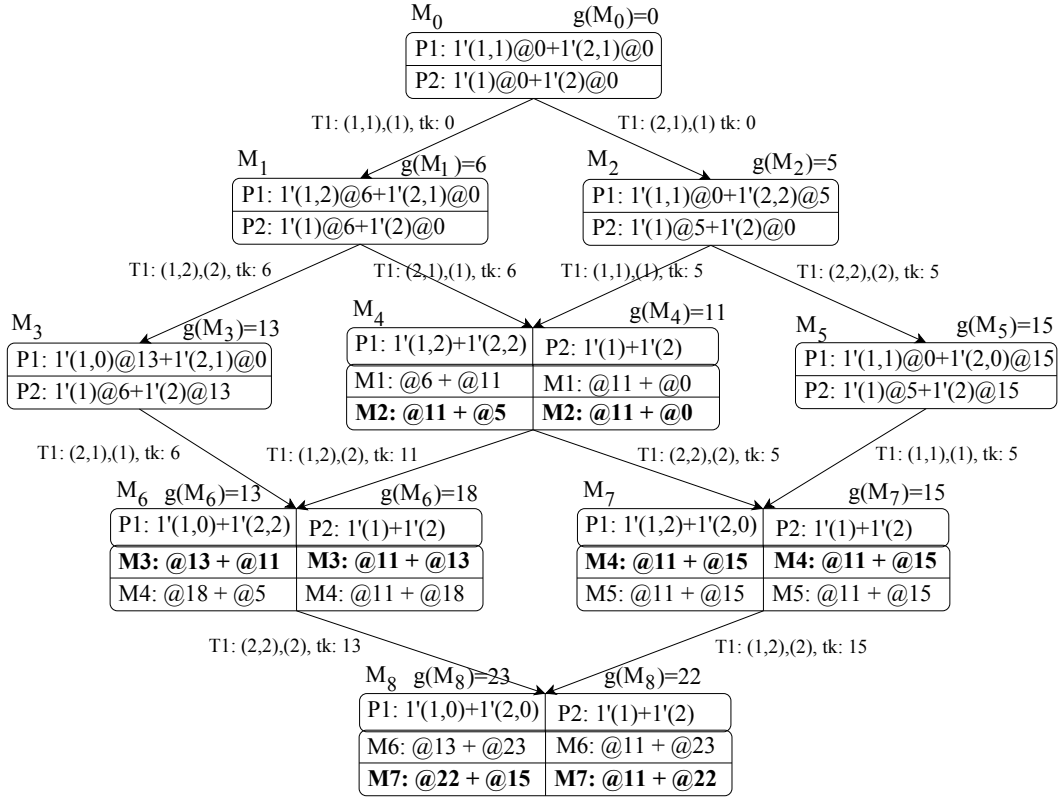


Fig. 2.6. The CESS graph of the 2×2 job shop instance using $g(M)$.

Algorithm 2.1 A* search with TCPN execution

Require: $TCPN$, M_0, M_g
1: $d_m \leftarrow 0$
2: $g(M_0) \leftarrow 0$, $f(M_0) \leftarrow h(M_0)$
3: $OPEN \leftarrow \{M_0\}$, $CLOSED \leftarrow \{M_0, d_m\}$
4: **while** $OPEN \neq \emptyset$ **do**
5: $M \leftarrow OPEN \setminus \{M_{best}\}$
6: **if** $IsM_g(M)$ **then**
7: $M_f \leftarrow M'$, construct solution path
exit
8: **else**
9: **for all** enabled transitions $t \in T : M[t\rangle_{\tau_k} M', \nexists \tau_{k'} < \tau_k : M[t\rangle_{\tau_{k'}}\}$ **do**
10: **if** $M'(M'_u) \notin CLOSED$ **then**
11: $CLOSED \leftarrow CLOSED \cup \{M', i+1\}$
12: $OPEN[i+1] \leftarrow OPEN[i+1] \cup \{M'\}$
13: **else**
14: $CSS((f(M_{stored}), g(M_{stored})), (f(M'), g(M')))$
15: **end if**
16: **end for**
17: **end if**
18: **end while**
19: **return** M_f and solution path

Like A*, most heuristic search algorithms use two data structures: the open and closed lists. The open list (OPEN) is a queue that stores the markings that have been generated but not yet expanded, whereas the closed list (CLOSED) which is usually represented by a hash table, stores the already-expanded (visited) markings. The heuristic search algorithm determines how OPEN is implemented, as a priority or non-priority queue. A* uses a priority OPEN in which markings are sorted in the increasing values of $f(M)$. Contrary to the standard approach, TIMSPAT implements CLOSED as a list that keeps both the open and closed markings. This is due to the high run-time cost incurred on performing duplicate detection on a queue. To avoid duplicating markings on both lists, TIMSPAT keeps only the pointers to the open markings in OPEN and their corresponding heuristic cost values. The common data structure allows TIMSPAT to integrate different heuristic search algorithms in the tool.

The pseudocode for the A* search combined with TCPN execution is given in Algorithm 2.1. Here, the algorithm uses both $f(M)$ and $g(M)$ for the CSS duplicate detection procedure $CSS((f(M_{stored}), g(M_{stored})), (f(M'), g(M')))$ in order to provide a more accurate estimate in selecting the most promising time stamp set. Fig. 2.6 reveals that it is quite difficult to break ties (Marking M_4) using $g(M)$ as the criterion to discard untimed marking duplicates. Also, the myopic evaluation of $g(M)$ can prevent the search algorithm from obtaining the best path that leads to an optimal solution. Although a good lower bound $f(M)$ estimate is required for the CSS procedure.

2.5.2 Heuristic Functions

Three admissible heuristic functions are commonly used in PNAIHES approach. The first one sets $h_1(M) = 0$ assuming no heuristic information is available. This is suitable for FMS with routing flexibilities or alternative routings and in cases where the run-time overhead for heuristic computation is quite high. However, the resulting lower bound $f(M)$ might be too weak to reach an optimal schedule in a reasonable time. On the other hand, it can be very useful in cases where the algorithms

are designed to return suboptimal solutions quickly [87] **Paper V**. The second one is called the job heuristic function [114, 115]: $h_2(M) = max_k\{\xi_k(M), k = 1, 2, \ldots, N\}$ calculated as the maximum of each kth job remaining time on uncompleted operations, $\xi_k(M)$ and N is the total number of jobs. The third is called the machine heuristic function [26]: $h_3(M) = max_i\{\xi_i(M), i = 1, 2, \ldots, R\}$ where $\xi_i(M)$ is the sum of operation times of those remaining operations for all jobs which are planned to be processed on the ith resource when the current system marking is represented by M and R is the total number of resources. These three functions have been used in [78, 80, 81] **Papers II**, **III** and **IV**.

In the formulations of $h_2(M)$ and $h_3(M)$, the timed marking information is not used in the computation. To this effect, Li et al. [115] propose tighter lower bound estimates for TPN that consider the earliest available time of machines and jobs based on the information from the timed marking. The individual time stamps of tokens are used to calculate the lower bound. The modification $f_{2m}(M)$ for TCPN as proposed in [115] is as follows: Given a token s_j of a job J_k in place p_n with color variables j (job identifier) and op (operation), to be processed on a set of machine tokens s_{m_i} in place p_m, where $i = \{op_s, op_s + 1, \ldots, op_f\}$ and m_i is the machine list used for the job's operations from the next one op_s to the last op_f. Then, the $f_{J_k}(M)$ for each job is estimated as:

$$f_{J_k}(M) = max\left\{tm[s_j] + \sum_{op_s \le i \le op_f} D(t_{\langle b(j)=k, b(op)=i\rangle}), tm[s_{m_{op_s}}] + \sum_{op_s \le i \le op_f} D(t_{\langle b(j)=k, b(op)=i\rangle}),\right.$$
$$\left. tm[s_{m_{op_s+1}}] + \sum_{op_s+1 \le i \le op_f} D(t_{\langle b(j)=k, b(op)=i\rangle}), \ldots, tm[s_{m_{op_f}}] + D(t_{\langle b(j)=k, b(op)=op_f\rangle})\right\} \tag{2.3}$$

where $max(tm[s_j], tm[s_{m_{op_s}}])$ corresponds to $g_{J_k}(M)$, the earliest available time (firing time) of job J_k and $D(t_{\langle b(j)=k, b(op)=i\rangle})$ is the transition delay for bindings $b(j) = k$ and $b(op) = i$, the processing time of the job for each operation.

The overall lower bound $f_{2m}(M)$ for $h_{2m}(M)$ is given as:

$$f_{2m}(M) = max\left\{f_{J_k}(M)\right\}, \quad k = 1, 2, \ldots, N \tag{2.4}$$

A similar modification is made to $f_3(M)$ where the earliest start time of the next job operation on a machine is computed using $f_{J_k}(M)$ before adding the sum of the operation times. To show the effectiveness of these heuristic functions, Fig. 2.7 depict the A* search of the CESS graph of Fig. 2.6 using the previous $f_2(M)$ and modified $f_{2m}(M)$ job heuristic functions, while Fig. 2.8 shows those of the machine heuristic functions. The x and y variables in the marking identifier M_{x-y} represent the order of expansion of the CESS by BFS (Fig. 2.6) and A* respectively. As observed in the graphs, the improved heuristic functions proved to be more informed than the previous ones, expanding and storing fewer markings i.e. $h_2(m) \le h_{2m}(M)$ and $h_3(m) \le h_{3m}(M)$. Also, Fig. 2.7a shows the importance of using a good estimate as the A* search degenerated into a breadth-first.

While the job and machine heuristic functions can be used separately, they can also be formulated as $f(M) = max(f_{2m}(M), f_{3m}(M))$ [115] to give a more accurate lower bound. The computation may become more time consuming, especially for TCPNs. However, in FMS with alternative routings in which more than one machine can be used to process the operation (with different processing times) of some jobs, only the job heuristic function and $f_1(M)$ seem applicable.

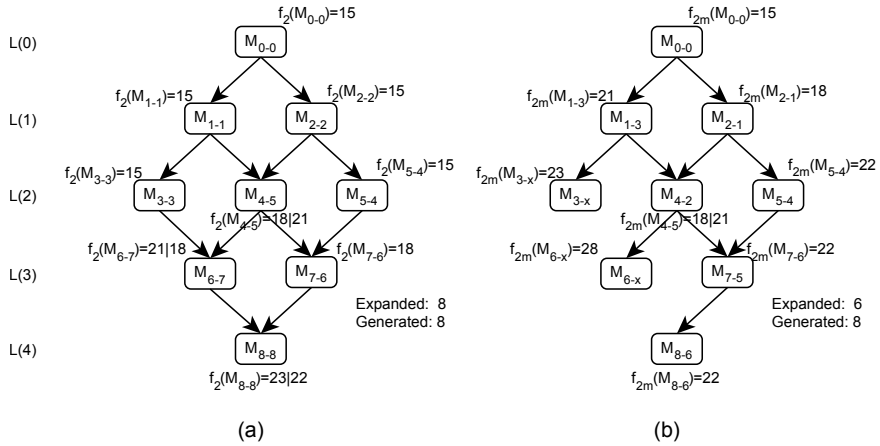


Fig. 2.7. A* search using (a) $f_2(M)$, and (b) $f_{2m}(M)$.

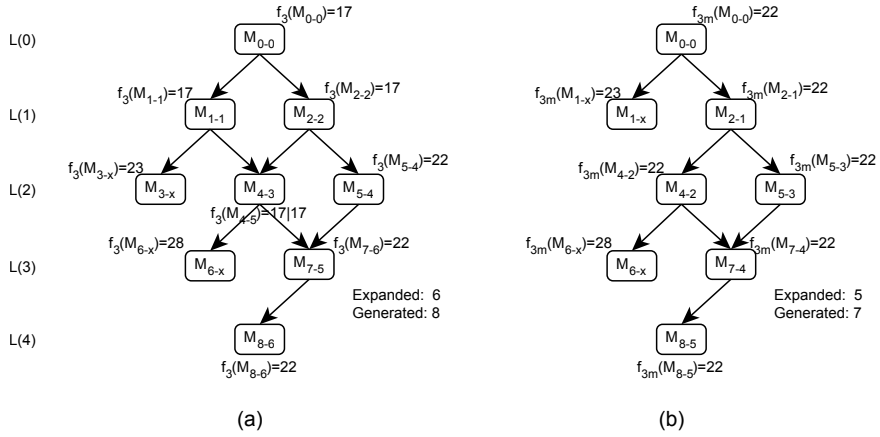


Fig. 2.8. A* search using (a) $f_3(M)$, and (b) $f_{3m}(M)$.

The job heuristic function is modified for alternative routings by replacing the processing time of each operation, and the time stamp of machines with the minimum processing time, and the machines with the earliest available time respectively.

$$
\begin{aligned}
f_{J_k}(M) = max \Bigg\{ & tm[s_j] + \sum_{op_s \le i \le op_f} min(D(t_{\langle b(j)=k, b(op)=i \rangle})), \\
& min_{s_{m_{op_s}} \in S_m}(tm[s_{m_{op_s}}]) + \sum_{op_s \le i \le op_f} min(D(t_{\langle b(j)=k, b(op)=i \rangle})), \\
& min_{s_{m_{op_s+1}} \in S_m}(tm[s_{m_{op_s+1}}]) + \sum_{op_s+1 \le i \le op_f} min(D(t_{\langle b(j)=k, b(op)=i \rangle})), \\
& \ldots, min_{s_{m_{op_f}} \in S_m}(tm[s_{m_{op_f}}]) + min(D(t_{\langle b(j)=k, b(op)=op_f \rangle})) \Bigg\}
\end{aligned}
\tag{2.5}
$$

where $tm[s_{m_i}]$ is the time stamp list of machines that can be used to process a job for a given operation. In this formulation, the earliest available time of the machine with the minimum time stamp is updated each time it is selected as the candidate machine. The machine will be considered

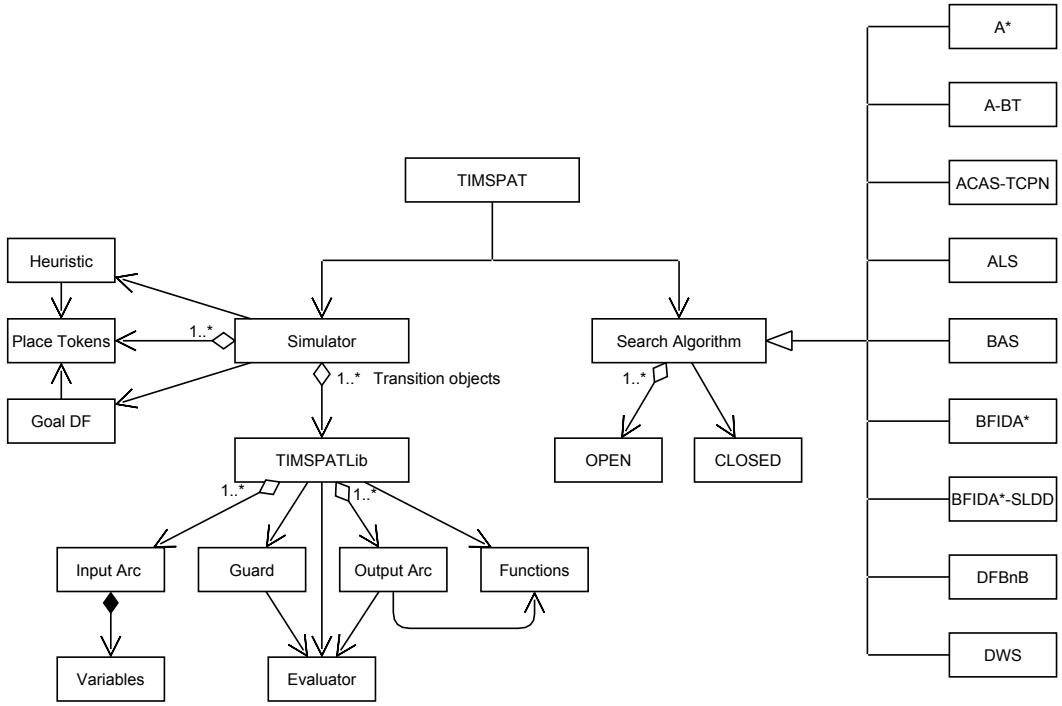


Fig. 2.9. Relationship between the three components and the classes used in TIMSPAT.

to have processed an operation in case it is part of another set of alternative machines to be used for subsequent operations. This ensures that the calculation of the lower bound advances without overloading a machine, and resource utilization is spread out to the other machines.

A* offers completeness and optimality guarantee. However, it requires a large amount of search space and computational time effort before an optimal solution can be reached. A* keeps all nodes in memory, which has limited its applicability to small problems. Besides A*, 7 other algorithms have been implemented: 1. Breadth-first iterative deepening A* search (BFIDA*) [77, 78] (**Paper III**), 2. BFIDA* with scalable layered duplicate detection (BFIDA*-SLDD) [80] (**Paper IV**), 3. Beam A* search (BAS) [63], 4. A* with backtracking (A*-BT) [70], 5. Dynamic window search (DWS) [62, 79], 6. Anytime layered search (ALS) [78] (**Paper III**), and 7. Anytime column adaptive search (ACAS) [81] (**Paper II**). ESS and CESS form the base classes of the heuristic search algorithms in TIMSPAT despite the fact that BAS and DWS selects only a subset of successors generated at each marking. The details of each algorithm can be found in its respective citation.

Fig. 2.9 shows the interaction between the three components and the relationships between the classes used in TIMSPAT. The evaluator class is used to evaluate and compute guard conditions, functions, and other mathematical expressions in the output arc.

3

State of the Art Review on PNAIHES Approach

3.1 Introduction

There are two approaches to dealing with the space and time requirements of the reachability graph under the PNAIHES methodology. The first approach termed heuristic function-dependent methods (*HFDM*) adapts classical heuristic search algorithms like A* and beam search [116], and devise efficient heuristic functions to reduce the search space to explore and possibly minimize the time depending on the kind of function employed. A heuristic function can be admissible or non-admissible. An admissible heuristic function is a lower bound that does not overestimate the cost to the goal marking and guarantees that an optimal solution is obtained whereas a non-admissible overestimates the remaining cost to the goal marking.

The commonly used heuristic search algorithm is A*. However, the performance of A* highly depends on the strength of the heuristic function. A tight lower bound (strong) function is usually needed so that an optimal solution can be reached quickly. Conversely, a strong heuristic function is usually too expensive to compute [117]. The closer the function value is to the exact $h(M)$, the lesser the number of markings to be explored. The decision to choose between a strong and a weak heuristic function is a function of the space and time complexity. Using a strong and admissible heuristic function may lead to a reduction in the search space. However, this does not automatically reduce the search time. Apart from the time involved in computing a strong heuristic function, it is generally impossible to predict the effectiveness of the function to deal with the time and memory requirements. Most works adopts non-admissible heuristic functions to reduce the search space at the cost of losing optimality.

The second approach focuses on the development of effective algorithms that combines one or more search algorithms called hybrid heuristic search (*HHS*). The algorithms developed in this thesis are related to the second approach. In PNAIHES, the common objective function considered is the minimization of the completion time (makespan). Criteria like total tardiness and mean flow time have been rarely used.

3.2 HFDM Review

Shih and Sekiguchi [118] propose the first application of a heuristic search method to on-line FMS scheduling with routing flexibility. They combine transition-timed PN and beam search to solve scheduling conflicts when one or more transitions are enabled. The beam search algorithm applies the beam width at each level of the reachability graph in which partial schedules are constructed and evaluated within a given beam depth until a complete schedule is obtained. The beam width and beam depth are used to restrict the number of marking at each level and the number of levels in the graph respectively. The approach does not guarantee optimality. Nonetheless, this is not a requirement for on-line scheduling.

Lee and DiCesare [41] are the first authors that employ an intelligent global heuristic search method called L1 algorithm, by adapting the A* search to TPN scheduling. They propose three non-admissible heuristic functions in order to find near-optimal schedules in a reasonable amount of time. The first heuristic function prefers markings that are deeper in the reachability graph (i.e. closer to the goal marking), the second favors markings which has an operation ending soon, while the third is a hybrid of the first two. The three functions are evaluated on different FMS examples with routing flexibility and material handling systems (robot). Lee and DiCesare [119] extend the application of the L1 algorithm to integrated scheduling of FMS employing AGVs.

Jeng and Chen [120, 121] propose a modified heuristic function based on PN state equations that considers the global state information unlike the previous functions proposed by Lee and DiCesare [41]. A* search is adapted to avoid traversing the whole reachability graph and a pruning technique based on concurrency information of the reachability graph decides whether or not to remove markings with the same parent node from the state space. Experimental results show that the method is better than [41, 119]'s solutions.

Also, Jeng and Chen [121] exploit the linear characteristics of the state equation. Because of the mathematical properties, the use of this method is limited to small size problems. Jeng et al. [122] use the same heuristic search method in [120, 121] for exploring the structural properties of generalized symmetric net and asymmetric net. They propose a new heuristic function based on the multiplication of a scaling factor and state equation solution to schedule FMS with assembly operations.

Elmekkawy and Elmaraghy [123] evaluates the performance of three heuristic functions on the HHS algorithm proposed by Abdallah et al. [124] to optimize the mean flow time for deadlock-prone FMS. The functions are the remaining processing time, the average operation waiting time, and the use of dispatching rules such as shortest processing time. These functions are used to relax the optimality guarantee of the hybrid algorithm in order to obtain a quick solution.

Lee and Lee [114] propose four new heuristic functions (admissible and non-admissible) that are useful for multiple lot size scheduling problems in FMS. The functions are evaluated using A* search. The authors claim that the heuristic functions are more efficient in terms of space and computation time than the admissible heuristic function based on resource cost reachability matrix proposed by [64]. However, Huang et al. [125] provide some counter examples that demonstrate that one of the proposed functions is not always efficient.

Huang et al. [126] propose a combination of admissible and non-admissible heuristic functions to generate a more informed function in order to reduce the search time of A* search. Li et al. [115] improved both the machine heuristic function proposed by [26] and the job heuristic functions from [114, 126] for the single and multiple lot size scheduling problems. The modifications were done by taken into account the earliest available time of shared resources and subparts. Also, they emphasize on the role of heuristic functions in the A* search process and several functions were evaluated on different sets of benchmark problems. The new new heuristic functions proposed proved to be more informed than the existing ones.

Luo et al. [127] extend the HHS algorithm called dynamic window search (DWS) proposed by Moro et al. [79] to deadlock-free scheduling, by integrating deadlock control policies. Also, they presented three heuristic functions to improve the search performance. Huang et al. [128] propose an admissible heuristic function for FMS with alternative routings based on place-timed PN. The function is then used with a dynamic weighting A* search strategy for scheduling. All the aforementioned research works are focused on TPN scheduling.

Cavalieri et al. [86] are the first authors that applied a TCPN-based scheduling framework with heuristic search to improve a flexible semiconductor manufacturing system of the SGS-Thomson plant in Italy. The L1 algorithm [41] is adapted by introducing color and a new heuristic function to solve the problem of dispatching at each machine. They propose a linear combination of three heuristic functions for multicriteria optimization, with each related to the minimization of a given performance objective. The first two functions proposed by [41] is used to minimize the work-in-process and makespan objective, and the third one calculates whether the current operation is late with respect to its schedule in order to meet the due date.

3.3 HHS Review

The efficiency of the existing PN-based HHS algorithms can be classified into three categories according to the space/time tradeoff criterion: 1. Space-efficient (SE), 2. Time-efficient (TE), and 3. Space/Time-efficient (STE).

For an algorithm to be considered SE, the reduction of the memory requirements must not affect the optimality of the schedule i.e. the heuristic function must be admissible and no inadmissible pruning technique should be adopted. These algorithms are oriented toward obtaining optimal solutions if given sufficient time, in addition to an efficient use of memory. Upper bounds are used to remove paths whose markings will not lead to a better solution. The SE algorithms do not terminate the search until the optimal solution is found.

For time efficiency, the only criterion is that the algorithm returns a solution (either optimal or near-optimal) in a reasonable amount of time irrespective of the type of heuristic function and the pruning technique employed. Here, optimality is sacrificed for computation time and memory reduction. The TE algorithms terminate the search as soon as the first solution is obtained.

STE algorithms must meet the SE requirements in addition to returning solution at a reasonable amount of time. Basically, they consist of anytime algorithms that report solution at different time intervals and are guaranteed to provide the best solution obtained so far whenever interrupted. STE algorithms can be considered as a special class of HHS methods. They do not stop the search at the first solution. Instead, the solution is continuously improved over time until the search obtains the optimal solution provided the available memory is sufficient to guarantee optimality. The algorithms trade off solution quality and computational time. The incumbent best solution is used as an upper bound to restrict the number of generated successors and to periodically prune markings that will not lead to a better solution. According to the given classification, we review the existing HHS algorithms.

3.3.1 SE Class

Not so much importance has been given to the SE class in the literature. The interest in this area is practically non-existent for the PNAIHES approach. This may be due to the time requirements and the trend of current research methods toward obtaining near-optimal solutions in a reasonable amount of time. However, they are still very applicable to off-line scheduling problems where enough time is provided before execution. The SE class is more of a research area explored by the model checking and AI communities. Examples are: sweep-line method [129], frontier search [130], transition locality [77, 131, 132] and breadth-first iterative deepening A* search [77].

The only existing method that closely matches the SE requirements is the time-line search proposed for TCPN-based scheduling by Mujica and Piera [133], Mujica Mota and Piera Eroles [134], Mujica Mota and Piera [135]. However, it was only used as an aid to reduce the memory requirements of the state generation phase in the two-phase algorithm that is targeted toward a TE solution rather than SE. Hence, the time sweep-line capability was not fully exploited.

The time-line search is based on the sweep-line method proposed by Jensen et al. [129] in the model checking community. The sweep-line uses the concept of progress measure to delete markings from the memory during state space exploration so as to reduce the peak memory usage. Markings are stored and explored in a layered manner according to their progress values. Once all the markings with the least progress value in a given layer are expanded, they are removed from memory and the search continues exploration with the next layer. This search method is equivalent to the breadth-first generation of the state space. The markings with a lesser progress measure

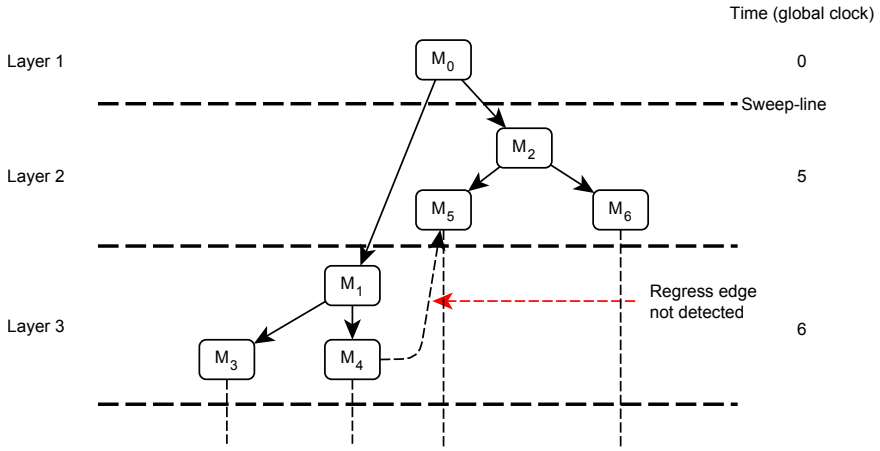


Fig. 3.1. The time sweep-line exploration of the ESS graph in Fig. 2.5 using the global clock as the progress value.

are safely deleted from memory since they will never be reached again and thus, not needed for duplicate detection.

Mujica and Piera [133], Mujica Mota and Piera Eroles [134] adopt a notion of time that uses the firing time as the progress measure for TCPN-based scheduling in which markings with the same firing time are kept at the same level of the state space. Though, a closely related method was first presented by Christensen et al. [136] for the model checking of TCPN. They propose a time sweep-line method using the increasing creation time values of markings for the CSS generation of TCPN. The creation time of a marking is defined in [136] as the time at which the firing of a transition changed the marking of the TCPN into this marking. Clearly, this is synonymous to the firing time notion used by [133, 134] for the time-line search.

Notwithstanding, the time-line search does not consider the case of regress (backward) edges [137] in which the successors of markings with higher progress values are duplicates of markings with lower progress values, already removed from the state space. This may result in a repeated exploration of some parts of the state space (cycles). Apparently, it is quite difficult to detect regress edges if only progress values are used. As an example, let us explore the ESS graph in Fig. 2.5 with the time sweep-line search. Fig. 3.1 shows the first 3 layers of the time sweep-line graph using the global clock as the progress value. Before marking M_1 is expanded, Layer 2 must have been deleted from the memory. As a result, M_4 will not be detected as an untimed marking equivalence of the already deleted M_5. This exploration will default to the classic ESS with memory savings but clearly defeats the use of CSS for TCPN.

3.3.2 TE Class

The TE Class is a well-studied area. It is mainly composed of incomplete heuristic search algorithms that do not guarantee optimality in which the aim is to find a first solution very quickly.

Sun et al. [138] propose a Limited-Expansion A* search algorithm that uses the idea of stage search by employing a pruning procedure that removes less promising markings from the reachability graph when the OPEN list exceeds a given maximum capacity. The idea makes the algorithm analogous to the beam search method. The algorithm uses one of the heuristic function proposed by [41] for quick solution termination and employs a non-delay scheduling similar to the RSS generation. The search method is applied to a PN modeling of an entire FMS that includes AGV scheduling and control for multiple AGVs.

Xiong et al. [139] propose two HHS algorithms that combines best-first strategy (BF) with controlled backtracking (BT) to reduce the memory requirements at the expense of narrowing the evaluation scope. They present an admissible heuristic function based on the operation time of the remaining operations for all jobs. The first HHS algorithm called BF-BT applies BF at the top of the reachability graph while BT is applied at the bottom through the depth-bound strategy. The second HHS algorithm called BT-BF reverses the ordering of the search algorithms; BT at the top and BF at the bottom. It is based on the notion that the quality of a schedule is more affected by the decision taken at the early stages rather than toward the completion of the schedule (goal marking).

Xiong and Zhou [140] extend the two algorithms to deadlock-free scheduling of FMS with shared resources and resources constraints such as limited buffer capacity and material handling system. Also, Xiong and Zhou [26] evaluates the two HHS algorithms on semiconductor test facility scheduling.

Moro et al. [79] propose a search method that deals with two aspects of search strategy:

1. Scope of selection: "the degree to which a search procedure allows the generation for further exploration of all possible alternatives of a marking.

2. Scope of recovery: "the degree to which a search allows recovery from disappointing search decisions to access previously suspended alternatives."

They propose an intelligent successor generation (IGS) based on active scheduling and a HHS algorithm called dynamic window search (DWS). The DWS follows the basic working principles of A* and stage search. It aims to reduce the size of the search graph by reducing the scope of selection and recovery of A* and a truncation of the number of candidate markings. DWS keeps a specified search window size of a certain number of levels in the graph guided by two parameters, *bottom-depth* and *top-depth*. The search window dynamically advances using two rules: when the size of the *bottom-depth* equals zero and when the number of markings at the *top-depth* exceeds a given number of markings called *max-top*. This is done so as to reduce the backtracking capability of A* and the number of paths for further exploration. DWS uses a third rule that limits the number of markings stored at each level of the graph, called *max-size*.

The work of Moro et al. [141] is practically the same paper as Moro et al. [79] that uses the same IGS and DWS, termed differently as controlled generator of successors (CGS) and dynamic look-ahead stage search (DLSS).

Reyes et al. [62] propose a new class of PN called buffer-nets and a new heuristic function based on resource cost reachability (h_{RCR}) matrix. The h_{RCR} uses the properties of the buffer-nets and assumes that jobs can always follow the path with the lowest operation time. The DWS algorithm is then adapted for scheduling.

Yu et al. [64] use the same h_{RCR} function proposed by Reyes et al. [62] in conjunction with two HHS algorithms. The first HHS algorithm combines A* and Hill climbing by limiting the maximum number of successors that can be generated at each node in the graph while the second adapt the DWS algorithm [79] on a different set of FMS examples.

Mejia and Odrey [63] propose a HHS that combines A* search with beam search called Beam A* search (BAS) to find near-optimal schedules in a timely manner. The BAS algorithm uses three pruning strategies (all non-admissible) to reduce the space and time complexity. The first pruning avoids state explosion using controlled search deepening by limiting the number of markings to be stored and expanded at each level called beam width. The second reduces the search space by eliminating non-promising markings using non-delay scheduling. The third reduces the size of the OPEN list by pruning markings from the list when the size reaches a certain cutoff value. However,

the authors confirm that the non-delay scheduling strategy (RSS) is not suitable for deadlock-free scheduling of FMS with a high number of deadlocks. They demonstrate with an extensive set of benchmark examples that the proposed algorithm improves the performance and speed of prior search algorithms proposed by [26, 41, 139, 142]

Mejia and Montoya [32], Mejía and Montoya [143, 144] extend the BAS algorithm to other applications such as the minimization of the total tardiness, deadlock-free scheduling of FMS with blocking (no buffer systems), and resource assignment and scheduling respectively.

Kim et al. [145] present a reactive graph search algorithm for dynamic scheduling of machines and AGVs. The algorithm consists of Real-Time A* search [146] and a rule-based supervisor (dispatching rule) used to find near-optimal solutions with small computational efforts. The objectives are the minimization of the makespan and total tardiness. Instead of the conventional scheduling technique where the scheduling process must be completed before the first execution, the proposed method alternates the search phase and execution phase based on the current state of the system. This is done to make the scheduling system adapt to unexpected changes in the production environment. Also, they propose a modeling method that divides the TPN into two submodels: *System Net* and *Process Net*. The *System Net* describes the physical behavior of FMS such AGV and work-in-process behavior while the *Process Net* represents the logical behavior that includes the job scheduling process on machines

Huang et al. [69] propose a combination of A* and DFS to reduce the computation time requirements of A*. It uses DFS to prioritize markings at the deeper level of the A* search graph. In addition to the search speed, the algorithm controls the quality of the solution obtained such that the cost does not exceed the optimal cost by more than a factor $1 + \epsilon$.

Huang et al. [70] present a hybrid heuristic search scheduling strategy by combining A* and backtracking (BT). The algorithm performs A* locally and BT search globally to overcome the drawbacks of the best performing HHS algorithm BT-BF proposed by Xiong and Zhou [26], Xiong et al. [139]. The improvement over the previous algorithms Xiong and Zhou [26], Xiong et al. [139] is demonstrated on a particular set of FMS examples.

All the works discussed so far are TPN-based. Only a few articles have considered TCPN-based scheduling. The works of Mujica et al. [112], Mujica and Piera [133], Mujica Mota and Piera Eroles [134], Mujica Mota and Piera [135], Mujica and Piera [147] are the most representative. They are all based on a two-phase algorithm for optimizing scheduling problems based on TCPN. The first step called the generation phase uses the DFS to generate the CSS in order to find a feasible schedule. This step only considers the untimed state space and separates the time values from the state space evaluation. The second step called the optimization phase analyze and optimize the time values of the obtained feasible solution path including the stored time stamp set of equivalent untimed markings.

Since the full CSS cannot be generated, different search strategies were provided by the authors to improve the two-phase algorithm and further reduce its memory requirements. [133–135] integrate the time-line search into the CSS method for the generation phase. Mujica et al. [112] propose two non-admissible heuristic functions based on absolute time values and probabilistic values, that helps guide the search process efficiently in the optimization phase. Mujica Mota and Piera [135] propose an improved version of the time-line search that employs consistency evaluation and better detection of duplicate markings to improve the time efficiency.

Mota and Piera [108] propose three improvements for the two-step algorithm related to the state space exploration of TCPN vis-Ãă-vis transition evaluation, data management, and information search for duplicate detection. The aim is to reduce the time consuming tasks in marking generation and storage. For data management, an efficient data structure is presented to avoid the storage of

redundant information in markings. It is based on the notion that there is only a minimal difference in the marking information between a parent marking and its successor. Hence, it is not necessary to store all the marking information each time a new marking is generated. However, most of these improvements were mostly applied to small-sized problems (job shop problems like 3×3 and 6×6) compared with prior works on TPN scheduling.

3.3.3 STE Class

The only STE algorithm applied to PN scheduling can be traced back to 1998. Abdallah et al. [124, 142] present an efficient heuristic search algorithm to obtain optimal deadlock-free schedules for a class of FMS called Systems of Sequential Systems with Shared Resources S^4R. [142] is an improved journal version of the conference paper [124]. The algorithm combines DFS with branch and bound in two steps: initialization, and optimization.

The initialization step uses DFS to obtain a quick initial solution. This solution is then set as an upper bound for the next step. The marking generation algorithm is based on priority rules such as Least Work Remaining (LWKR) and shortest processing time (SPT), to determine which transitions to fire first if there is more than one enabled transition. It fires only one transition at a time just like in a simulation context using the global clock. The remaining transitions are stored and used during the optimization process.

In the optimization step, the algorithm backtracks to the stage where alternative transitions exists and repeats another DFS until a new goal marking is found. The search continues with the backtracking-DFS procedure until all markings have been explored and the optimal solution is reached. The upper bound keeps track of the best solution found and it is used to prune markings whose time is greater than the current bound. Also, the search algorithm is extended with the use of truncation techniques based on PN siphon concept. Though, the algorithm described only returns the last found solution (i.e. optimal), the notion of time is not discussed and the incumbent solution path (current best solution) is not stored. The algorithm is similar to DFBnB but differs in the way backtracking is initiated. While the backtracking is controlled by the number of alternative transitions, the DFBnB is strictly depth-first for all iterations.

3.4 Summary

This chapter has presented a comprehensive review on the application of PN and heuristic search methods to FMS scheduling. Two different approaches have been identified. Since the inception of the PNAIHES in 1991, several algorithms and heuristic functions have been proposed. In spite of this, most of the algorithms have only been tested on TPN models. Also, no bechmarking platform exists for these algorithms. From the review, only two areas have been well exploited by the PN research community; the HFDM and the TE class of the HHS method. In this light, this thesis focuses on the less developed areas: SE and STE classes.

4
Paper II

Olatunde T. Baruwa & Miquel A. Piera

Anytime heuristic search for scheduling flexible manufacturing systems: a timed colored Petri net approach

The International Journal of Advanced Manufacturing Technology 2014;75:123–137

5
Paper III

Olatunde T. Baruwa, Miquel A. Piera, & Antoni Guasch

Deadlock-free scheduling method for flexible manufacturing systems based on timed colored Petri nets and anytime heuristic search

IEEE Transactions on Systems, Man, and Cybernetics: Systems 2015;45(5):831–846

6
Paper IV

Olatunde T. Baruwa & Miquel A. Piera

Identifying FMS repetitive patterns for efficient search-based scheduling algorithm: A colored Petri net approach

Journal of Manufacturing Systems 2015;35:120–135

7
Paper V

Olatunde T. Baruwa & Miquel A. Piera

A colored Petri net-based hybrid heuristic search approach to simultaneous scheduling of machines and automated guided vehicles

Revised Manuscript Submitted for Publication in International Journal of Production Research

Abstract

To achieve a significant improvement in the overall performance of a flexible manufacturing system, the scheduling process must consider the interdependencies that exist between the machining system and transport system. However, most works have addressed the scheduling problem as two independent decision making problems, assuming sufficient capacity in the transport system. In this paper, we study the simultaneous scheduling (SS) problem of machines and automated guided vehicles using a colored Petri net (CPN) approach under two performance objectives; makespan, and exit time of the last job. The modeling approach allows the evaluation of all the feasible vehicle assignments as opposed to the traditional dispatching rules, and demonstrates the benefits of vehicle-controlled assignments over the machine-controlled for certain production scenarios. Based on CPN modeling, SS is performed using a hybrid heuristic search algorithm to find an optimal or near-optimal schedule by searching through the reachability graph of the CPN with heuristic functions. Large-sized instances are solved in relatively short computation times, which were a priori unsolvable with conventional search algorithms. The algorithm's performance is evaluated on a benchmark of 82 test problems. Experimental results indicate that the proposed algorithm performs better than the conventional ones, and compares favorably with other approaches.

Keywords: Flexible manufacturing systems · Petri nets · Simultaneous scheduling · Automated guided vehicles · Hybrid heuristic search · Simultaneous scheduling of machines and AGVs · Timed colored Petri nets

7.1 Introduction

In flexible manufacturing systems (FMS) [119], automated material handling systems (MHS) facilitate the movement of raw materials and work-in-process between workstations according to a given sequence of operations or task. The handling operations are usually performed by robots, conveyors, automated guided vehicles (AGV) etc, to reduce the labor-intensive and time-consuming tasks thereby increasing productivity as well as shortening the delivery time of products. Due to their high degree of flexibility, AGVs have found increasing applications in modern manufacturing systems. They are battery-powered unmanned vehicles that move along a defined path guided by either wire or optic or magnetic. The advantages offered by AGVs such as increased flexibility, better space utilization, improved floor safety, reduction in overall operating cost, and easier interface with other automated systems [148], make them a suitable alternative to traditional MHS.

Scheduling is a decision making process that plays a vital role in improving the performance of an FMS. In traditional machine scheduling models, it is assumed that MHS are always ready and available to move parts whenever needed [149] such that material handling times are ignored in the scheduling process. This assumption holds at the academic level and for those FMS in which infinite transport capacity can be assured. In practice however, the physical layout constrains the transport capacity, corroborating that material handling operations can have a considerable influence on the overall performance of an AGV-served FMS. An AGV system is considered a critical component of an FMS. There exist different spatio-temporal interdependencies between machines and AGVs which requires a causal analysis of the tight couplings between machining and transport operations. The schedule of a job for the next operation depends on the transportation times of the AGV and vice versa. When the scheduling of machines and AGVs are treated separately, the lateness in the delivery of the next part for processing would result to machine idling. This can

create a gap within the system which can subsequently lead to bottlenecks in the system, and an underutilization of its resources [150]. As a result, an optimal machine schedule becomes an underestimation of the performance objective since the schedule of job operations depends on the AGV scheduling within the same system.

To bridge the gap, several researchers have demonstrated the benefits of coordinating AGV with machine scheduling [1, 151, 152] called simultaneous scheduling (SS), in terms of cost and lead times. As noted by Ulusoy et al. [117], a significant improvement in the performance of the FMS would be expected as a result of making the scheduling of AGVs an integral part of the overall scheduling activity. Consequently, the complexity of FMS scheduling increases with the integration of AGV scheduling. The SS problem involves not only the sequencing of job operations on machines but also the assignment of material handling tasks to AGVs, and the conflict-free routing of vehicles. To simplify the scheduling problem, most works have addressed the problem as two independent decision making problems [152]. The two subproblems are both known to be NP-hard [117].

This paper investigates the SS of machines and AGVs (SSMV) using a Petri net (PN) approach. PNs have been used extensively to model, simulate, and analyze FMS characterized as discrete event systems [27] due to their capability to mathematically and graphically model concurrency, parallelism, causal dependency, shared resources, and synchronization. The advantage of using PN is its ability to describe the system dynamics and the performance evaluation of the SS problem as a single model [119, 138], as opposed to the separate scheduling of other approaches [1, 153]. Colored PN (CPN) modeling (a high level PN) is preferred since it provides a concise representation of the system with the use of a data value called colored token, while maintaining the same modeling power of PN. The problem is formulated using timed colored Petri net (TCPN) modeling. The inclusion of the time concept allows one to conduct the performance analysis of the system which can be used to evaluate the different manufacturing scenarios under one or more objectives.

Based on CPN modeling, a reachability graph (or state space) is constructed to explore all the possible alternatives in terms of the firing sequence of transitions, in order to determine the best schedule that optimizes a performance objective. However, an exhaustive state enumeration is practically impossible due to the well-known state explosion problem and cannot be used to find optimal or near-optimal solutions to large-sized problems within reasonable computation times. As a result, most scheduling methodologies based on PN modeling [26, 32, 64, 70, 80] employ AI-based heuristic search methods to simulate the best scenarios by exploring a partial reachability graph with heuristic functions.

A* [154] is the commonly used baseline search method due to its completeness and optimality guarantee. However, the time and memory requirements have limited its application to small problem instances. To reduce the long computation times, several hybrid heuristic search methods based on PN modeling have been proposed. They combine two or more search methods to find suboptimal solutions quickly at the expense of losing optimality. Previous works on FMS scheduling have combined the A* search with backtracking [26, 70, 139, 155], beam search [32, 63], depth-first search [69], and stage search [62, 64, 141]. Most of these algorithms perform inadmissible pruning [138] to reduce the memory requirements of A*. Also, they terminate the search at the first solution even when the memory available can still be used to improve the obtained solution. Moreover, there is no guarantee that a solution will be returned at memory run out.

Two types of scheduling schemes have been considered: off-line, and on-line. The off-line approach schedule all operations for the entire planning horizon in which all parts are assumed to be available before the start of activities, whereas on-line scheduling attempts during execution to schedule operations one at a time as the scheduling decision is needed (or as the system status changes) [30]. While the time to solve off-line scheduling is not critical, on-line scheduling is time-constrained such that a limited amount of computation time is given to produce a solution.

The main contribution of this paper is twofold. First, we present two SSMV TCPN models that use an event-driven approach to evaluate all the possible alternatives for vehicle assignments without an imposition of a specific dispatching rule. The approach differentiates and examines the benefits of a vehicle-controlled assignment over the classical machine-controlled. Unlike previous methods [151, 153, 156], priority rules are not imposed on the solution models, in which the system's performance depends on the vehicle dispatching rule adopted. Two objective functions are considered: the makespan, and the exit time of the last job in the system. Second, we adapt and improve the hybrid heuristic search method called anytime layered search (ALS) [78], based on the reachability graph of TCPN in which optimal or near-optimal schedules can be obtained in relatively short computation times. It is aimed at tackling one of the inherent problems of A* when dealing with FMS problems where only weak heuristic functions can be applied [1]. The algorithm combines A* with suboptimal breadth-first branch and bound (sBFBnB) [77] and backtracking. It does not stop the search when the first solution is found. Instead, the search offers an anytime feature [81, 157] by finding a sequence of improving solutions while keeping track of the best solution cost until the search converges to optimal. Also, it is guaranteed to produce the best solution found even if the memory available is not sufficient to reach convergence. By convergence, we mean the algorithm needs to verify that the incumbent solution is the optimal before terminating the search process. The solution time efficiency makes it possible to adapt the proposed algorithm to on-line scheduling where decisions must be made in a short period of time [158].

7.2 Related Work

This section exclusively reviews the relevant literature on the SSMV problem. The problem has been formulated using different modeling techniques. A number of approaches describes the problem with mixed integer linear programming [1, 151, 153, 159–164] while a few works have considered PN [119, 138, 165], and disjunctive graph modeling [166]. The other works whose methods are based on evolutionary algorithms like genetic [117, 152, 167–169], differential evolution [170, 171], and simulated annealing [172], use a solution vector with fixed-length strings to represent a schedule called chromosome.

Due to intractability, the existing scheduling methodologies based on mathematical formulations adopt a decomposition framework. They solve the SSMV problem in two steps. First, a job-shop scheduling heuristic procedure is used to find an optimal or near-optimal machine schedule that excludes material handling activities. Then, given the machine schedule, a vehicle dispatching rule finds a feasible solution to the vehicle scheduling problem by integrating AGV assignments. Bilge and Ulusoy [1] present an iterative solution procedure that links the two subproblems in order to facilitate the search for a good solution. The machine schedule is generated using two algorithms, the Giffler and Thompson's active and non-delay schedule algorithms, while the vehicle schedule is handled by a sliding time window heuristic. They analyze the impact of processing and travel times, and the complexity of material flow pattern on the process route, and on different layout configurations. The other solution approaches like Raman et al. [159] use the concept of project scheduling under resource constraints, Lacomme et al. [153] propose a branch-and-bound coupled with discrete-event simulation framework, while Caumond et al. [162] extend the heuristic framework proposed in [153].

Instead of decomposing the problem, the metaheuristic solution methods use a chromosome to represent operation sequencing and AGV assignment. Ulusoy et al. [117] present the first genetic algorithm to solve the SSMV problem. They improve the solutions produced by the sliding time window heuristic in Bilge and Ulusoy [1]. The other metaheuristic algorithms described in [152, 167, 169–172] follow a similar approach. However, they differ in the solution representation and evaluation, and the vehicle assignment heuristic algorithms used for AGV scheduling. Although Lacomme et al. [166] modeled the problem as a job shop with several transport robots using disjunctive graph, they propose an efficient memetic algorithm whose objective is to provide solutions for large instances in short computational time.

Different FMS configurations have been considered depending on the guide-path layout,the number of AGVs and machines, and other resource constraints like buffer size limit. Bilge and Ulusoy [1] propose the most relevant benchmark instances on the scheduling problem which have been used by several publications [117, 152, 158, 161, 167, 169–172]. The problem consists of 82 test instances with 4 different path layouts, and 2 AGVs. Also, the variants of this benchmark problem have been studied: alternative routing of parts [170], and single AGV-based FMS with limited input/output buffer capacity at machines [153, 162]. Others consider just-in-time production of complex assemblies under multiple capacity constraints [151], conflict-free routing [163], and single AGV-based FMS in a closed loop [156, 160].

7.3 SSMV Problem Description

Consider an FMS that consists of a set of m machines $M = \{M_1, M_2, \ldots, M_m\}$, the load/unload (L/U) station where parts enter and leave the system, and a set of identical k vehicles (AGVs) $V = \{V_1, V_2, \ldots, V_k\}$ used for transportation of parts between two machines. There is a set of n jobs $J = \{J_1, J_2, \ldots, J_n\}$ to be processed on one or more machines. Each job J_j consists of an ordered sequence of n_j operations $O_{ij}(i = 1, \ldots, n_j)$. Each operation O_{ij} must be processed on a dedicated machine $\mu_{ij} \in \{M_1, \ldots, M_m\}$ without preemption for $p_{ij} > 0$ time units [173]. A machine can perform at most one operation at a time. Each machine has input and output (I/O) buffers in which parts are stored before and after processing. The buffers serve as pick-up and delivery (P/D) points for the AGVs.

Parts visit different machines in the system for different operations which in turn generates P/D requests for the AGVs. An AGV performs a transportation operation between any two operations O_{ij} and $O_{i+1,j}$, to move a job from the source machine μ_{ij} to the destination machine $\mu_{i+1,j}$ for the next processing. AGVs perform two types of trips; a loaded trip, and a deadheading (or empty) trip. A loaded trip is a delivery operation where the AGV moves a part from the output buffer of a machine μ_{ij} to the input buffer of another machine $\mu_{i+1,j}$. In an empty trip, the AGV moves from an idle position at a machine M_k without carrying a job in order to pick up a job waiting to be transferred from μ_{ij} to $\mu_{i+1,j}$, where $M_k \neq \mu_{ij}$. Let t_{ij} represents the travel time between any two machines M_i and M_j. The travel times are job independent and machine dependent.

The SSMV problem is formulated as follows: Given the FMS environment described, determine the sequence of operations and the starting and completion times of each job on each machine, and the trips between machines together with the assignment of transportation tasks to vehicles according to two criteria:

- To minimize the makespan $C_{max} = max_{j=1}^{n}\{C_j\}$, where C_j represents the completion time of the last operation $O_{nj,j}$ of job J_j.

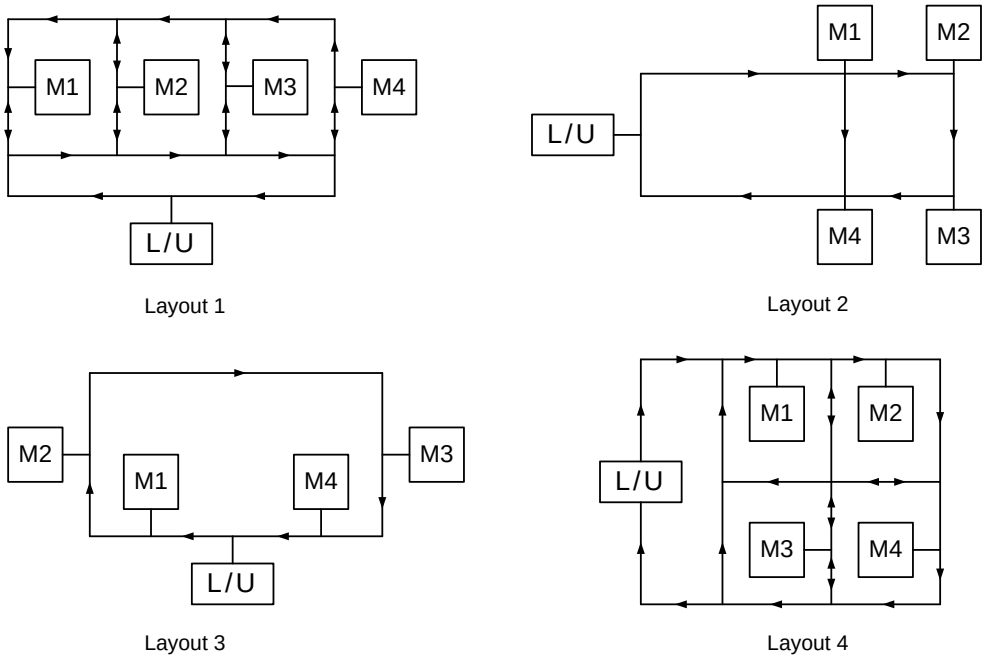


Fig. 7.1. Layout configurations used in the test example [1].

- To minimize the exit time of the last job from the system, where AGVs must return jobs to the unload station $C_{max-exit} = max_{j=1}^{n+1}\{C_j\}$, where C_j represents the completion time of the last transport operation $T_{\mu_{n_j,j},M_0}$, from the last processing operation $O_{nj,j}$ of job J_j to the L/U station M_0.

The problem is formulated under the following assumptions [1]:

- Machine operations and vehicle trips are non-preemptive, and there is sufficient I/O buffer space at each machine and L/U station to avoid deadlocks.
- Processing, loading and unloading times are deterministic and known in advance.
- The number of AGVs is known, and they initially start from the L/U station.
- Guide-path layout is given, and the guide paths can be either unidirectional or bidirectional. Travel times on each segment of the path are known.
- AGVs carry a single unit-load at a time, and they move along predetermined shortest paths, with the assumption of no delay due to congestion.
- Traffic control issues like conflicts and congestion, and other unexpected events like machine failure or downtime, scraps, rework, and vehicle dispatches for battery changer are ignored here.

Figure 7.1 shows the four different layout configurations for the FMS problem proposed by [1]. Each FMS layout is composed of four machines, one L/U station, and two AGVs. The job sets and travel times for the example are given in Appendices B.1 and B.2.

7.4 TCPN Modeling for SSMV Problem

This section briefly introduces and recalls the basic definitions and concepts of TCPN, followed by the TCPN models of the FMS problem using two different types of vehicle assignments. Interested readers are referred to [47, 174] for a complete tutorial on CPN formulations and theory.

7.4.1 TCPN Preliminaries

A CPN is a directed bipartite graph with two node types called places and transitions, where the nodes are connected via directed arcs. CPN allows the modularization of events and describes the set of logical relationships that determine the interaction between subsystems' components. A finite set of places $P = p_1, p_2, \ldots, p_q$ is used to specify the system components. In an FMS description, a place represents a resource or job status. Each resource or job is described by tokens in the place while a token consists of one or more colors describing the entity attributes, and carries a weight called cardinality. An FMS operation corresponds to as an event whether machining or transportation or AGV assignment. A finite set of transitions $T = t_1, t_2, \ldots, t_n$ describes the events (the start or completion) that may occur (or fire) based on the preconditions of input arc expressions and guards. Graphically, places, transitions, arcs, and guards are represented by circles, boxes, arrows, and square brackets respectively.

For performance evaluation and scheduling purposes, a CPN is extended with a time notion expressed by the introduction of a global clock. The global clock represents the model time, and each token has a time attribute called the time stamp. The time stamp describes the earliest time at which a token becomes available. A TCPN [175] is formally defined as a tuple $TCPN = (CPN, R, r_0)$ where CPN satisfies the requirements of a non-hierarchical CPN [174], R is a set of timed values called time stamps, a subset of $\mathbb{R}$ closed under + and containing 0 and r_0 is an element of R, called the start time. In a TCPN, transitions can be associated with a delay interpreted as production or transportation time in the FMS environment, represented as '@+time value'. Also, the CPN formalism allows time delays to be specified for places or arcs. This paper focuses on transition delays that makes the transition behave as an event with start and release times using the holding duration concept [71].

The current state of the system is defined by the distribution of tokens over the places called marking. A marking maps each place into a timed multi-set of token elements and a timed marking is a pair (m, r^*) which consists of the marking m together with the time stamps of the tokens and $r^* \in \mathbb{R}$ the value of the global clock [47]. The initial timed marking m_0 consists of the markings of each place in the model representing the initial state of the system. The untimed marking m_u of a marking m i.e. $m(m_u)$ is obtained by removing all the time stamps from the tokens in places.

A transition t is said to be time-enabled at time τ_k in a marking m denoted by $m[t\rangle_{\tau_k}$ if all the tokens to be consumed from the input places have a time stamp not later than time τ_k. The enabling time τ_k of a transition t is the maximum of all the time stamps of the tokens consumed [109]. If a transition t fires at time τ_k, it changes m to a new marking m' denoted by $m[t\rangle_{\tau_k} m'$. m' is said to be reachable from m. This means that the tokens are removed from the input places and added to the output places of the firing transitions. The number and color of the tokens are determined by the arc expressions, evaluated for the occurring bindings [174]. A transition delay applies to all output tokens created at transition firing. The time stamp of a token is defined at its generation time. Firing a transition t at time τ_k with a delay d, time stamps the output tokens with the time value $\tau_k + d$.

7.4.2 SSMV TCPN Models

While a job route (process plan) is specified for the machine scheduling problem, no prior vehicle route exists for the AGV scheduling. The first step is to determine how vehicles should be assigned to jobs or jobs to vehicles, called the vehicle assignment problem. This subsection proposes the two TCPN models that provide solutions to the assignment problem within the SSMV framework in which the vehicle assignment is either controlled by the machines, called machine-controlled SSMV (MCSS) or vehicles, called vehicle-controlled SSMV (VCSS) during the scheduling process. This paper employs an event-driven approach that does not take into account the number of vehicles available or the number of outstanding P/D requests, but rather the state. As such, heuristic rules for dispatching vehicles [176] may not be necessary in situations where a complete schedule needs to be produced off-line or in a rescheduling process if the scheduling algorithm can output the best schedule after evaluating all the possible combinations.

7.4.2.1 MCSS Model

Typically, a P/D request is generated whenever a machine unloads a part into its output buffer after processing. In this classical approach, an AGV waits for a P/D request from the machine before starting a trip. Figure 7.2 shows the TCPN model of the MCSS with the color set and variable definitions for job set 1 and layout 1. It consists of eight places ($P1, P2, \ldots, P8$) and four transitions ($T1, T2, \ldots, T4$). The interpretation of the places (including the conditional arc expression *T4P1*) and colors is given in Table 7.1. The arc expression *T4P1* describes the precedence constraints. It is not placed in the figure for legibility. Likewise, the initial tokens in places $P6$ ($MP6$) and $P7$ ($MP7$) are not added due to the large number of tokens in each place. For clarity, the standard timed multiset operator has been replaced with a single $+$ while logical operators like *andalso* and *orelse* by & and || respectively. The initial marking m_0 consists of 5, 2, 4, 25, and 13 tokens in $P1$ (jobs), $P2$ (AGVs), $P3$ (machines), $P6$, and $P7$ respectively while the others are empty. The time stamps of the tokens are written after the symbol @, and the global clock is at time 0. Places $P6$ and $P7$ are untimed since they contain the transportation times between machines and the processing times of each job operation. The same model can be used for different job sets and layouts by replacing the initial tokens in $P1$, $P6$, $P7$, and the arc expression *T4P1*.

Transition $T1$ describes the vehicle assignment event. It is an immediate transition without duration. Assignment depends on the availability of the AGVs in place $P2$ when the jobs in the output buffer of the machines require a P/D service. Also, a job must not have completed its last operation, described by the guard condition ($op < 4$). Since the assignment is machine-controlled, an idle AGV waits for a task assignment from the machine before embarking on a P/D trip.

Once the vehicle is assigned after firing transition $T1$, the AGV has to perform two trips depending on its current position (pos). In this case, the AGV is said to be in a busy state. First, it performs an empty trip (transition $T2$) either between the I/O buffers of the same machine or between the input buffer of one machine and the output buffer of another machine. It is then followed by a loaded trip (transition $T3$) from the machine that issues the service request. An empty trip is required between the I/O buffer of the same machine given that the distance can be non-zero [162]. The loaded travel time (d) in transition $T3$ includes both the loading and unloading time of the job. An AGV becomes available (output arc *T3P2* returning a token) as soon as it delivers the job at the input buffer of the destination machine (output arc *T3P4*). The AGV stays idle at the destination machine until another job request its service.

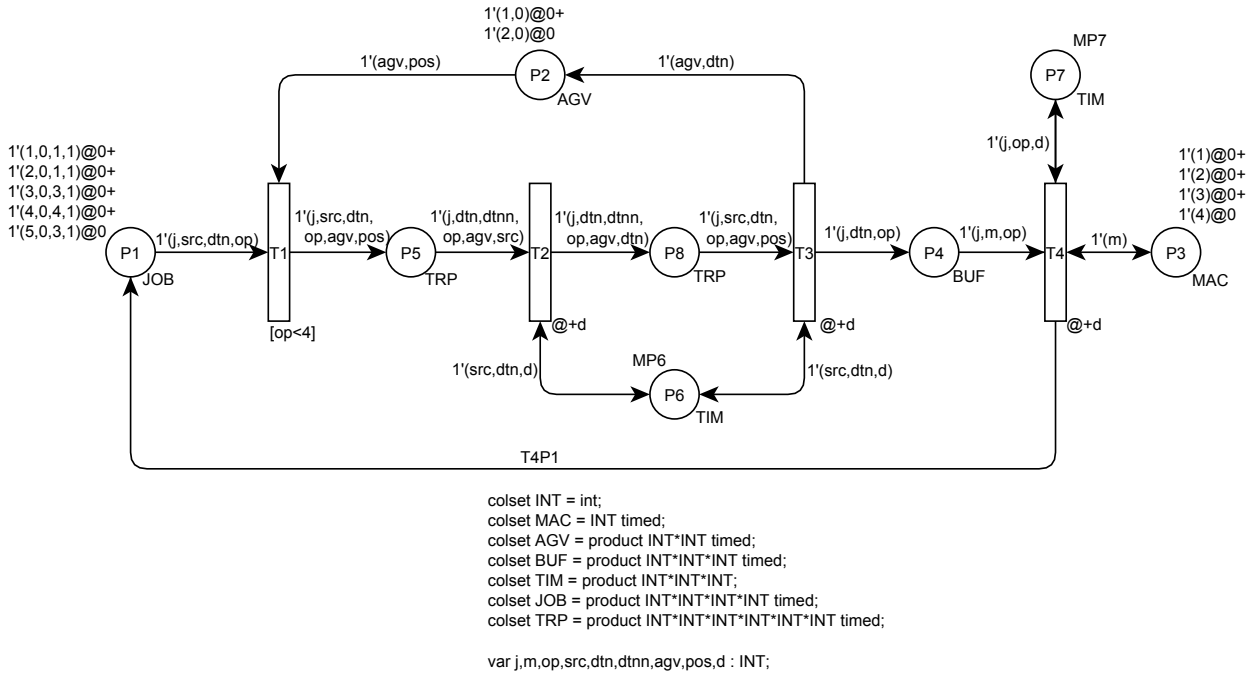


Fig. 7.2. The TCPN model of the MCSS for job set 1 and layout 1.

Table 7.1. Interpretation of places and colors in the MCSS model.

Place	Description	Color	Description
$P1$	Available jobs at the loading station for the first operation or at the output buffers of machines, for AGV assignment	j	Job identifier
$P2$	Available AGVs with current position in the layout	m	Machine identifier
$P3$	Available machines	op	Operation type identifier
$P4$	Jobs at the input buffer waiting to be processed by machine	src	Source machine
$P5$	Assigned AGV to pickup job	dtn,$dtnn$	Destination machine
$P6$	AGV transportation time matrix indicating the source and destination machines, and the travel time between the two	agv	AGV identifier
$P7$	The processing time of each job operation on machine	pos	AGV current position
$P8$	Assigned AGV to load job	d	Empty travel time (T2), loaded travel time (T3), machine processing time (T4)
$T4P1$	if $j = 1\&op = 1$ then $1'(j,m,2,2)$ else if $j = 1\&op = 2$ then $1'(j,m,4,3)$ else if $j = 2\&op = 1$ then $1'(j,m,3,2)$ else if $j = 2\&op = 2$ then $1'(j,m,2,3)$ else if $j = 3\&op = 1$ then $1'(j,m,4,2)$ else if $j = 3\&op = 2$ then $1'(j,m,1,3)$ else if $j = 4\&op = 1$ then $1'(j,m,2,2)$ else if $j = 5\&op = 1$ then $1'(j,m,1,2)$ else $1'(j,m,0,4)$		

Transition $T4$ represents the machine scheduling event. It describes execution of a job's operation in a machine with an assigned duration of $@+d$–the start and completion of the job processing. $T4$ is enabled if the machine assigned to process the job is free, and there are jobs waiting to be processed at the input buffer of the machine.

7.4.2.2 VCSS Model

Here, the decision of what machine to visit for a P/D task is determined by the vehicles. One of the advantages of this approach is the reduction of the idle time of vehicles. For instance, in a situation where there is no P/D request in the MCSS model often due to machining operations of jobs or all the jobs are in the input buffers of machines or jobs are being transported by other vehicles, an AGV stays idle until a part is deposited in the output buffer of a machine. Using the VCSS model, the time between the ready time of jobs for pickup and the vehicle assignment could be minimized if the AGVs can anticipate empty trips to machines with potential P/D request immediately after a loaded trip according to a defined parameter. As such, the MCSS becomes a subset of the VCSS ($MCSS \subseteq VCSS$) since the VCSS considers more alternatives that can be explored during scheduling.

Since the machines' buffers are the main source of P/D demands, the VCSS model predicts potential service request using the I/O buffer size as a parameter for assignment. Hence, a vehicle can be assigned to a machine with at least a part in its input or output buffer. Heuristic rules such as maximum buffer size, ready time of jobs are not considered as measures as all possible alternatives must be taken into account. An AGV can dispatch to any machine with a job waiting to be transferred or wait for a job that is currently being processed, without restrictions on time ordering or priority rules.

Figure 7.3 shows the TCPN model of the VCSS for the same job set and layout as in the MCSS. The places have the same meaning as those in the MCSS with the exception of $P5$ with a different color set. Place $P5$ (with five tokens) keeps the status of the buffers at each machine or L/U. Each token in $P5$ contain four colors with variables $< m/dtn, inp, out, ins >$ representing the source or destination machine, the input buffer, the output buffer, and the I/O buffer size synchronization between machine processing and AGVs' P/D service respectively. Also, place $P2$ now includes an additional color to specify an AGV state, described using color ctl. The arc expression *T3P1* is the same as *T4P1* in the MCSS model.

Transitions $T1$ and $T2$ are used for AGV scheduling while $T3$ for machine scheduling. Transition $T3$ performs the same function as $T4$ in the MCSS model with the addition of place $P5$ for updating the machine buffers before and after processing. For the makespan minimization objective, the output buffer is not updated for the last operation of the jobs (arc *T3P5*). No P/D service is required at this point since the objective function does not take into account the transportation of completed jobs to the L/U station. In this model, an AGV has two states: available for assignment after a loaded trip, $ctl = 0$, and busy, $ctl = 1$. Unlike the MCSS model, an AGV in a busy state can either be idling at a machine or performing a loaded trip.

The vehicle assignment is done by the AGVs via transition $T2$. Considering the buffer sizes (guard ($bin > 0$ or $bout > 0$)), an available AGV selects the next machine to visit by performing an empty trip immediately after the delivery of a job instead of staying idle at the last visited machine. As such, both the assignment and empty vehicle trips are concurrently executed using transition $T2$. The input buffer size is included as the assignment parameter for potential service request in order to handle the non-availability of jobs at the output buffers of machines.

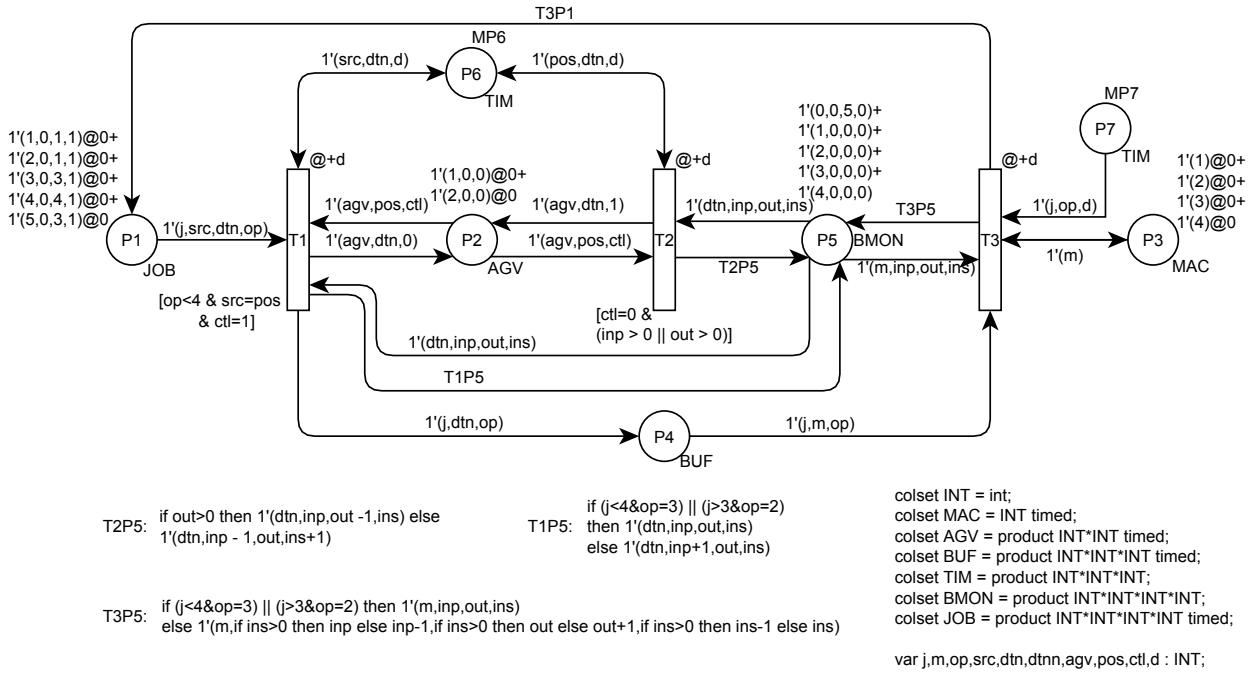


Fig. 7.3. The TCPN model of the VCSS for job set 1 and layout 1.

Transition $T1$ is mainly used for loaded vehicle trips. An assigned AGV picks up a ready job at the output buffer of the source machine, and transports it to the input buffer of the destination machine. The AGV is expected to be at the same machine as the job to be picked up (guard $src = pos$).

7.5 Heuristic Search for Timed State Space Exploration

The optimization process involves finding the optimal sequence of transition firings that will transform an initial marking m_0 to a given final or goal marking m_f. This is usually done by generating the reachability graph (or the state space) of a TCPN in order to evaluate all the different configurations of the FMS. For the makespan minimization (the completion time of the last job operation), the goal marking (without time stamps) for the MCSS model can be represented as: $m_f = P1\ \ 4'(*,*,*,4);\ \ P2\ \ 2'(*,*,0);\ \ P3\ \ 4'(*);\ \ P4\ \ empty;\ \ P5\ \ 1'(0,0,0) + 4'(*,*,*);\ \ P6\ \ 25'(*,*,*);\ \ P7\ \ 13'(*,*,*);\ \ P8\ \ empty$; (* means any color value). This means that all the jobs must have completed their operations and must be in the output buffer ($op = 4$ in place $P1$), the AGVs and machines are free ($P2$ and $P3$), there are no parts in the input buffer of machines ($P4$), and all the jobs must have been unloaded from L/U for processing.

A* explores the state space in a best-first order, and expands markings according to the heuristic function (f-cost); $f(m) = g(m) + h(m)$ where $g(m)$ is the actual makespan cost to reach marking m from the initial marking m_0, and $h(m)$ is an estimate on the remaining cost to reach the goal marking m_f from m. A* guarantees that the search always finds an optimal solution if $h(m)$ is admissible i.e. it is a lower bound that does not overestimate the cost to goal, $h(m) \leq h^*(m),\ \forall m$ where $h^*(m)$ is the cost of the optimal path from m to the final marking [41]. A* maintains two lists: the open list, and the closed list. The open list is implemented as a priority queue that stores the markings that have been generated but not yet expanded whereas the closed list which is usually

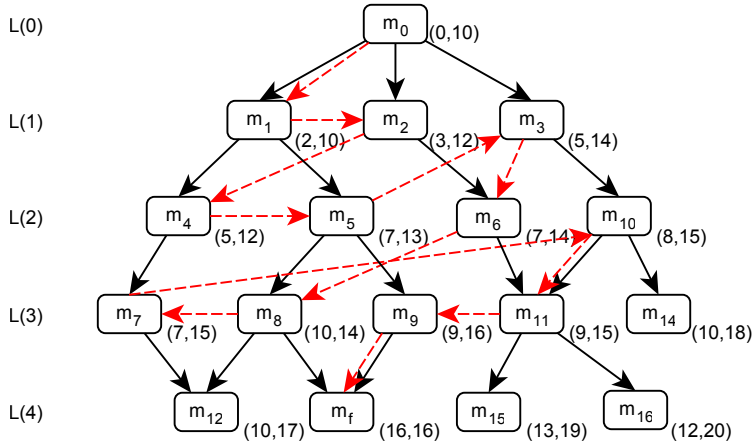


Fig. 7.4. The expansion behavior of A* in a breadth-first manner.

represented by a hash table, stores already-expanded (visited) markings. Expanding a marking involves the computation of its successors while a visited marking is one that has been expanded or encountered for the first time.

The performance of A* highly depends on the strength of the heuristic function. A tight lower bound function is usually needed so that an optimal solution can be reached quickly. Conversely, a strong heuristic function is usually too expensive to compute [117]. For the SSMV problem, it is quite difficult to obtain a strong heuristic function due to unknown vehicle routes. A weak function that is fast and easy to compute is considered acceptable for searching the state space. However, it may result to exploring a large number of markings leading to an increased computational effort.

Assume A* is explored in a breadth-first (BF) manner using a layered structure such that all the markings with the same minimum f-cost (f_{min}) in a layer $L(i)$ are expanded before proceeding to the next layer $L(i+1)$. $N = \bigcup_{i=0}^{d_{max}} L(i)$ where d_{max} is the maximum depth of the optimal solution path (the number of layers required to reach the optimal solution). f_{min} is the least f-cost of the open list . A layer $L(i)$ comprises the set of markings with an exact distance of i (the level index) from m_0. Using Fig. 7.4 as an example, A* starts expanding from m_0 with $g(m) = 0$ and $f(m) = 0$. It breaks ties between markings with equal f-cost using the $g(m)$ value. The red dotted arrow indicates the path taken by A* to reach m_f. Each time least f-cost value changes, the search considers another path with the f_{min} at the top of the priority queue. In a BF ordering, A* implicitly backtracks or performs a breadth-search to the layer having f_{min} in a situation where the f-cost of the successor markings at the currently-expanding layer $L(i)$ is greater than f_{min}. This behavior can be seen in paths (m_1, m_2), (m_5, m_3), (m_8, m_7), (m_7, m_{10}) etc. When frequent backtracking occurs, it may take a long computation time before a goal marking is reached.

Also, A* generates and stores all the successors of an expanded marking, some of which are never expanded since they have an f-cost value greater than the optimal schedule. These markings fill up the memory, and may prevent the search from reaching a goal marking if the search space runs out of memory. Examples of these markings are m_{12}, m_{14}, m_{15}, and m_{16} in Fig. 7.4.

We consider two admissible heuristic functions for the scheduling problem. The first one $h_1(m)$ sets $h(m) = 0$ i.e. $f(m) = g(m)$, and the other $h_2(m)$ is adapted from [26]; $h(m) = max_i\{\xi_i(m), i = 1, 2, \ldots, NR\}$ where $\xi_i(m)$ is the sum of operation times of those remaining operations for all jobs which are planned to be processed on the ith resource when the current system marking is represented by m. NR is the total number of machines. Although the AGV is a resource,

it cannot be used to compute $h(m)$ since the vehicle to be selected to perform the remaining material handling operations is not known in advance.

7.5.1 Hybrid Heuristic Search Algorithm

To enhance the efficiency of A*, the adapted ALS algorithm from [78] extends the A* explored in a layered manner with sBFBnB [77] to obtain a quick suboptimal solution. Backtracking is then used to find a stream of improving solutions until the search terminates with the optimal solution. The ALS algorithm was originally proposed for the breadth-first iterative deepening A* search. Instead of performing an immediate backtracking each time f_{min} changes, the ALS splits the state space into two parts controlled by two upper bounds, the A* f_{min} upper bound *mUB*, and sBFBnB suboptimal upper bound, *sUB*. The two upper bounds show a complementary behavior, more like that of IDA* f_{min} and DFBnB respectively [177]. Using [177] definitions, *mUB* is the lower bound on the cost to the optimal solution and increases in each iteration until it reaches the optimal solution, whereas *sUB* is the current best solution found so far, the upper bound on the cost to the optimal solution and decreases until it reaches the optimal solution.

The ALS performs successive iterations of A* and sBFBnB. In the first iteration, it starts with A* with $f(m_0)$ as the first mUB value. If the search is unable to find a goal marking with mUB i.e. A* expansion stops at a depth $d_{frontier}$ called the frontier layer, it continues to expand markings from $d_{frontier+1}$ using sBFBnB until a goal marking is reached. The search terminates if the f-cost of the goal marking is less than or equal to the f_{min} of any unexpanded marking in OPEN between $d = 0$ and $d_{frontier}$. At this point, the search is said to have converged, and an optimal solution has been reached. Otherwise, it backtracks to the deepest layer having the f_{min} to start another iteration. The search continues to find improved solutions until an optimal goal marking is reached.

In the upper part of the search space (layer $L(0)$ to $L(d_{frontier})$), markings are expanded with $f(m) = mUB$ according to A* while only markings with $f(m) < sUB$ are considered for expansion in the lower part from $L(d_{frontier+1})$ to $L(d_{max})$. New markings are generated using the earliest time state space (ESS) of a TCPN defined as $m[t\rangle_{\tau_k} m', \nexists \tau_{k'} < \tau_k : m[t\rangle_{\tau_{k'}}\}$ [110]. ESS enables an event-driven exploration of the timed state space without taking into account time constraints for transition firings. It does not restrict the number of markings to be explored, thereby, offering an optimality guarantee [71]. The frontier layer $d_{frontier}$ is the last depth of the A* search in the current iteration where the markings have successors with $f(m) > mUB$. After each iteration, mUB is set to the f_{min} of the OPEN list in the A* area. Also, sUB is used to periodically prune the state space when the incumbent sUB is improved. It removes markings with $f(M) \geq sUB$ to avoid keeping a large number of unexpanded markings that would not lead to an optimal solution. Like sBFBnB [77], sBFBnB expands markings in a BFS order, and uses the same heuristic function as A*. However, sBFBnB only considers the markings at the frontier of the A* i.e. it starts constructing its own search space from $d_{frontier} + 1$ and does not remove layers from memory.

The steps of the ALS algorithm are given as follows:

1. Set current layer index $i = 0$, $d_{frontier} = \infty$, and $d_{max} = \infty$.
2. Compute the f-cost of the initial marking $f(m_0) = h(m_0)$, set $mUB = f(m_0)$ and $sUB = \infty$.
3. Put the initial marking m_0 on the list CLOSED and its corresponding pointer (marking pointer on CLOSED, $g(m_0)$, $f(m_0)$) on list OPEN[i].
4. if $mUB \geq sUB$, terminate the search. Return the final best solution sUB obtained and its corresponding goal marking M_f.

5. Set upper bound UB to mUB.
6. If $i > d_{max}$ or the list OPEN[i] is empty, go to Step 11
7. For all markings m on list OPEN with $f(m) \leq UB$, do the following:
   (a) Remove the first marking m from OPEN[i].
   (b) Find the enabled transitions of m according to ESS $m[t\rangle_{\tau_k} m', \nexists \tau_{k'} < \tau_k : m[t\rangle_{\tau_{k'}}\}$ and generate the successor marking for each enabled transition.
   (c) For each successor m' of m, do the following:
      i. Compute $h(m')$ and $f(m')$.
      ii. If m' is a goal marking, do the following:
         A. If $f(m') \geq sUB$, go to Step 7c.
         B. Set $sUB = f(m')$ and construct the solution path from m' back to m_0.
         C. If $d_{max} = \infty$ or $d_{max} > i + 1$, set $d_{max} = i + 1$.
         D. Go to Step 7c.
      iii. If $f(m') < sUB$, do the following:
         A. If the untimed marking m'_u of m' is already on CLOSED, compare $f(m')$ with $f(m_{stored})$ of the existing marking m_{stored}. If $f(m_{stored})$ is lower, discard the new marking m'. If $f(m')$ is lower, replace m_{stored} with m' on CLOSED and on OPEN if it has not been expanded. If m_{stored} has been expanded, put m' on OPEN[i+1] and prune the descendants of m_{stored}. If $f(m')$ is equal to $f(m_{stored})$, compare their $g(m)$ values and follow the replacement steps for lower $g(m)$ if necessary. In case the tie continues, use the firing time τ_k and completion time $\tau_k + d$ of the last operation of both markings to break it.
         B. If the untimed marking m'_u of m' is not on CLOSED, put m' on CLOSED and its corresponding pointer on OPEN[i+1].
8. Increment layer index i by 1, $i = i + 1$.
9. If the least f-cost of OPEN[i] is greater than mUB, set $UB = sUB - 1$ and $d_{frontier} = i - 1$.
10. Go to Step 6.
11. Get the new value of mUB from the list OPEN[k], $\forall k \in [0, d_{frontier}]$.
12. Backtrack to the deepest layer l with the first marking from OPEN with mUB. Set $i = l$ and update $d_{frontier}$.
13. If sUB has improved in the last iteration, prune all the markings and their pointers from the lists CLOSED and OPEN respectively with $f(m) \geq sUB$.
14. Go to Step 4

The improved ALS algorithm uses a different data structure for the OPEN list by storing only the pointers of the markings in the CLOSED list. This is to avoid checking both the OPEN and CLOSED lists for duplicates. As a priority queue, the OPEN list incurs a high runtime overhead if it performs the simultaneous function of storing unique markings and sorting them based on f-cost. During the duplicate detection procedure (Step 7(c)iii), the algorithm uses the condensed state

space (CSS) method [81, 111] that factors out the notion of time when markings are compared. Here, the time stamps are ignored in the detection process to avoid keeping a potential set of duplicates with equivalent untimed markings but differing time evolutions. This method has been used to schedule FMS using TCPN and AI-based heuristic search in [80, 81].

One of the notable difference between the improved algorithm and that of [78] is the use of both $f(M)$ and $g(M)$ for the CSS duplicate detection procedure $CSS((f(M_{stored}), g(M_{stored})), (f(M'), g(M')))$ in order to provide a more accurate estimate in selecting the most promising time stamp set. Previous studies [80, 81] reveal that it is quite difficult to break ties using $g(M)$ as the criterion to discard untimed marking duplicates. Also, the myopic evaluation of $g(M)$ can prevent the search algorithm from obtaining the best path that leads to an optimal solution. Although a good lower bound $f(M)$ estimate is required for the CSS procedure.

The ALS algorithm corresponds to an A* search if it returns an optimal solution at the first iteration. The ALS algorithm is complete and optimal provided that: 1. $h(M)$ is admissible. 2. $g(M)$ does not discard markings leading to an optimal solution in the time stamp evaluation for CSS, and 3. The two upper bounds converge before memory runs out or before the search is terminated. To provide timely suboptimal solutions, we adopt the expansion width parameter ω in [78] to limit the number of markings to be expanded at each layer in each iteration.

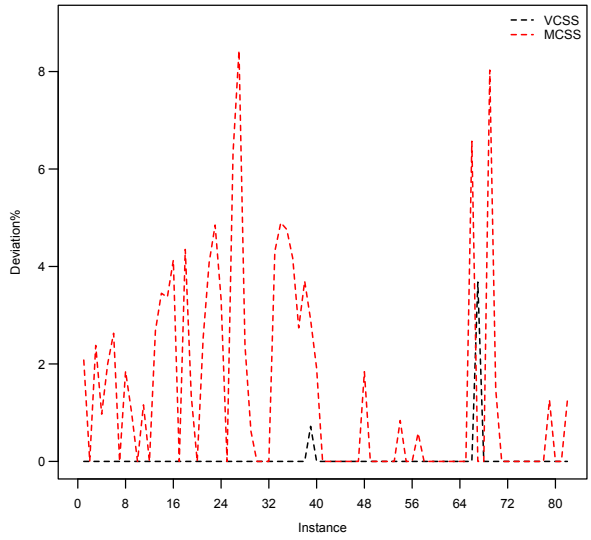
7.6 Experimental Results

Discrete event simulation can be conducted to evaluate the performance of TCPN models using the CPN Tools software [55]. Each simulation run corresponds to a path in the reachability graph. As such, the performance optimization of TCPN models with a large number of decision variables requires a large number of simulation runs [60]. Also, CPN Tools offers no support to integrate heuristic search methods, and its limitations to support TCPN models for performance optimization has been described in [71].

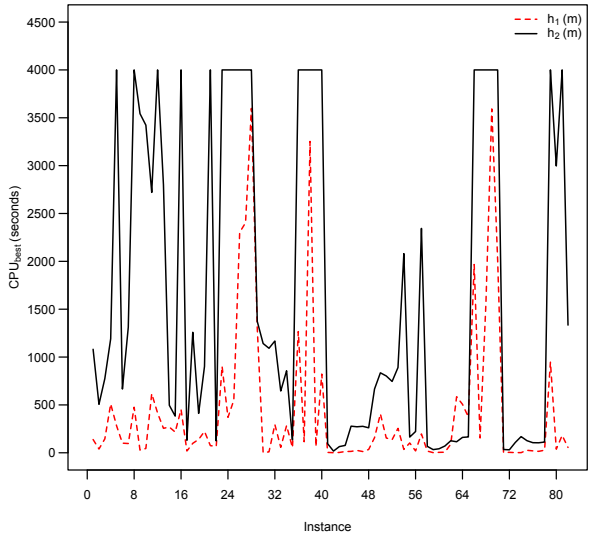
In light of this, a non-graphical TCPN tool called TIMSPAT (`http://grupsderecerca.uab.cat/timspat/`) is developed for the experiments, specifically designed for performance evaluation with heuristic search methods. It takes the specification of the TCPN model structure from ASCII files where the description of each transition (arc and guard expressions) is written into a separate file. The syntax description follows the standard rules of CPN formalism. Also, the initial and goal marking definition, and heuristic function are placed in a separate file.

The TCPN simulator together with ALS and A* algorithms were coded in C++. Several experiments were performed on the 82 test problems proposed by [1] in Fig. 7.1 on a 2.60GHz AMD Opteron processor PC with 4GB RAM. The CPU time limit was set to 3600s for the ALS algorithm, and an expansion width of 5 was selected for quick and improving solutions [81]. The test problems are grouped into two sets. The first set contains 40 instances whose t_{ik}/p_k ratios are greater than 0.25, and the other set contains 42 instances with t_{ik}/p_k ratios lower than 0.25. Each instance code is designated with prefix EX followed by two digits that indicate the job set and the layout. An extra digit of 0 or 1 is appended to the code of the second instance set. This implies that the process times are doubled or tripled respectively, where in both cases, travel times are halved.

To select the best SSMV model that offers a better performance, Fig. 7.5a gives the performance comparison between the MCSS and VCSS models on the 82 test instances using the ALS algorithm. The percentage deviation from the best known solution (BKS) is used to compare the solution quality. The deviation is calculated as: $(C_{best} - BKS)/BKS \times 100$. C_{best} represents the best solution obtained by the model. Clearly, the VCSS outperforms the MCSS in about 50% of the instance set.



(a) Performance comparison between MCSS and VCSS models in terms of solution quality using percentage deviation.



(b) CPU performance comparison between $h_1(m)$ and $h_2(m)$ for the best solution obtained within the given time limit.

Fig. 7.5. Performance comparison of assignment policies and heuristic functions.

The MCSS performs as good as the VCSS mostly in the second instance set (from instance number 41 to 82) with $t_{ik}/p_k < 0.25$, outperforming the VCSS in only one of the instances (EX720). As seen in Fig. 7.5a, the efficiency of the FMS is influenced by the vehicle assignment strategy implemented. The VCSS proves to be the best option, and it is used as the base model for subsequent experiments.

One of the aims of the proposed approach is to obtain the first solution quickly, and return improving solutions with less computation time. This is highly dependent on the heuristic function

Table 7.2. Performance comparison between A*-CSS and ALS algorithms for the instance set with $t/p > 0.25$.

Inst.	A*-CSS			ALS									
				First solution		Sol. at CPU (min)				Best solution		Convergence	
	Nmark	C_{max}	CPU	C_{first}	CPU	5	10	20	40	C_{best}	CPU_{best}	Nmark	CPU
EX11	560,956	96	615.2	154	0.1	**96**				**96**	138.5	828,467	800.8
EX12	464,803	82	511.0	129	0.2	**82**				**82**	39.2	832,435	800.6
EX13	528,081	84	578.7	120	0.2	**84**				**84**	145.1	792,098	762.5
EX14	602,510	103	652.9	202	0.1	104	**103**			**103**	510.2	870,520	832.8
EX21	2,724,500	100*	3702.2	197	0.2	100	–	–	–	100	282.4	–	–
EX22	850,640	76	1146.2	159	0.2	**76**				**76**	100.5	544,245	666.2
EX23	2,016,376	86	2752.3	147	0.2	86	–	–	–	86	96.6	–	–
EX24	3,500,335	108*	4487.7	218	0.2	112	108	–	–	108	475.9	–	–
EX31	3,035,665	99*	3966.2	161	0.2	99	–	–	–	99	27.7	–	–
EX32	2,704,103	85	3496.9	118	0.3	85	–	–	–	85	44.9	–	–
EX33	2,580,088	86	3386.7	108	0.2	88	87	86		86	617.3	–	–
EX34	4,201,454	111*	5499.5	199	0.2	113	111	–	–	111	414.9	–	–
EX41	3,862,536	112*	4361.2	213	0.2	112	–	–	–	112	255.4	–	–
EX42	3,272,458	87*	3672.5	121	0.2	87	–	–	–	87	268.7	–	–
EX43	3,246,865	89*	3639.2	132	0.2	89	–	–	–	89	216.5	–	–
EX44	3,870,393	121*	4456.3	189	0.2	126	121	–	–	121	452.0	–	–
EX51	690,141	88	751.9	158	0.1	**87**				**87**	18.4	891,855	857.1
EX52	478,605	69	529.0	103	0.3	**69**				**69**	98.7	897,285	869.7
EX53	578,850	74	640.6	122	0.1	**74**				**74**	139.4	861,335	821.7
EX54	712,713	96	779.8	191	0.1	**96**				**96**	223.2	935,923	885.0
EX61	10,340,913	118*	13823.8	214	0.2	118				118	74.7	–	–
EX62	4,569,909	98*	6199.1	161	0.2	98				98	66.6	–	–
EX63	6,803,150	103*	9104.9	153	0.2	104	104	103	–	103	902.6	–	–
EX64	10,485,849	120*	14041.4	284	0.2	123	120	–	–	120	370.2	–	–
EX71	o.o.m	o.o.m	o.o.m	222	0.2	114	111	–	–	111	549.3	–	–
EX72	o.o.m	o.o.m	o.o.m	173	0.2	87	87	87	85	79	2303.3	–	–
EX73	o.o.m	o.o.m	o.o.m	142	0.3	90	90	86	85	83	2403.3	–	–
EX74	o.o.m	o.o.m	o.o.m	300	0.2	131	131	129	128	126	3598.0	–	–
EX81	o.o.m	o.o.m	o.o.m	189	0.2	164	164	164	161	161	1300.6	–	–
EX82	o.o.m	o.o.m	o.o.m	193	0.2	151	–	–	–	151	2.7	–	–
EX83	o.o.m	o.o.m	o.o.m	167	0.3	153	–	–	–	153	9.3	–	–
EX84	o.o.m	o.o.m	o.o.m	296	0.2	163	–	–	–	163	295.8	–	–
EX91	1,862,100	116	2075.1	219	0.2	**116**				**116**	57.0	2,723,154	2721.4
EX92	1,260,261	102	1400.8	169	0.2	**102**				**102**	284.0	2,631,991	2643.0
EX93	1,475,114	105	1654.4	160	0.2	**105**				**105**	54.1	2,574,261	2602.3
EX94	2,008,506	120	2216.0	229	0.2	**120**				**120**	1266.5	2,827,318	2867.3
EX101	o.o.m	o.o.m	o.o.m	236	0.2	146	–	–	–	146	115.5	–	–
EX102	o.o.m	o.o.m	o.o.m	213	0.3	137	137	137	136	135	3252.9	–	–
EX103	o.o.m	o.o.m	o.o.m	211	0.3	139	–	–	–	139	66.6	–	–
EX104	o.o.m	o.o.m	o.o.m	355	0.2	159	159	157	–	157	822.2	–	–

–solution obtained by A with CPU > 3600s, Bold solution–converged within CPU time limit.
Inst.–instance, o.o.m–out of memory, Dash–no solution returned for the time interval.

that picks out the best ordering of markings in the search space. To select the heuristic function that best matches this purpose for the SSMV problem, we analyze the performance of the two heuristic

functions $h_1(m)$ and $h_2(m)$ on the two instance set. Fig. 7.5b compares the CPU performance when the best solution returned by the two functions matches. A value of $CPU = 4000\ s$ is used when a function fails to return the best solution as the other within the given time limit. For most of the instances, $h_1(m)$ produces more best solutions in lesser computation times than $h_2(m)$. $h_2(m)$ requires more CPU time to achieve the same solution quality as $h_1(m)$ especially for the first instance set. $h_1(m)$ obtained better solutions than $h_2(m)$ in 25 of the 82 instances within the same limit. For the SSMV problem, $h_1(m)$ proves to be the more effective function that produces quick and improving solutions for the ALS algorithm. As such, the type of heuristic function adopted plays a crucial role in determining the strength of the algorithm.

Table 7.2 compares the performance between the A*-CSS (A* based on condensed state space), and the ALS algorithms for the first instance set. We use the CSS approach to enhance the A* search efficiency as the conventional one could not solve any of the instances. The Nmark column indicates the number of markings expanded. Since the A*-CSS search returns only one solution at termination, we use $h_2(m)$ due to the slower convergence of $h_1(m)$ as observed in the table. The A*-CSS could only solve 16 instances within 1 hour of CPU, 12 instances over 1 hour, and it took over 3 hours to find the optimal solution for the EX6* instances. However, A*-CSS could not solve the larger instances EX7*, EX8*, and EX10*. The ALS algorithm obtained the same optimal solution as A*-CSS in relatively short computation times (less than 3 min), even though the ALS did not converge on time. As shown in the table, the capability of the ALS is not limited by the problem size. It guarantees that solutions are always returned irrespective of the problem size.

Table 7.3 shows the performance comparison of the ALS algorithm with four off-line and two on-line scheduling algorithms for test problems with $t/p > 0.25$. The off-line approaches are: sliding time window heuristic (STW) [1], two hybrid genetic algorithms, AGA [167] and PGA [152], and a hybrid local search with simulated annealing (SALS) [172]. The on-line algorithms are taken from Erol et al. [158]: a multi-agent system (MAS), and the best performing dispatching rule, shortest traveling distance (STD) in the comparison made by [158]. The percentage deviation (Dev%) from the best known solution is used to measure the solution quality. ALS found two new best known solutions and outperforms the other algorithms with the exception of EX103. The SALS algorithm proves to be the most efficient evolutionary algorithm for the first instance set. More recently, [164] obtained a new solution for EX103 with makespan of 137 using tabu search. The results obtained by the on-line algorithms (MAS and STD) could not compete with ALS. Almost all the solutions were significantly outperformed by ALS in less than 3 min given the time intervals in Table 7.2. Computation times were not reported in [158]. As such, the CPU times cannot be compared. This demonstrates the suitability of the ALS algorithm for on-line scheduling, and the superiority of the VCSS model over dispatching rules. Fig. 7.6 shows the Gantt chart of the new best known solution obtained for EX104 instance. For the second instance set ($t/p < 0.25$), the ALS obtains the best known solutions for all instances and solves most of them in less than 3 min with an average of about 5 min across the 42 test instances.

The $C_{max-exit}$ minimization criterion has been rarely studied in the literature. So far, only two works have solved the SSMV instances considering the exit time of the last job: the SALS algorithm, and the memetic algorithm (MEMA) proposed by [166]. Table 7.4 gives the performance comparison of the results obtained for the 40 instances of the first instance set. For the ALS algorithm, the table summarizes the solutions produced by presenting the exit time value and CPU of the first and best solutions. The ALS outperforms SALS and compares favorably with MEMA. Out of the 40 instances, the proposed approach found 35 best known solutions and two new solutions. However, the ALS needed over 1h CPU time to obtain the best known solutions for 5 instances.

In general, an off-line schedule remains acceptable as long as the operating conditions of the

Table 7.3. Performance comparison of ALS with existing approaches for problems with $t/p > 0.25$.

Inst.	STW		AGA		PGA		SALS		MAS		STD		ALS		
	C_{best}	Dev%	C_{best}	Dev%	C_{best}	Dev%	C_{best}	Dev%	C_{best}	Dev%	C_{best}	Dev%	mUB	C_{best}	Dev%
EX11	96	0	96	0	96	0	96	0	130	35.42	126	31.25	95	96	0
EX12	82	0	82	0	82	0	82	0	98	19.51	104	26.83	81	82	0
EX13	84	0	84	0	84	0	84	0	109	29.76	110	30.95	83	84	0
EX14	108	4.85	103	0	103	0	103	0	168	63.11	164	59.22	102	103	0
EX21	105	5.00	102	2.00	100	0	100	0	143	43.00	147	47.00	59	100	0
EX22	80	5.26	76	0	76	0	76	0	86	13.16	104	36.84	47	76	0
EX23	86	0	86	0	86	0	86	0	98	13.95	118	37.21	49	86	0
EX24	116	7.41	108	0	108	0	108	0	169	56.48	172	59.26	66	108	0
EX31	105	6.06	99	0	99	0	99	0	142	43.43	138	39.39	58	99	0
EX32	88	3.53	85	0	85	0	85	0	114	34.12	116	36.47	48	85	0
EX33	86	0	86	0	86	0	86	0	103	19.77	126	46.51	48	86	0
EX34	116	4.50	111	0	111	0	111	0	167	50.45	182	63.96	66	111	0
EX41	118	5.36	112	0	112	0	112	0	198	76.79	220	96.43	74	112	0
EX42	93	6.90	88	1.15	87	0	87	0	129	48.28	151	73.56	58	87	0
EX43	95	6.74	89	0	89	0	89	0	155	74.16	143	60.67	63	89	0
EX44	126	4.13	126	4.13	126	4.13	121	0	242	100	247	104.13	85	121	0
EX51	89	2.30	87	0	87	0	87	0	130	49.43	124	42.53	86	87	0
EX52	69	0	69	0	69	0	69	0	98	42.03	101	46.38	68	69	0
EX53	76	2.70	74	0	74	0	74	0	109	47.30	103	39.19	73	74	0
EX54	99	3.13	96	0	96	0	96	0	168	75.00	168	75.00	95	96	0
EX61	120	1.69	118	0	118	0	118	0	153	29.66	162	37.29	56	118	0
EX62	100	2.04	98	0	98	0	98	0	123	25.51	135	37.76	45	98	0
EX63	104	0.97	104	0.97	103	0	103	0	128	24.27	143	38.83	46	103	0
EX64	120	0	120	0	120	0	120	0	189	57.50	190	58.33	60	120	0
EX71	119	7.21	115	3.60	111	0	111	0	129	16.22	143	28.83	40	111	0
EX72	90	13.92	79	0	79	0	79	0	92	16.46	109	37.97	28	79	0
EX73	91	9.64	86	3.61	83	0	83	0	93	12.05	109	31.33	30	83	0
EX74	136	7.94	127	0.79	126	0	126	0	156	23.81	173	37.30	47	126	0
EX81	161	0	161	0	161	0	161	0	196	21.74	217	34.78	56	161	0
EX82	151	0	151	0	151	0	151	0	172	13.91	180	19.21	46	151	0
EX83	153	0	153	0	153	0	153	0	172	12.42	182	18.95	46	153	0
EX84	163	0	163	0	163	0	163	0	251	53.99	246	50.92	66	163	0
EX91	120	3.45	118	1.72	116	0	116	0	178	53.45	163	40.52	115	116	0
EX92	104	1.96	104	1.96	102	0	102	0	123	20.59	128	25.49	101	102	0
EX93	110	4.76	106	0.95	105	0	105	0	119	13.33	132	25.71	104	105	0
EX94	125	4.17	122	1.67	122	1.67	120	0	181	50.83	190	58.33	120	120	0
EX101	153	4.79	147	0.68	147	0.68	147	0.68	188	28.77	193	32.19	57	146*	0
EX102	139	2.96	136	0.74	135	0	135	0	154	14.07	164	21.48	51	135	0
EX103	143	3.62	141	2.17	139	0.72	138	0	158	14.49	180	30.43	52	139	0.72
EX104	171	8.92	159	1.27	158	0.64	159	1.27	246	56.69	249	58.60	64	157*	0

*–new solutions.

system do not change [86]. However, whenever unexpected events or disturbances occur such as delays in machine processing, disruptions on the transport network or status changes due to machine breakdown or arrival of new jobs, the system deviates from the original schedule. As a result, a change in the currently implemented schedule is required. Off-line scheduling, considered

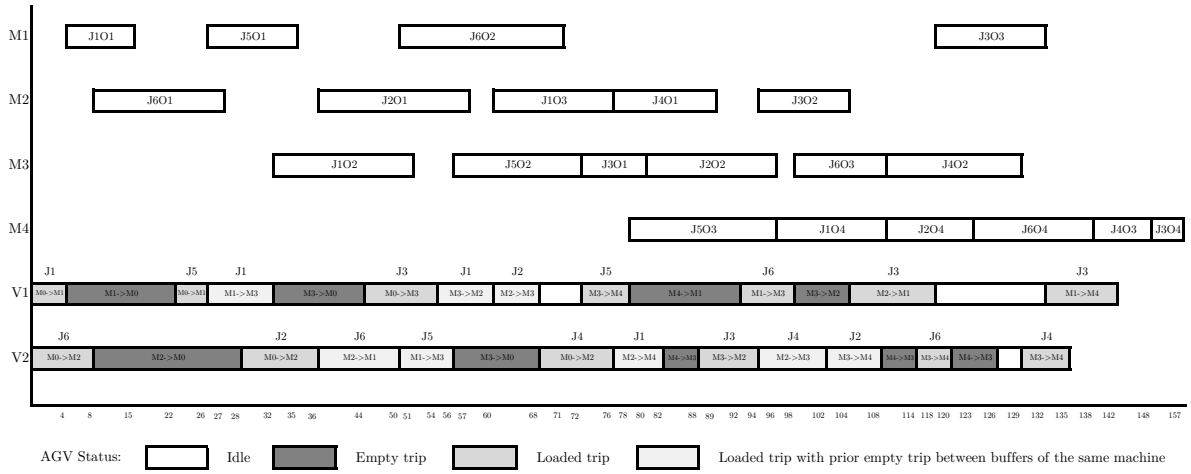


Fig. 7.6. Gantt chart of the new best known solution for EX104 instance.

an open-loop control strategy, may not be robust enough to handle these changes. The proposed approach supports a closed-loop solution that implements an integrated scheduling and control scheme [59, 178] through a hybrid off-line/on-line (scheduling/rescheduling) procedure. Here, the scheduling problem is solved iteratively as different instances of off-line scheduling, taken into account the current shop floor status. This procedure requires an interface with the control system.

1. Given the TCPN model and an initial marking of the system, solve the scheduling problem using the ALS algorithm without CPU time constraints or with the CPU time required before the start of activities.
2. Deliver the SSMV schedule to the shop floor.
3. In an event of disturbance or disruption, get the current state of the system and send it as an initial marking to the ALS algorithm.
4. Execute the ALS algorithm with CPU time limit to generate a new schedule.
5. Repeat Steps 2-4.

7.7 Conclusion

We have presented a TCPN-based approach to solve the SSMV problem. The scheduling technique implements a hybrid heuristic search algorithm based on TCPN modeling by combining the A* search with sBFBnB and backtracking. Two vehicle assignment methods were considered and their impact was analyzed on the overall performance of the FMS. The proposed algorithm have been shown to outperform conventional heuristic search algorithms in terms of providing a quick first solution and improving ones at different time intervals. The consideration of the vehicle-controlled method in the modeling approach makes it more efficient for vehicle assignments over heuristic dispatching rules for the analyzed systems. The proposed approach is different from the existing ones in that it can be adapted to both off-line and on-line scheduling problems without

Table 7.4. Performance comparison for the first instance set with $t/p > 0.25$ based on the $C_{max-exit}$ criterion.

Instance	SALS		MEMA		ALS				
	C_{best}	Dev%	C_{best}	Dev%	C_{first}	CPU_{first}	C_{best}	Dev%	CPU_{best}
EX11	114	0	114	0	162	0.18	**114**	0	14.6
EX12	90	0	90	0	132	0.19	**90**	0	223.5
EX13	98	0	98	0	144	0.2	**98**	0	40.6
EX14	140	0	140	0	203	0.17	**140**	0	3.5
EX21	116	0	116	0	169	0.25	116	0	167.6
EX22	82	0	82	0	145	0.25	82	0	53.4
EX23	89	0	89	0	132	0.25	89	0	656.5
EX24	134	0	134	0	185	0.23	134	0	407.6
EX31	121	0	121	0	191	0.25	121	0	122.1
EX32	89	0	89	0	163	0.23	89	0	130.2
EX33	96	0	96	0	127	0.30	96	0	101.8
EX34	148	0	148	0	196	0.27	148	0	352.7
EX41	138	1.47	138	1.47	168	0.25	136*	0	316.6
EX42	100	0	100	0	133	0.28	100	0	89.3
EX43	102	0	102	0	156	0.25	102	0	525.8
EX44	163	0	163	0	198	0.25	163	0	226.5
EX51	110	0	110	0	146	0.17	**110**	0	8.8
EX52	81	0	81	0	116	0.19	**81**	0	116.8
EX53	89	0	89	0	109	0.19	**89**	0	154.3
EX54	134	0	134	0	194	0.16	**134**	0	120.3
EX61	129	0	129	0	170	0.31	130	0.78	314.7
EX62	102	0	102	0	141	0.31	102	0	74.4
EX63	105	0	105	0	142	0.31	105	0	2661.4
EX64	153	1.32	151	0	230	0.25	151	0	471.9
EX71	135	1.50	134	0.75	192	0.34	133*	0	1187.6
EX72	86	0	86	0	137	0.39	87	1.16	**4776.2**
EX73	93	0	93	0	146	0.36	95	2.15	601.1
EX74	161	0	161	0	211	0.36	161	0	**6314.1**
EX81	167	0	167	0	255	0.31	167	0	**6138.7**
EX82	155	0	155	0	171	0.31	155	0	6.4
EX83	155	0	155	0	169	0.34	155	0	25.8
EX84	178	0	178	0	268	0.28	178	0	798.8
EX91	129	1.57	127	0	178	0.25	127	0	63.6
EX92	106	0	106	0	151	0.27	106	0	916.9
EX93	107	0	107	0	155	0.25	107	0	99.3
EX94	149	0	149	0	190	0.22	149	0	44.7
EX101	153	0	153	0	226	0.34	153	0	384.2
EX102	139	0	139	0	220	0.34	139	0	**4722.2**
EX103	141	1.44	139	0	179	0.39	139	0	**6996.6**
EX104	183	0	183	0	270	0.38	183	0	2890.6

*–new solutions, bold C_{best}–convergence, bold CPU_{best}–over 1h.

losing optimality guarantee provided that certain conditions are met. Besides the computational time reduction, the modeling aspect has also proved to be useful in describing the dynamics and interdependencies between the machining and AGV systems as a single and compact model. The

benefit of the proposed framework is that it can be used to solve most scheduling problems without requiring a change in the algorithm.

The algorithm has been tested on a number of test problems with different job sets and layouts. The experimental results demonstrate the importance of obtaining high quality solutions at reduced computation times as opposed to waiting until the termination of the search algorithm. This can be of benefit to industrial practitioners aiming to provide solutions to real-time problems at the slightest possible time. The ALS algorithm competes favorably with the evolutionary algorithms without the need to tune parameters or carry out random searches at different times.

The timeliness of the improving solution produced by the ALS algorithm is sensitive to the problem size. The frequency of returning improving solutions for large-sized problems is influenced by the variables involved in the enabling and firing transitions, and the expansion width used. Also, it is difficult to ascertain whether or not the search algorithm has reached the optimal solution when no new solutions are returned for a large amount of time. Even though the optimal solution has been found, the algorithm must explore the remaining markings with f-cost values between mUB and sUB in order to verify the convergence of the last solution obtained. The computation and convergence times can be further improved by parallelizing the search on a number of processors or workstations. Its application to SS problems with conflict-free routing of AGVs and limited buffer capacity will be investigated in a future work.

8

Simultaneous Scheduling of Machines and AGVs with Conflict-free Routing (SSMV-CFR)

8.1 Introduction

For the successful application of an integrated schedule in some production facilities, the specific paths to be taken by the AGVs for P/D requests must be explicitly defined in the routing problem of the FMS model. The SSMV problem description used by most works in the literature assumes a direct and conflict-free path between the source and the destination machines, ignoring the possibility of collisions, transportation delays, and congestion on the guide-path layout. This assumption may be less realistic in FMS with multiple AGVs where the transportation times depend not only on the capacity of the nodes and path segments on the guide-path network, but also on the movement of other AGVs within the same network. Hence, the SSMV schedule becomes infeasible if the traffic conditions resulting from the physical constraints are not taken into account. The travel time of AGVs include the actual travel time of the selected path and the delays encountered along the path due to the traffic conditions [179]. As a result, controlling the movement of the vehicles on the guide path becomes critical to the overall performance of the system.

Notwithstanding, the SSMV assumptions are suitable for situations where: multiple lane guide paths exist between nodes that allows for simultaneous travel of more than one AGV in the same direction, employing adequate separations; buffering areas are available for traveling AGVs in which all nodes have facilities for holding blocked vehicles; layouts are designed to avoid collisions and deadlocks [180]; or in flexible manufacturing cells [8].

8.2 Problem Description

The guide-path layout is modeled as a graph consisting of a set of nodes and edges (or arcs). The nodes represent the P/D stations (machines, L/U) and intersection points, while the edges are the directed path segments connecting the nodes and indicating the travel directions. The guide path can be either unidirectional or bidirectional or mixed. The vehicle routing problem involves the selection of the path to be taken by the AGV from its current node to its assigned destination node (given by the P/D assignment), and the control of the AGV movement as it travels along the path. The path selection depends primarily on the layout and it specifies the sequence of nodes to be visited by the AGV to reach its destination. The routing objective is to minimize the travel time of each P/D request. Two types of routing strategies have been considered: static, and dynamic [180]. Here, we focus only on static routing which assumes that the AGVs always follow a fixed shortest path route between any two P/D stations.

While moving through the selected path, an AGV may block or collide with another in a multi-AGV system, which can propagate delays to other parts of the system. Blocking occurs when a loaded AGV encounters an idle AGV on its traveling route. Also, collision may occur if two vehicles attempt to occupy the same path segment. The possible conflicts that may arise in the system include: head-on collision, two AGVs traveling in opposite directions on a segment; head-to-tail collision, two AGVs moving in the same direction at different speeds; and collision at junction, two AGVs moving toward the same node from different directions [31]. A control strategy is needed to coordinate the movement of AGVs to ensure a conflict-free route.

For collision avoidance, we adopt the zone control technique [181, 182] that divides the guide-path layout into several disjoint zones with restrictions on the movement of vehicles. Here, a zone is associated with each edge on the layout. The zone control constraints are: 1. Only one vehicle can travel on an edge at a time, and 2. Each node can only be occupied by one vehicle. The Gantt chart in Fig. 7.6 and those in [172] will lead to collisions given these constraints. In addition to

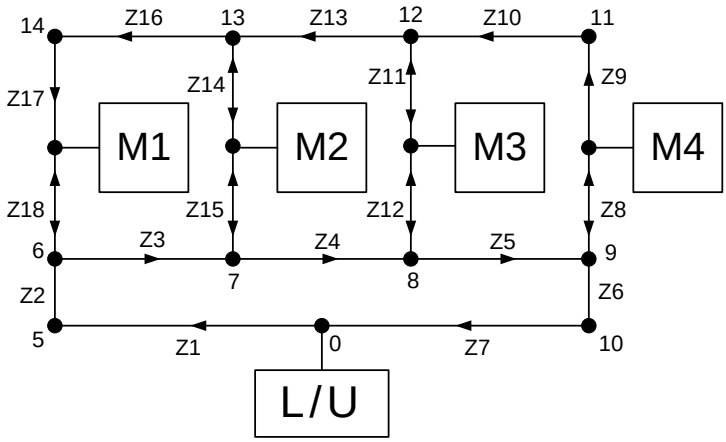


Fig. 8.1. A mixed guide-path layout showing the node numbers and zone specification.

the above constraints, the following assumptions are made:

- Each node has a unit-capacity including the L/U station node.
- The traveling time on each edge is known and it is the same for both loaded and empty AGVs.
- No buffering area exists for idle vehicles.
- The loading and unloading times of vehicles are included in the travel times.

Incorporating the conflict-free routing of AGVs makes the scheduling problem even more difficult to solve. The SSMV-CFR problem includes the SSMV, the specification of the conflict-free route path for each P/D request, and the determination of arrival and departure times of AGVs at each node in the conflict-free route path with the objective of minimizing the makespan of the overall schedule. Figure 8.1 shows the layout of the problem considered (layout 1 in Fig. 7.1) with the node numbers and zone specification. The travel time on each edge is 2 time units.

8.3 Related Work

Due to the scheduling complexity, the SSMV-CFR problem has received little attention in the literature. The only work that has attempted to solve this problem from the PN domain dates back to 1994. Sun et al. [138] propose a Limited-Expansion A* algorithm for the SSMV-CFR of a prototype AGV-served FMS that includes the conflict-free routing of 2 AGVs on a simple unidirectional guide-path layout of 5 nodes. They use the zone control approach and a push-AGV strategy to avoid collisions and blocking respectively.

The other two works based on mathematical modeling [163, 183] employ a decomposition framework different from the conventional ones used in solving the SSMV problem due to the increase in the number of decision variables. Nishi et al. [163] propose a bilevel decomposition algorithm that solves the machine scheduling and AGV task assignment as a master problem at the upper level using Lagrangian relaxation technique. At the lower level, a distributed routing algorithm finds the conflict-free routing of the upper level solution. Saidi-Mehrabad et al. [183] use a similar approach with a 2-stage ant colony algorithm with the objective of minimizing the makespan. To the best of our knowledge, this is the first study that applies a TCPN-based approach to solve the SSMV-CFR problem.

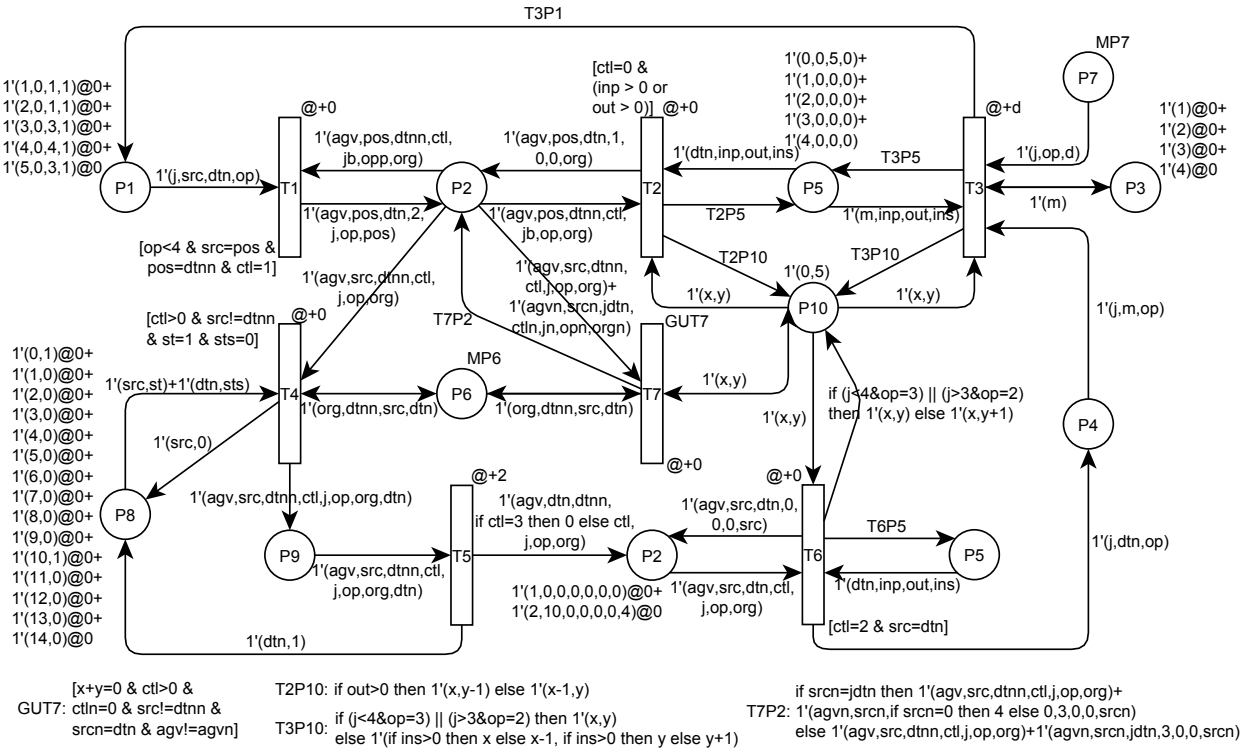


Fig. 8.2. The TCPN model of the MCSS for job set 1 and layout 1.

8.4 SSMV-CFR TCPN model

Using the zone control technique, we develop a new TCPN given in Fig. 8.2 by expanding the VCSS model to include the description of the guide-path network and the movement of the AGVs between zones. The meaning of the new and modified places and colors is given in Table 8.1. Places $P2$ and $P5$ are duplicated for clarity. The arc expression $T6P5$ is equivalent to $T1P5$ in Fig. 7.3, while $T2P5$ and $T3P5$ remain the same. Each node in the guide path is modeled as a resource in place $P8$, and a zone is described with two nodes denoting the entry and exit points (arc expression $P8T4$). The initial marking m_0 has 5 jobs at the L/U station ($P1$) and 2 AGVs stationed at nodes 0 (L/U) and 10 respectively.

In the TCPN, an AGV can be in one of the four states specified with color $ctl \in \{0, 1, 2, 3\}$ indicating idle, pickup, delivery, and push-move. Unlike the net in Fig. 7.3, transition $T1$ describes only the pickup operation of jobs, for assigned AGVs at transition $T2$ that have traveled through the designated zones with transitions $T4$ and $T5$ to the pickup station (guard $src = pos$).

An AGV moves from one source node to the other using transition $T5$. Before it can travel on a selected path segment, the AGV needs to acquire the control right of the zone (transition $T4$) based on the status of the neighboring nodes. To reserve the path, the AGV checks to make sure that the next adjacent node dtn on the route path given in the routing table ($P6$) is free i.e. $sts = 0$. If the guard conditions are met, the AGV leaves the current node (the firing of $T4$) and starts moving to the next node (place P_9). Otherwise, it waits until the next node is free or may decide to move to another adjacent node provided an alternative path exists. During this operation, the current node is freed (arc expression $T4P8$) to inform the other AGVs of its availability. Once the move is completed (by firing transition $T5$), the destination node dtn status st is set to occupied (1).

Table 8.1. The interpretation of the new and modified places and colors in the VCSS-CFR model.

Places	Color set	Colors	Descriptions
$P2$	$AGV = product\ INT * INT * INT * INT * INT * INT * INT\ timed$	$\langle agv, pos, dtnn, ctl, j, op, org \rangle$	Vehicle agv at current node pos traveling to destination node $dtnn$ to pickup or deliver $ctl = 1, 2$ a job j with operation op from P/D location org or a vehicle at idle or push-move state $ctl = 0, 3$
$P6$	$BMON$	$\langle org, dtnn, src, dtn \rangle$	The routing table of the guide-path layout from a source P/D station org to a destination P/D station $dtnn$ given the zone node points src and dtn to traverse.
$P8$	$NOD = product\ INT * INT\ timed$	$\langle src, st \rangle$	Status st (free=0 or occupied=1) of node src
$P9$	$TRP = product\ INT * INT * INT * INT * INT * INT * INT * INT\ timed$	$\langle agv, pos, dtnn, ctl, j, op, org, dtn \rangle$	A traveling vehicle agv for P/D ($ctl = 1, 2$) leaving node pos to the next adjacent node dtn of the zone to traverse
$P10$	$BF = product\ INT * INT$	$\langle x, y \rangle$	Monitors the total number of P/D requests x in the input buffers and y in the output buffers of machines and L/U

With the active assignment policy of the VCSS, AGVs are expected to pick up request immediately after delivery. An AGV starts traveling for the next assignment provided there are P/D requests to be served. In this situation, blocking is expected to minimally occur in layouts without buffering areas for idle AGVs. To avoid blocking, we adopt the push-AGV control strategy in [138]. This strategy (modeled as transition $T7$) allows a traveling AGV to issue a push request to an idle AGV on its path, instructing the idle AGV to leave the occupied node and move to the next adjacent one. Frequent push requests can be issued if vehicles' states do not change instantaneously after a delivery service. This may result in the complete blockage of the system (deadlock). To avoid this problem, a push request is only issued if there is no workload in the system (place $P10$ and guard $x + y = 0$) i.e. the I/O buffers of machines and loading station are empty.

In spite of its advantages, a zone-controlled AGV system is still vulnerable to deadlocks [181]. This can prevent the scheduling algorithm from producing a feasible schedule in the static routing strategy where no alternative paths are provided. Deadlock can occur if an AGV tries to move to a node currently occupied by another AGV whose fixed routes are in either direction. For example, the two bidirectional path segments in Fig. 8.1 connecting nodes 1 and 6, and 4 and 9, are sources of deadlocks for vehicles traveling in opposite directions. To minimize deadlock and make the TCPN valid for static routing, zones $Z2$ and $Z18$ are merged into one zone. The same modification applies to zones $Z6$ and $Z8$. Overall, the routing module of the model can be adapted to any number of AGVs by modifying the tokens in place $P2$ as well as any layout type by replacing the tokens representing the routing table in place $P6$.

Table 8.2. Computational results for the SSMV-CFR problem instances with 2, 3, and 4 vehicles.

Instance	Number of vehicles												
	2					3				4			
	C_1	CPU_1	C_b	CPU_b	BKS-SSMV	C_1	CPU_1	C_b	CPU_b	C_1	CPU_1	C_b	CPU_b
EX11	114	37.8	97	4410.2	96	99	1.7	90	50.0	98	60.6	80	181.1
EX21	130	1.5	106	2270.6	100	107	54.4	94	226.8	110	68.8	90	1414.2
EX31	152	4.6	108	384.9	99	121	69.0	98	132.2	116	79.3	98	3801.0
EX41	144	1.6	117	1399.0	112	112	75.4	100	1212.0	100	92.2	94	1588.5
EX51	116	35.1	89	696.6	87	85	1.5	74	3042.5	84	56.1	68	557.8
EX61	143	3.6	122	296.2	118	121	71.0	116	1805.6	142	127.3	116	1149.1
EX71	146	2.0	120	1474.7	111	134	14.4	104	2935.3	108	48.3	104	379.9
EX81	171	62.8	169	62.8	161	163	30.7	163	30.7	173	76.6	163	792.5
EX91	150	1.9	119	1096.5	116	119	69.1	112	277.3	112	83.3	108	1223.2
EX101	172	2.4	156	4695.6	146	156	96.6	146	1618.4	159	17.7	148	416.9

C_1 – first solution, C_b – best solution, CPU_1 – CPU time of first solution, CPU_b – CPU time of best solution.

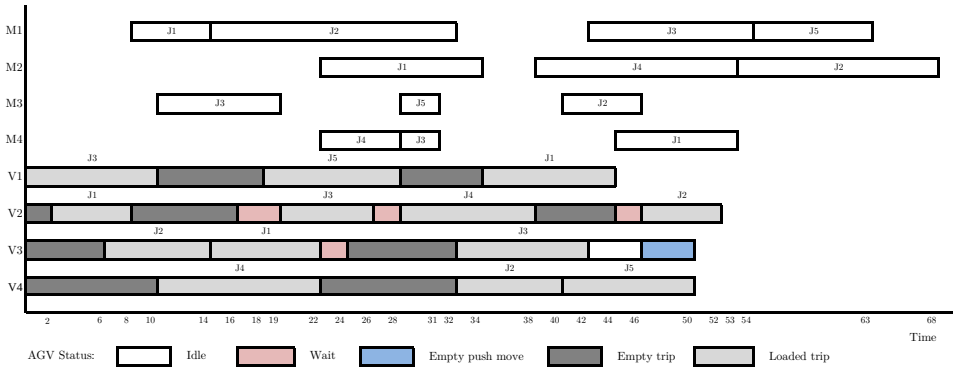
8.5 Deadlock-free Heuristic Search Algorithm

There are two approaches to handling deadlocks in PN-based scheduling [78]: conservative approach, and optimistic approach. The first approach integrates deadlock control policies to make the PN live (deadlock-free) before scheduling. The downside is that it is system specific and does not guarantee optimality since restrictions are imposed on the system evolution. On the other hand, the optimistic approach performs scheduling on deadlock-prone PNs. In contrast, it does not rely on control policies but on the effectiveness of the search algorithms to find optimal or near optimal schedule that is deadlock-free. This approach removes the overhead to guarantee the liveness of the PN [41] before scheduling is performed. As such, deadlock control policies are not a requirement to schedule a deadlock-prone FMS. Provided the system is accurately described by the TCPN model (resource sharing, precedence relations, technological constraints, etc.), the defined goal marking guarantees that a deadlock-free schedule is obtained. The generated schedule from the firing sequence ensures that deadlock is avoided in the operation of the FMS if the schedule is followed. For deadlock-free scheduling, we use the optimistic approach given in [78] where deadlocks are allowed occur and are detected during the timed state space generation, and measures are put in place to prevent its future generation.

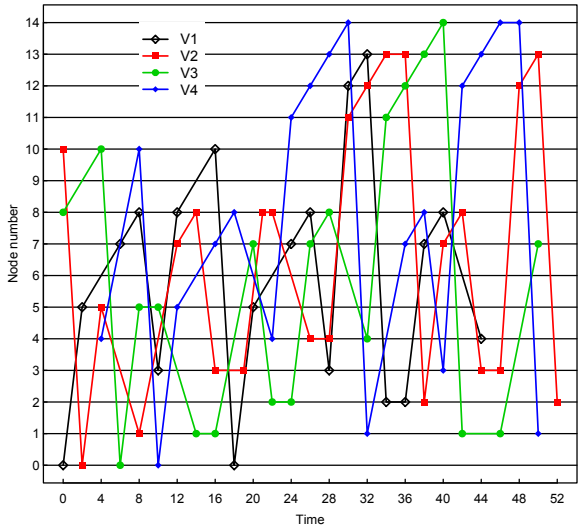
8.6 Experimental Results

No standard benchmark examples exist for the SSMV-CFR problem. As a result, we evaluated the performance of the ALS deadlock-free algorithm on the TCPN model given in Fig. 8.2 with the objective of finding an optimal or near-optimal deadlock and conflict-free integrated schedule. Here, we analyze the impact of parameters such as, the number of jobs with different operation sets and the number of AGVs, on the overall schedule. The 10 job sets in the SSMV problem are tested on layout 1. Each job set is experimented with an increasing number of vehicles (2, 3, and 4 AGVs). The 4 AGVs $V1, V2, V3, V4$ are initially located on nodes 0, 10, 8 and 4 respectively. Due to the complexity of the SSMV-CFR problem, each experiment is given up to 2h to run.

The computational results are given in Table 8.2. The solutions obtained for the 2-vehicle



(a) Gantt chart of the SSMV-CFR solution for EX51 instance.



(b) Detailed routing of the 4 vehicles on the guide path for EX51 instance.

Fig. 8.3. The integrated schedule of the EX51 instance including the conflict-free routing of 4 vehicles.

experiments compare well with the SSMV's taken into account the capacity constraints and the control of vehicles on the guide path. The schedules showed an improvement from 2 to 3 vehicles. From 3 to 4 vehicles however, not all instances gave a sign of improvement except for the EX11, EX21, EX41, EX51, and EX91 instances. The other solutions stay the same or perform worse as in EX101. This is likely due to the congestion occurring at different times on the guide path leading to delays in serving P/D requests, as the number of AGVs is increased. To illustrate this effect, we show in Fig. 8.3, the Gantt chart and the detailed path routing of the AGVs for EX51 instance using 4 AGVs. On the other hand, the experimented instances showed a significant improvement in performance between 2 and 4 vehicles excluding EX61, EX81, and EX101. As observed in the table, the process plans of jobs including the number of jobs are a determinant factor on the system performance.

The results are not conclusive since none of the solutions converged. Some of the experiments ran out of memory before reaching the CPU limit. Nevertheless, the results can give an insight as to how increasing the number of vehicles could affect the overall performance. Also, it can help in making decisions on the number of vehicles required for a particular layout and job set at the tactical level. To guarantee convergence and a possible return of improved solutions, the memory and CPU limit can be increased for a 64-bit version of the algorithm.

9
Empirical Evaluation

9.1 Case Study

We consider a case study of a real FMC of an eyeglass production system [113, 147]. The system consists of three CNC machines $M1$, $M2$, and $M3$, an automated dual-gripper crane $R(G1, G2)$, and a conveyor. The layout of the cell is shown in Fig. 9.1. There are three types of eyeglasses E_1, E_2, and E_3 in which each eyeglass is composed of two lenses (left and right). Each lens must be processed separately since the machine processing time of some operations depends on the lens type. Hence, the total number of part types to be processed is six: J_1 and J_2 for E_1, J_3 and J_4 for E_2, and J_5 and J_6 for E_3.

Parts undergo two machining operations in the same sequence as in a flow shop. The first operation is performed on $M1$, and the second on either $M2$ or $M3$. Machine $M1$ is used to verify whether the lenses have the correct dimension specification, whereas machines $M2$ and $M3$ perform the same function and are used to bevel the lenses. All the part types have the same processing time of 4 s in the first operation but the beveling operation varies depending on the part type: 120 s for E_2, 540 s for E_3, and 215 s and 220 s for J_1 and J_2 of E_1 respectively. Each machine can process at most one operation at a time.

Parts arrive in buckets of a pair of lens ($B_1(E_1)$, $B_2(E_2)$, and $B_3(E_3)$) on the conveyor to the load/unload (L/U) area. Each bucket contains an eyeglass type, and the L/U area is the working area of the crane operations on the conveyor. The crane is used to transfer the parts (in a horizontal movement) between the conveyor and the machine and back to the L/U area after the part processing is completed. However, the L/U area is constrained to three buckets. This is due to the restricted working area of the crane's gripper to load and unload parts. A slot is freed only when the two parts in a bucket are fully processed and moved out from the working area. The crane can load and unload parts at four pickup/delivery (P/D) points: 1. 0 - conveyor loading

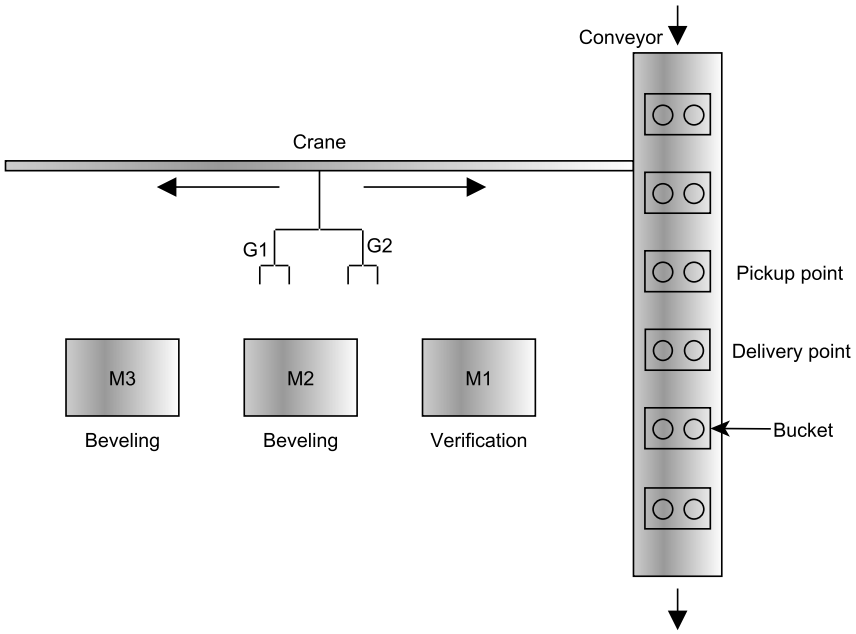


Fig. 9.1. The layout of the flexible manufacturing cell.

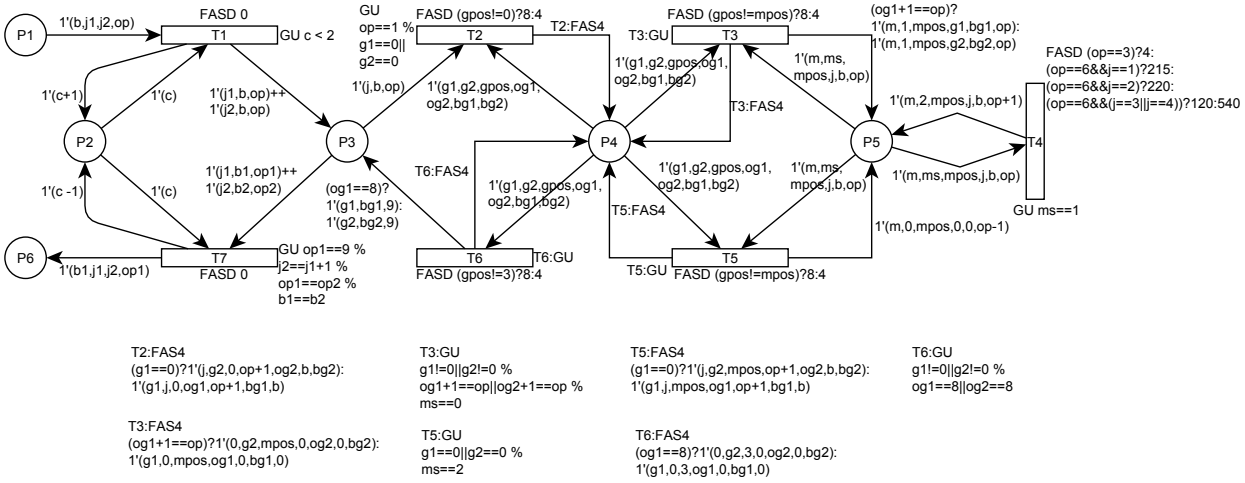


Fig. 9.2. The TCPN model of the FMC developed using TIMSPAT syntax library.

position, 2. 1 - $M1$, 3. 2 - $M2/M3$, and 4. 3 - conveyor unloading position. The movement time of the crane from one position to the other, and the loading and unloading of parts takes 4 s each. Both arms of the crane can be used to load and unload parts. The operation of the dual-gripper crane works more like a multi-load AGV that performs empty and loaded trips [87].

Each job has a total of eight operations (six handling and two machining operations): 1. Unload or pickup from bucket, 2. Transport and delivery or load to $M1$, 3. Processing in $M1$, 4. Unload from $M1$, 5. Transport and delivery to $M2$ or $M3$, 6. Processing in $M2$ or $M3$, 7. Unload from $M2$ or $M3$, and 8. Transport and delivery to bucket. The scheduling problem is formulated as follows: Given the FMS layout and the production mix, determine the starting and completion times of each part on each machine and the movement operations of the crane between machines together with the assignment according to the makespan minimization objective C_{max}.

Figure. 9.2 shows the TCPN model of the manufacturing cell. The TCPN has six places and seven transitions. The meaning of the places together with the color set and variables, and transitions is given in Table 9.1 and Table 9.2 respectively. The main file contains the initial marking of the production mix $E_1 = 1$, $E_2 = 1$,and $E_3 = 1$ with zero starting times. For dynamic scheduling purposes, the starting times can be changed to reflect the current state of the system.

$$1\ 1'(1,1,2,1) + 1'(2,3,4,1) + 1'(3,5,6,1); 2\ 1'(0); 3\ ; 4\ 1'(0,0,0,0,0,0,0); 5\ 1'(1,0,1,0,0,3)$$
$$+1'(2,0,2,0,0,6) + 1'(3,0,2,0,0,6); 6\ ;$$
$$0+0+0; 0;\ ; 0; 0+0+0;\ ;$$
$$\text{CS}\ WKL; CNT; BCK; CRN; MCH; OUT;$$
$$\text{EF}\ \#; 1'(0); \#; 1'(0,0,*,0,0,0,0); 3'(*,0,*,0,0,*); 3'(*,*,*,9);$$

A P/D request involves 5 operational sequence: the prior assignment of the crane to the part, the movement of the crane to the resource location (conveyor or machine) for pickup if its current position is different from the P/D location, the unloading of the part from the resource, the delivery of the part to the destination resource, and the loading of the part to the resource. Transitions $T2$ and $T5$ execute the first three operations concurrently for conveyor and machine pickup respectively, while $T3$ and $T6$ perform the last two operations for machine and conveyor delivery respectively. The production flow of a single part in the system gives the sequence of transition firings: $T1 \rightarrow T2 \rightarrow T3 \rightarrow T4 \rightarrow T5 \rightarrow T3 \rightarrow T4 \rightarrow T5 \rightarrow T6 \rightarrow T7$.

Table 9.1. Interpretation of places and colors.

Place	Color set (CS)	Color(s)	Description
$P1$	$WKL = 4, timed;$	$\langle b, j1, j2, op \rangle$	Production mix workload: Left lens $j1$ and right lens $j2$ in bucket b with starting operation sequence identifier op
$P2$	$CNT = 1, timed;$	$\langle c \rangle$	Counter for the number of unprocessed buckets on the conveyor in the loading area
$P3$	$BCK = 3, timed;$	$\langle j, b, op \rangle$	Buckets on the conveyor: Unprocessed lens j in bucket b if op=1, otherwise processed if op=9
$P4$	$CRN = 7, timed;$	$\langle g1, g2, gpos, og1, og2, bg1, bg2 \rangle$	Crane status: Crane at position $gpos$ with grippers $g1$ and $g2$ (free = 0, busy > 0), and operation sequence identifiers $og1, og2$ and bucket identifiers $bg1, bg2$ for each gripper. $og1, og2, bg1, bg2 > 0$ if parts (lens) are held in grippers
$P5$	$MCH = 6, timed;$	$\langle m, ms, mpos, j, b, op \rangle$	Machine status: Machine m at position $mpos$ with status ms (0 - available, 1 - busy, 2 - waiting for part to be picked up). $j, b, op > 0$ if $ms > 0$
$P6$	$OUT = 3, timed;$	$\langle b, j1, j2, op \rangle$	Processed buckets moved out from the working area

The TCPN model is quite different from the nets proposed by Narciso et al. [113], Mujica and Piera [147]. In the previous nets, the movement of the crane is not properly controlled as the source or destination of a P/D request is not included in the crane's behavior. The crane can move to either of the potential P/D positions without a prior assignment for P/D request. Since the studied FMC is a bufferless system, the TCPN minimizes blocking using P/D assignment together with the status of the machines as specified in the guards of transitions $T2$ and $T5$. However, deadlock occurrence is still inevitable if the crane grippers are holding parts whose destination machines have parts that are waiting to be picked up by the crane. The deadlock-free scheduling approach used in this thesis has been treated in Chapter 5.

Table 9.2. Transitions and their meanings

Transition	Description
$T1$	Conveyor move in of unprocessed buckets in the L/U area
$T2$	Unloading of parts from bucket for P/D including the empty move of the crane if applicable
$T3$	Movement of crane to the destination machine for delivery including the loading of parts
$T4$	Processing of parts in machines
$T5$	Movement of crane to the destination machine for pickup including the unloading of parts
$T6$	Final delivery of processed parts to bucket in the conveyor's L/U area including unloading of parts
$T7$	Conveyor moves out processed buckets

Table 9.3. Production mix instances for the three eyeglass types.

Instance	Number of eyeglass types			Number of parts
	E_1	E_2	E_3	
BGS1	1	1	1	6
BGS2	2	2	0	8
BGS3	4	0	0	8
BGS4	4	2	1	14
BGS5	3	3	3	18
BGM1	2	5	3	20
BGM2	7	4	2	26
BGM3	4	7	4	30
BGM4	4	8	5	34
BGM5	7	5	5	34
BGM6	6	10	4	40
BGL1	7	13	2	44
BGL2	10	17	5	64
BGL3	15	20	7	84
BGL4	20	20	20	120

9.2 Performance Evaluation and Benchmarking

This section evaluates the performance of the nine algorithms on the case study. We consider 15 different production mix scenarios, shown in Table 9.3. The instances contain 5 small ($BGS1$–$BGS5$), 6 medium ($BGM1$–$BGM6$), and 4 large ($BGL1$–$BGL4$) workload mixes. The first large instance $BGL1$ ($30\%\ E_1, 60\%\ E_2, 10\%\ E_3$) represents a real mix for the manufacturing cell. The experiments were conducted on a 2.60GHz AMD Opteron processor PC with 4GB RAM.

Each algorithm is classified according to the space-time trade-off criterion given in Chapter 3. BFIDA* and BFIDA*-SLDD fall under the SE class. They perform a series of breadth-first searches with a given cost threshold which is used as a bound to control the memory usage of A* in order to avoid keeping the least promising paths in the state space. In addition to the cost threshold pruning, BFIDA*-SLDD exploits the repetitive patterns found in the state space graphs of FMS of similar configurations to further reduce the memory requirements.

The TE class includes the HHS algorithms like BAS, A*-BT, and DWS. These algorithms use predefined pruning rules to limit the memory consumption of the search space and find feasible solutions in a reasonable amount of time. Basically, they limit the frequent backtrackings of the A* search to prevent the search space from degenerating into a breadth search. A controlled deepening search is favored to drive the search towards a suboptimal solution quickly. The STE class consists of anytime algorithms; ALS, ACAS, and DFBnB.

Each algorithm is benchmarked against the others in its category, and the best performing in each class is then compared with those of the other classes. Since each class has its own trade off, CPU time of 3600 s and 4GB RAM limits were set for the TE and STE classes whereas only the maximum memory limit was set for the SE class. A* is not considered for comparison as it was only able to solve the small instances, even with the CESS.

Table 9.4. Scheduling results of SE algorithms.

Instance	State space			BFIDA*		BFIDA*-SLDD				DFS-CSS [113]	
	Itr	Depth	EM	Peak	CPU (s)	Peak	CPU (s)	Reduction (%)	C_{max}	C_{best}	CPU (s)
BGS1	4	54	150,548	114,371	59.8	4,869	59.1	95.74	975	1027	45077
BGS2	5	72	171,140	117,869	71.9	4,932	71.4	95.82	783	911	54207
BGS3	3	72	19,905	18,297	7.9	496	7.9	97.29	978	1111	2978
BGS4	5	126	1,608,257	1,014,911	718.2	21,237	715.8	97.91	1794	___	___
BGS5	5	162	4,735,124	3,000,133	2216.9	57,783	2211.7	98.07	2802	___	___
BGM1	5	180	5,211,153	3,363,369	2436.2	55,308	2431.2	98.36	2835	___	___
BGM2	5	234	11,718,087	6,104,427	5663.3	73,095	5583.1	98.80	3301	___	___
BGM3	5	270	29,217,707	___	___	131,865	14463.4		4110	___	___
BGM4	5	306	32,719,717	___	___	163,967	16060.0		4794	___	___
BGM5	5	306	32,228,078	___	___	186,191	16056.4		5089	___	___
BGM6	6	360	52,626,061	___	___	201,008	26330.5		4965	___	___
BGL1	5	396	30,251,710	___	___	125,448	15030.2		4489	6790	513540
BGL2	6	576	179,031,180	___	___	406,956	93592.5		7359	___	___
BGL3	5	756	274,552,404	___	___	820,818	159148.0		10009	___	___
BGL4	4	1080	1,061,683,289	___	___	2,295,016	649915.0		18330	___	___

Itr - Number of iterations

9.2.1 SE Class

The SE algorithms use the cost threshold CT as a bound to prune markings with $f(M) > CT$ in each iteration of the breadth-first branch and bound search. The algorithms start with $f(M_0)$ as the initial CT. If no goal marking is found, the search is repeated with a new CT value until a solution is reached. Successive values of CT relies on the minimum $f(M)$ (f_{min}) of the unexpanded markings in the previous iteration. However, it comes with an overhead of marking re-expansion each time the search is restarted. If the increments are too small to reach $f(M_g)$ in a reasonable amount of time, it may result in a large number of iterations. To circumvent this problem, [184] propose to double CT after each iteration for the classical IDA*. This measure can degenerate to a breadth-first search. To achieve a good space-number of iteration trade-off, we computed successive CTs as $CT = max(f_{min}, ubf * f_{min}), \forall ubf \in \{1.0, 2.0\}$ where ubf is a multiplier. We experimented with different values of ubf to determine a good factor for the instances. From the preliminary results obtained, a ubf value of 1.4 achieves a good trade-off.

Table 9.4 shows the scheduling results of the two SE algorithms on the 15 instances. The two are practically the same. They expand the same of number of markings (*EM*) and have an almost equal CPU time but differ in the number of stored markings as shown in the *Peak* column. BFIDA*-SLDD leverages the regular structures found in FMS to reduce the number of stored markings. The *Depth* is the total number of operations or fired transitions required to reach the optimal M_g from M_0, where each operation corresponds to a level in the state space.

As observed on the result table, the ubf of 1.4 significantly reduces the number of search iterations across all instances. As a result, the computational times were also reduced for the small and medium instances. For example, the BFIDA* search of $BGS1$ using the standard increment of $ubf = 1$ solved the instance in 156 iterations with a CPU time of 3753 s and EM of 9.6×10^6. As the instance size becomes larger, the standard BFIDA* ran out of memory (o.o.m) starting from the $BGM3$ instance due to the exponential increase of the state space size. BFIDA*-SLDD solved all the instances with a minimal amount of memory space. It used less than 2GB RAM for the largest instance that would require over 200GB RAM if the entire state space were to be stored in memory. The CPU times can be considered reasonable up to instance $BGL1$ considering the fact that it is

an optimal algorithm. However, it took over 38 h, 64 h, and h to solve $BGL2$, $BGL3$, and $BGL4$ respectively. When compared with the depth-first search (DFS-CSS) algorithm proposed by [113] where some instances coincide, it is a significant improvement in terms of both CPU time and best makespan (C_{best}) obtained.

9.2.2 TE Class

Each algorithm in the TE class has its own input parameter settings, used as a measure to reduce the memory and computational time complexity. However, they share a common parameter that determines a priori the number of markings to be stored. Since increasing the input values do not guarantee a high solution quality, different values must be experimented to achieve a good solution quality-time trade-off.

The A*-BT algorithm requires only one input parameter called threshold, M_{max} which is used to control the maximum number of markings stored in OPEN. It starts exploring the state space in a best-first order using A*. Each time M_{max} is reached, it creates a new search region or level by initiating a backtracking to the previous level where the best marking in OPEN is used as a root node for another A* search until a solution is found.

BAS requires two inputs: 1. beam width bw, and 2.cutoff co. The beam width is used to limit the number of markings expanded at each depth of the state space whereas cutoff limits the size of OPEN to a certain value to avoid an exponential growth. On the other hand, DWS restricts the state space to a dynamic search window between a minimum depth, bottom-depth bd and a maximum depth, top-depth td. For the search window to advance, DWS constrains the number of markings generated at td to a certain value called max-top $maxt$. Once $maxt$ is exceeded, the values of bd and td are increased by one. To avoid exponential growth, DWS also keeps the most promising markings to be explored at each depth of the search window to a fixed size called max-size $maxz$. If a level becomes full (i.e. $maxz$ has been reached), a new marking M is added to the level only if there is a stored marking M_s with $f(M_s) > f(M)$. As such, three input values $(td, maxt, maxz)$ are required for DWS to run. The bottom depth starts from zero and [79] propose to set $maxt = maxz$.

To provide a fair comparison, we experimented the following values for the three algorithms: $M_{max} \in \{10, 50, 100, 200, 500, 1000\}$, $(bw, co) \in \{(10, 150), (50, 750), (100, 1500), (200, 3000), (500, 7500), (1000, 15000)\}$ and $(td, maxz) \in \{(30, 10), (30, 50), (30, 100), (30, 200), (30, 500), (30, 1000)\}$. Since we are not restricting the maximum number of successors to be generated at each marking, the input values must be large enough to reach a goal marking. For the DWS, the proposed initial $td = 15$ did not generate a feasible solution for most of the instances.

Table 9.5 presents the results obtained by the three algorithms for the 15 instances. For each algorithm, we show the input parameter value that returned the best solution taking into account the CPU time and compare the relative percentage deviation (RPD) from the best solution (RPD_C) and CPU time (RPD_{CPU}) between the three algorithms. The A*-BT algorithm provided the best solution-time quality trade-off considering the little computation time used to obtain the first solutions. Unlike the other algorithms, the CPU time is somewhat maintained across all instances without exceeding 12 s. In terms of solution quality, BAS is superior using the average deviation, albeit with a higher runtime overhead. Also, BAS proved to be more time-efficient than the others in 8 of the instances.

Table 9.5. Scheduling results of TE algorithms.

Instance	A*-BT					BAS					DWS				
	M_{max}	C_{best}	CPU	RPD$_C$	RPD$_{CPU}$	bw	C_{best}	CPU	RPD$_C$	RPD$_{CPU}$	$maxz$	C_{best}	CPU	RPD$_C$	RPD$_{CPU}$
BGS1	1000	1012	2.7	3.8	41.2	100	980	1.9	0.5	0.0	200	975*	3.9	0.0	106.0
BGS2	100	831	0.2	5.1	0.0	200	791*	5.2	0.0	2026.0	100	799	2.8	1.0	1072.3
BGS3	1000	1015	5.8	3.8	0.0	500	983	7.4	0.5	27.3	500	978*	12.2	0.0	110.0
BGS4	1000	1889	9.0	4.4	0.0	500	1822	25.2	0.7	180.3	1000	1809*	43.9	0.0	388.1
BGS5	100	2989	0.7	4.0	0.0	1000	2873*	68.8	0.0	9826.0	1000	2877	60.7	0.1	8665.2
BGM1	100	3079	0.8	3.8	0.0	500	2966*	39.2	0.0	5012.4	50	2974	4.7	0.3	509.5
BGM2	200	3580	1.9	2.1	0.0	200	3538	21.5	0.9	1032.3	50	3506*	8.1	0.0	326.9
BGM3	1000	5142	11.7	22.1	81.6	50	4212*	6.5	0.0	0.0	50	4270	49.4	1.4	664.0
BGM4	10	5172	0.2	3.1	0.0	10	5018*	1.6	0.0	561.9	50	5086	8.4	1.4	3321.0
BGM5	100	5527	1.6	3.5	0.0	10	5342*	1.6	0.0	1.6	100	5367	15.0	0.5	866.9
BGM6	10	5498	0.3	5.2	0.0	1000	5228*	170.2	0.0	50897.9	200	5396	33.6	3.2	9972.3
BGL1	10	4803*	0.3	2.3	0.0	200	4895	39.2	4.2	11970.7	500	4696	84.8	0.0	25967.5
BGL2	10	7954	0.5	0.5	0.0	10	7917*	3.2	0.0	486.5	10	7921	6.3	0.1	1065.6
BGL3	10	10507*	0.7	0.0	0.0	10	10803	4.3	2.8	499.3	50	11123	27.7	5.9	3760.3
BGL4	10	19215*	1.1	0.0	0.0	10	19306	6.5	0.5	504.7	50	19984	36.2	4.0	3285.4
Av. RPD				4.24	8.19				0.68	5535.12				1.18	4005.39

*-Best solution

9.2.3 STE Class

Like the TE class, the STE algorithms also define some input parameter settings with the exception of DFBnB. In ACAS, three parameters are needed before exploration: the initial column width ω, the column width increment α, and the maximum column width ω_{max}. ACAS is an adaptive search that focuses on improving the current best solution obtained in a minimal time whenever possible. It increases ω by α if the solution is not improved after a certain number of iterations. The width increment is stopped when ω reaches ω_{max} to avoid unnecessary memory usage if the solution cannot be improved within the time frame. Once the solution is improved, the column width ω is reset to its initial value. On the other hand, ALS does not require an input parameter a priori. However, to return solutions in a good time frame for problems with large branching factor (number of successors), it is advised to limit the number of markings expanded at each level called the expansion width $e\omega$. We set $\omega = e\omega = 5$, $\alpha = 5$, and $\omega_{max} = 50$ [81].

The three algorithms differ in two aspects. The first is the number of markings expanded at each level in an iteration. ACAS and ALS is determined by ω and $e\omega$ respectively, whereas DFBnB defaults to 1 for all iterations in the search. The second is the way in which backtracking is performed. Backtracking is chronological in ACAS, best-first in ALS, and depth-first in DFBnB.

Table 9.6 shows the scheduling results of the STE algorithms for the 15 instances. While the average deviation across all instances grants the ALS as the best out of the three algorithms, the varying degrees of solution quality per instance class must be taken into account. From this perspective and benchmarking the performance as the problem size increases, the DFBnB worked better for larger instances. It outperformed the others in the medium and large instances obtaining the best solutions at a much reduced computation time in 7 out of 10 instances. Evidently, this makes it more practical than others. ACAS and ALSS only performed better in the small instances. Another reason DFBnB is the best fit for this problem is that the CPU time required to return the first solution is more or less stable (< 1 s) for all the instances, unlike the other two that experienced a sharp increase in time for the last three large instances. Also, all the first solutions obtained by DFBnB are clearly better.

Table 9.6. Scheduling results of STE algorithms.

Instance	ACAS-TCPN					ALS					DFBnB				
	First solution		Best solution (CPU=3600)			First solution		Best solution (CPU=3600)			First solution		Best solution (CPU=3600)		
	C_f	CPU_f	C_{best}	CPU_{best}	RPD_C	C_f	CPU_f	C_{best}	CPU_{best}	RPD_C	C_f	CPU_f	C_{best}	CPU_{best}	RPD_C
BGS1	999	0.09	**975**	88.4	0.0	999	0.09	**975**	94.8	0.0	992	0.03	**975**	3476.1	0.0
BGS2	863	0.13	**783**	54.2	0.0	863	0.13	**783**	93.7	0.0	843	0.03	787	309.8	0.5
BGS3	1051	0.14	**978**	13.8	0.0	1051	0.17	**978**	19.4	0.0	1038	0.03	**978**	1881.3	0.0
BGS4	1846	0.23	**1794**	704.7	0.0	1846	0.25	**1794**	752.2	0.0	1820	0.06	1810	17.6	0.9
BGS5	2938	0.31	2802	510.0	0.0	2938	0.31	2802	1220.4	0.0	2858	0.08	2858	0.1	2.0
BGM1	2947	0.34	2835	2513.8	0.0	2947	0.34	2835	778.3	0.0	2895	0.08	2895	0.1	2.1
BGM2	3485	1.67	3324	3036.2	0.3	3392	0.59	3316	3045.4	0.1	3332	0.11	3313	223.8	0.0
BGM3	4748	0.73	4127	2332.4	0.0	4217	0.53	4135	1891.4	0.2	4170	0.13	4166	0.2	0.9
BGM4	4901	0.61	4804	2291.0	0.0	4901	0.61	4809	1866.9	0.1	4854	0.14	4850	0.2	1.0
BGM5	5161	2.34	5114	1123.1	0.2	5199	1.09	5119	1302.3	0.3	5120	0.14	5106	235.7	0.0
BGM6	5226	4.77	5000	2345.7	0.2	5114	1.20	4997	67.6	0.2	4996	0.17	4988	4.5	0.0
BGL1	4713	5.32	4544	109.8	0.7	4597	1.30	4519	2240.0	0.2	4520	0.19	4512	4.9	0.0
BGL2	7570	8.60	7430	141.4	0.5	7497	10.76	7424	1851.6	0.5	7400	0.28	7390	55.6	0.0
BGL3	10420	24.01	10099	3572.5	0.2	10334	74.99	10148	399.5	0.7	10094	0.37	10082	57.0	0.0
BGL4	19043	30.67	18645	1989.7	1.2	18919	143.51	18506	1671.3	0.5	18430	0.58	18418	66.3	0.0
Av. RPD					0.23					0.17					0.49

Bold solution - Converged to optimal

STE algorithms are designed to produce feasible solutions quickly, and regularly improve the incumbent best solution C_{best} over time. They are able to guarantee optimality provided that the memory available and time allocated are large enough to reach the optimal solution. The incumbent best solution may have been obtained but cannot be considered optimal until all the markings with $f(M) \leq C_{best}$ have been expanded. As such, the time gap between when the incumbent best solution was returned and the convergence time varies depending on the number of markings remaining to be explored. For instance, DFBnB obtained the optimal solution for BGS3 at 1881.3 s but converged at 3213.9 s, while the gap is lower for ACAS and ALS that converged at 1370.3 s and 1364.7 s respectively for BGS4. On the other hand, both ACAS and ALS obtained the optimal solutions for BGS5 and BGM1 but did not converge within the CPU time limit.

9.3 Discussion

Each algorithm class has its strengths and weaknesses. The SE algorithms trade space for time. They are the best option when sufficient time is given for producing an optimal schedule. But it seems quite impractical for highly demanding and dynamic environments in which solutions must be returned in a short computation time. However, the STE algorithms offer an extra advantage in terms of both solution quality and time efficiency such that they can adapt to different memory and time constraints. One of the weaknesses of the STE class is that an additional running time does not necessarily lead to a better solution [185].

The SE class cannot be directly compared with TE (or vice-versa) because each stands at two ends of a continuum. On the other hand, the STE class can be benchmarked against SE in terms of the percentage of optimality lost (RPD from optimal solution) and the computation time reduction, and also against TE on the solution quality and computation time comparison of the first solution returned.

The optimality lost is quite low for the best performing algorithm in the STE class, DFBnB. It ranges between 0.3% and 2.1% for the non-converged solutions. The CPU time savings is about 99%

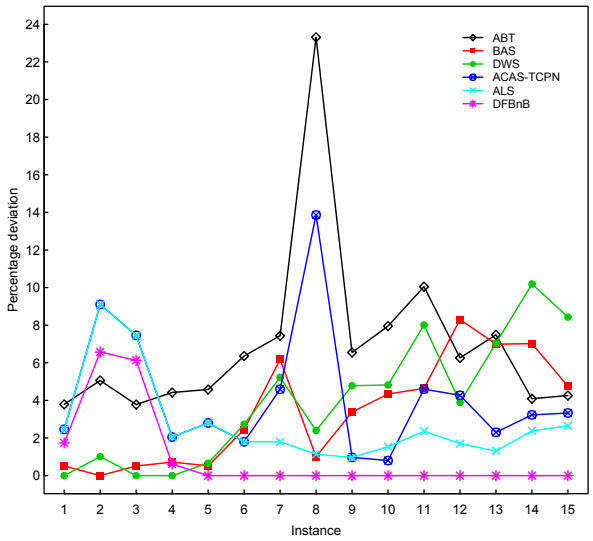


Fig. 9.3. Relative percentage deviation of the first solution returned by the TE and STE classes.

for most of the instances excluding the first three. Figure 9.3 shows the RPD of the first solution returned by the TE and STE classes. The algorithm that produces the best first solution for each instance takes a zero RPD. Clearly, the STE class outperforms the TE's with the exception of the small instances. The DFBnB performed better in nearly all the instances with the A*-BT as the least performing.

For the TE and STE algorithms, it is quite difficult to predict when the first solution will be returned, as it largely depends on the problem size. Although DFBnB achieved a stable CPU time for all the instances. Notwithstanding, these results are not conclusive and cannot be used as a benchmark for all systems. The performance of each algorithm may be different for another problem set. Each system has its own behavior, and an empirical evaluation may be required to determine the best-performing algorithm. With these results, it is pretty straightforward to draw a conclusion on which algorithm can be adapted to an off-line scheduling or on-line when the system deviates from its original schedule or in the event of a failure or disturbance.

While the overall computation time lies on how each search algorithm explores the reachability graph, it is worth benchmarking the time consumed on each computational task in the search exploration. To identify the main source of bottleneck in TIMSPAT, the distribution of the run time of four algorithms is given in Fig. 9.4. Clearly, the simulator dominates a larger proportion of the run time irrespective of the search algorithm employed. More time is spent on tasks like the enabling and firing of transitions for marking generation, and the computation of heuristic functions. This means that the overall efficiency of the tool relies on the simulator, which confirms that it is computationally expensive to simulate CPN models due to the difficulty in manipulating colors. The search part (OPEN and CLOSED) only consumes about 3% of the total time.

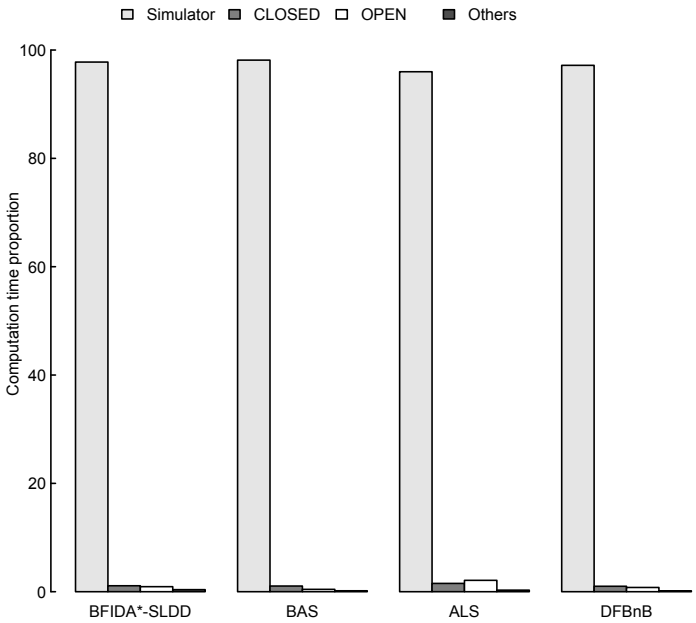


Fig. 9.4. Run time proportion of each component in TIMSPAT for the BGL1 instance.

10
Conclusion and Future Work

10.1 Summary of Contributions

This thesis has presented a quantitative and computational approach for the modeling and performance evaluation of scheduling problems in FMSs. A TCPN-based scheduling methodology is proposed whose underlying analysis relies on the reachability graph. However, its application has been limited to small-sized problems due to the computational complexity of production scheduling problems in FMSs. In this research work, a number of AI-based heuristic search algorithms have been proposed to alleviate the memory and time requirements of the reachability graph in order to increase its computational power and capability to handle challenging scheduling problems.

In spite of its ability to perform the automatic analysis of the modeled system, there is a lack of tool that supports the performance evaluation of TCPN through timed state space analysis via heuristic search methods. This motivated the development of a new tool (TIMSPAT) presented in Chapter 2 to deal with the shortcomings of existing tools on CPNs. Most tools using state space analysis focus on the model checking of untimed CPNs. Other shortcomings include simulation limitations, timed state space generation with global clock synchronization, absence of efficient search algorithms, and reliance on third-party software applications. TIMSPAT provides a syntax library based on C++ and CPN Tools token multiset, which allows the creation of CPN models in a textual format. The structure of the CPN files offers a localized enabling and firing of transitions. Also, heuristic functions that can be adapted to different production scheduling problems were discussed.

TIMSPAT provides a platform for describing CPN models as well as simulating the behavior of the system, and optimizing the performance of scheduling problems with different algorithms. One of the benefits presented by this tool is its ability to implement different heuristic search methods using the same syntax library and data structures. As a result, different FMS scheduling scenarios can be benchmarked to allow for correct conclusions to be drawn. The tool is expected to make production managers more flexible in their decision making process without being over reliant on a particular scheduling algorithm.

Chapter 3 presented an extensive literature review on the state-of-the art heuristic search algorithms of the PN-based scheduling methodology. Two areas are identified that involve works devising efficient heuristic functions for existing algorithms and the other, combining one or more search methods to make the search exploration more efficient. In the latter, the search algorithms are classified into SE, TE, and STE according to the space-time trade-off criterion. The review reveals the domination of TE algorithms because of the current research trend toward obtaining suboptimal solutions in short computational times. However, they have failed to explore those areas that can still be of great benefit in improving the performance of the system. This thesis took a step forward by expanding the body of knowledge on the SE and STE classes.

In Chapter 4, an STE algorithm called ACAS-TCPN is proposed for time-critical production scheduling. It overcomes the drawbacks of classical heuristic search algorithms like A* and SE algorithms that often take a long time to find optimal schedules, as well as TE algorithms that terminate the search when the first solution is found. The proposed algorithm not only returns quick solutions like the TE's, but also improve the solutions over time as well as reaching the optimal solution. Optimality is guaranteed if the available memory and specified time are sufficient to make the incumbent solution converge. The algorithm uses the condensed state space described in Chapter 2 as the underlying search graph to avoid a continuous reevaluation of untimed markings in the timed state space. Several benchmark problems on FMS scheduling were solved, and the results obtained showed the effectiveness of the algorithm over the existing ones.

Chapter 5 deals with deadlock-free scheduling problem in FMSs and presents a novel STE

algorithm (ALS). The paper highlighted the two approaches for deadlock-free scheduling using the PN formalism. The first is to create a deadlock-free model with deadlock resolution or control policies (DCP) and then, the optimal schedule is searched with the deadlock-free model. The second is to search the deadlock-prone model. The paper argue in favor of the latter; that the generation of a feasible and deadlock-free schedule in PN-based scheduling is not dependent on the incorporation of DCP, but rather on the approach used in solving the deadlock-prone FMS. DCP are an option and not a requirement as deadlocks are explicitly defined in the PN framework. However, there is a possibility that most of the markings in the reachability graph are deadlocked. Taken this into account, the first approach is promising because potential deadlock schedules can be prevented by the control policies. Also, DCP can be used to reduce the search space.

While the first approach seems promising, it has several drawbacks regarding the scheduling performance. First, the computation procedure of optimal control policies is NP-hard. As such, existing policies cannot capture all the possible rules or scenarios that do not restrict the firing of a feasible transition that can lead to a better scheduling performance. Next, the resolution policies are conservative and limit the number of alternatives that can be explored, which may prevent the system from reaching an optimal schedule. Lastly, not all policies work for all systems. Their scheduling performances are quite different from each other even when applied to the same system.

On the other hand, the second approach is faced with the challenge of dealing with deadlock situations when most of the nodes are deadlock. However, the handling of this problem is quite relative. Since the reachability graph is explored in fragments and guided by a heuristic cost function, not all deadlocks are encountered (or selected) by the search algorithm. Some deadlocks are pruned before being selected if they have an f-cost greater than the upper bound, while steps are taken to avoid a repeated occurrence of an encountered deadlock situation, using the information derived from the transition bindings. The paper demonstrated the effectiveness of the approach on a comprehensive set of deadlock-prone FMS example. The experimental results indicate that near-optimal solutions can be obtained in relatively short computation times under different FMS configurations.

The focus of the research work took a turn in Chapter 6. The paper investigates a possible release of memory during BFS exploration of the reachability graph, leading to SE algorithms. In BFS, the state space is partitioned into layers by default, where a layer comprises all the states with same minimal distance from the initial marking. The idea is to remove layers where no successors can be found from exploring markings in the most recent and future layers. To identify such layers, the approach relies on a profile called LDS, which examines the behavior of the graph on a per-model, per-layer basis. The LDS records, for each layer $L(i)$, the relative distance of other layers where successors of markings in $L(i)$ can be found. Using the LDS, the paper developed a new approach called SLDD. The SLDD is based on the notion that the state space graphs of a system with increasing problem size may contain repetitive patterns (structural equivalences) while the underlying system configuration is fixed. Since the state explosion is caused by a scalability problem, the knowledge of the system behavior obtained from explorable smaller problem sizes through LDS can be used to solve larger sizes. This approach has been used to solve multiple lot size scheduling problems in FMS and extended to solve problems of similar configurations where the problem size differ by the number of jobs, resources and operations.

While Chapters 4, 5, and 6 dealt with FMS scheduling problems with robots as the MHS, Chapter 7 expands the research to simultaneous scheduling of machines and AGVs. Unlike the fixed robots, AGVs are usually employed when parts have to be physically transported from one machine location to the other. The AGV scheduling process includes management functions like assignment or dispatching, and the routing of vehicles on the guide-path layout. The solution

method proposed offers a new approach and introduces some novel elements into the studies of simultaneous scheduling in comparison to the other methods used in the literature (mainly metaheuristics). The paper considers two schemes of the interaction between the machining and transport system: machine-controlled, and vehicle-controlled task assignment to AGVs. For each of these schemes, a TCPN model is proposed. The approach moves away from the conventional OR model by employing an event-driven approach to describe the logical behavior of FMS operations. Also, one of the main contributions of this study is that the TCPN-based methodology permits an integrated approach addressing the simultaneous scheduling as a single problem rather than usual hierarchical approach in which the problem is decomposed in subproblems, and each of them is solved separately. The results obtained using the ALS algorithm show that TCPN application is competitive with other state-of-the-art methods.

Chapter 8 extends the simultaneous scheduling work in the previous chapter by incorporating the detailed AGV routing problem on the guide path. Here, the movement and control of vehicles are properly managed for conflict-free routing using the zone-control technique. A valid schedule must then determine the arrival and departure times of vehicles at each path segment. In Chapter 9, an empirical evaluation of the implemented algorithms in TIMSPAT is performed on a real FMS case study with several experiments. The strengths and weaknesses of each algorithm were identified, including the bottleneck of TIMSPAT, the simulator.

As shown in Chapter 7, the proposed methodology could as well be applied to on-line scheduling where the occurrence of real time or unplanned events like machine breakdown, tool failures, early or late arrival of jobs etc during the execution of a schedule can affect the performance of the FMS. In this kind of situations, rescheduling needs to be done and solutions must be provided in the slightest possible time. The scheduling methodology meets this requirement and fits perfectly into the objective of on-line scheduling using the STE algorithms. The simulator can be easily integrated with the shop floor database to collect information on the current state of the system and the TCPN model can then be reinitialized with the new state to generate a new schedule.

10.2 Future Work

This research work has covered different aspects of TCPN-based scheduling that involves the development of tools and algorithms, and applications to a wide range of FMS scheduling problems. In spite of this, there is still much work to be done. The following have been identified as potential areas for future research, for improving the tool as well as developing more efficient search algorithms:

1. **CPN-XML Translator from CPN Tools to TIMSPAT Syntax Description**

   Although, analysis techniques can be performed without the users' intervention at the back-end in most cases, the absence of a graphical user interface for model development can easily put off non-research oriented users. To make the tool more attractive, a plug-in translator can be provided to convert models developed in GUI-based tools like CPN Tools to the TIMSPATLib CPN description. However, these models must comply with the standard syntax rules of TIMSPATLib.

2. **Extending the TCPN formalism**

   Several scheduling problems can be modeled by the TCPN formalism. This has been shown in several examples illustrated in this thesis. However, there are certain scheduling characteristics that limit the modeling power of TCPN. In scheduling problems that involve due-dates,

hard deadlines, time windows etc, the formalism has not provided the primitive to add extra time functions and/or handle time stamps as variables. The current TCPN standard only allows the user to efficiently model the logical behavior with a minimal control on time value manipulation.

To allow this functionality, the time stamps or values must be introduced as colors in the model, making it untimed. This defeat the modeling purpose of TCPN, and since the simulation of a TCPN is different from an untimed CPN, problems of this nature can hardly be modeled in TCPN. Even though these features can be added at the programming level, they cannot be directly expressed by the user. The TCPN formalism must be extended to support these features. To achieve this, an extra effort is needed as an extension would require a different simulator. This is an area that needs to be looked at in expanding the outreach of the TCPN formalism.

3. **Analytical proof of the SLDD concept**

   We have proved experimentally using an algorithmic approach that the LDS of N_0 can be scaled to a larger size using the structural equivalences determined from smaller instances of the problem. However, there are some open research questions that needs to be answered to make it applicable to other systems: 1. How good is the estimated N_0? It is quite difficult to determine whether the LDS of the estimated N_0 obtained is complete or incomplete. The LDS is said to be complete if it is globally stable i.e. it is an exact LDS that can be scaled to any N, whereas it is incomplete if the stability is achieved only for some N. 2. Can N_0 be determined analytically? including a proof on its global stability?

4. **Combining time sweep-line with SLDD**

   In the time sweep-line method, time (be it the global clock or firing time) is used as a monotonic progress measure that is most likely to increase for every event occurrence. In existing works, time is usually carried over into the state space (a property inherent to the markings) during duplicate detection. Since the time value used is part of the full state descriptor, duplicates of markings can always be found at the same sweep-line layer. However, the problem of regress edges arises when the duplicate detection procedure is reduced to only untimed markings in the condensed state space (Chapter 3). Here, the time information is separated from the untimed marking. As a result, the time progress measure only takes into account the timed state space. To solve this problem, two options exist. The first is to provide a compound progress measure with two values; one for the timed state space, and the other for the untimed. Using this option, there would not be any need to check for regress edges since the compound measure will ensure that duplicates are kept at the same sweep layer. The drawback of this solution is that the number of markings to be stored at each layer would become larger. The second option is to keep the time sweep-line state space as it is with the time progress measure and apply the SLDD to detect regress edges.

5. **Parallel and distributed TIMSPAT**

   It has been shown in Chapter 9 that about 98% of the computational time is spent on simulating the TCPN models, which is quite inefficient for the implemented search algorithms. The availability of multi-core computers, distributed computing, graphics processing units (GPUs) etc, can help in speeding up computation thereby improving the overall efficiency of the tool. The simulator could be more efficient as the workload of transition enabling and firing will be shared by several processors or computers. This directly impacts the search

algorithms. The first solution of large-sized problems can be returned very quickly and the convergence time of anytime algorithms will be reduced. One of the interesting points of this approach is the increasing need to solve large scale optimization problems, which can be easily applied to plant-wide scheduling problems.

Other areas that can be considered as part of the future work includes:

6. Instead of the reactive approach of scheduling/rescheduling, a simulation-optimization approach can be used to deal with uncertainties using stochastic CPNs to model variations in processing times and other disturbances such as machine breakdown.

7. The use of interval TCPN [109] to model and optimize dynamic scheduling problems with variable processing times.

8. The TCPN-based scheduling can be extended to other combinatorial optimization problems like project, crew, and aircraft scheduling problems.

9. The modeling approach allows the definition of new objective functions that can deal with scheduling policies oriented to lean manufacturing in which non-added-value operations would be minimized or a rapid manufacturing in which the total completion time would be minimized. This presents an opportunity to explore multiobjective optimization using the developed methodology.

10. The incorporation of metaheuristics like genetic algorithms into TIMSPAT.

11. A web portal is currently being developed for the tool to allow users simulate their models on-line as well as providing a service-oriented platform. For a possible technology transfer, appropriate interfaces will be needed for communication with manufacturing execution systems.

A
Paper IV Appendix

A.1 AMS Layout and CPN Model

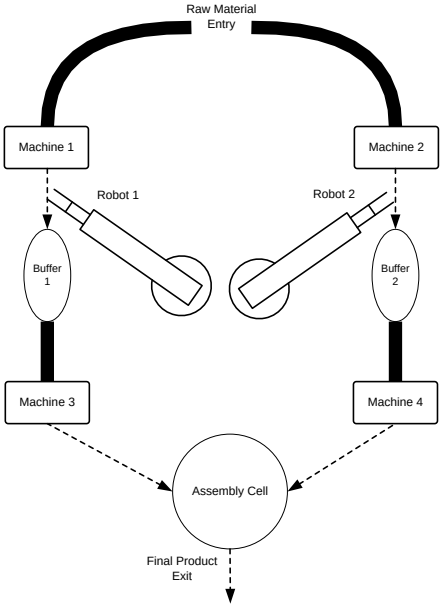


Fig. A.1. The layout of the AMS example [2].

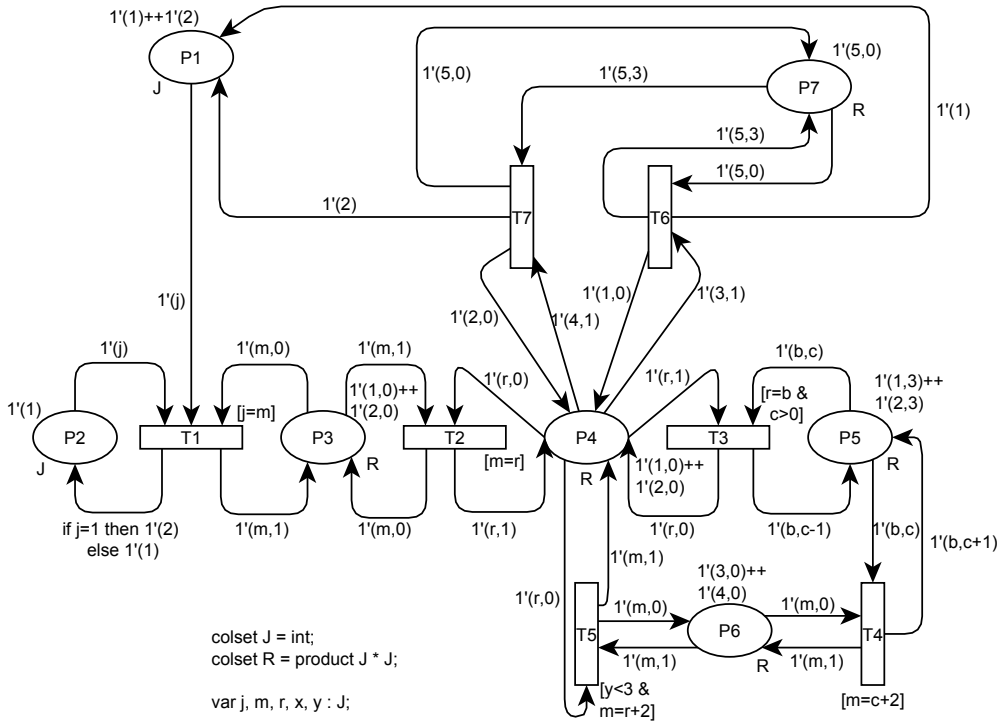


Fig. A.2. The CPN model of the AMS.

Table A.1. The interpretation of the places and transitions in the AMS CPN model.

Place	Description	Transition	Description
$P1$	Available parts for types F and G (F=1, G=2)	$T1$	Loading of raw materials into machines m_1, m_2
$P2$	Control of raw material flow into the system	$T2$	Unloading of parts from machines m_1, m_2 by robots
$P3$	Status of machines m_1, m_2 (0 = free, 1 = busy)	$T3$	Loading parts into buffers b_1, b_2 by robots
$P4$	Available robots (0 = free, 1 = busy)	$T4$	Unloading of parts from buffers into machines m_3, m_4
$P5$	Available buffers with finite capacity	$T5$	Unloading of parts from machines m_3, m_4 by robots
$P6$	Status of machines m_3, m_4 (0 = free, 1 = busy)	$T6$	Loading of part F into the assembly cell by robot r_1
$P7$	Assembly cell availability	$T7$	Assembly of parts

B
Paper V Appendix

B.1 Job sets

Job Set 1	Job Set 2
Job 1: M1(8); M2(16); M4(12)	Job 1: M1(10); M4(18)
Job 2: M1(20); M3(10); M2(18)	Job 2: M2(10); M4(18)
Job 3: M3(12); M4(8); M1(15)	Job 3: M1(10); M3(20);
Job 4: M4(14); M2(18)	Job 4: M2(10); M3(15); M4(12)
Job 5: M3(10); M1(15)	Job 5: M1(10); M2(15); M4(12)
	Job 6: M1(10); M2(15); M3(12)

Job Set 3	Job Set 4
Job 1: M1(16); M3(15)	Job 1: M4(11); M1(10); M2(7)
Job 2: M2(18); M4(15)	Job 2: M3(12); M2(10); M4(8)
Job 3: M1(20); M2(10)	Job 3: M2(7); M3(10); M1(9); M3(8)
Job 4: M3(15); M4(10)	Job 4: M2(7); M4(8); M1(12); M2(6)
Job 5: M1(8); M2(10); M3(15); M4(17)	Job 5: M1(9); M2(7); M4(8); M2(10); M3(8)
Job 6: M2(10); M3(15); M4(8); M1(15)	

Job Set 5	Job Set 6
Job 1: M1(6); M2(12); M4(9)	Job 1: M1(9); M2(11); M4(7)
Job 2: M1(18); M3(6); M2(15)	Job 2: M1(19); M2(20); M4(13)
Job 3: M3(9); M4(3); M1(12)	Job 3: M2(14); M3(20); M4(9)
Job 4: M4(6); M2(15)	Job 4: M2(14); M3(20); M4(9)
Job 5: M3(3); M1(9)	Job 5: M1(11); M3(16); M4(8)
	Job 6: M1(10); M3(12); M4(10)

Job Set 7	Job Set 8
Job 1: M1(6); M4(6)	Job 1: M2(12); M3(21); M4(11)
Job 2: M2(11); M4(9)	Job 2: M2(12); M3(21); M4(11)
Job 3: M2(9); M4(7)	Job 3: M2(12); M3(21); M4(11)
Job 4: M3(16); M4(7)	Job 4: M2(12); M3(21); M4(11)
Job 5: M1(9); M3(18)	Job 5: M1(10); M2(14); M3(18); M4(9)
Job 6: M2(13); M3(19); M4(6)	Job 6: M1(10); M2(14); M3(18); M4(9)
Job 7: M1(10); M2(9); M3(13)	
Job 8: M1(11); M2(9); M4(8)	

Job Set 9	Job Set 10
Job 1: M3(9); M1(12); M2(9); M4(6)	Job 1: M1(11); M3(19); M2(16); M4(13)
Job 2: M3(16); M2(11); M4(9)	Job 2: M2(21); M3(16); M4(14)
Job 3: M1(21); M2(18); M4(7)	Job 3: M3(8); M2(10); M1(14); M4(9)
Job 4: M2(20); M3(22); M4(11)	Job 4: M2(13); M3(20); M4(10)
Job 5: M3(14); M1(16); M2(13); M4(9)	Job 5: M1(9); M3(16); M4(18)
	Job 6: M2(19); M1(21); M3(11); M4(15)

B.2 Travel time data

Layout 1		L/U	Ml	M2	M3	M4
	L/U	0	6	8	10	12
	Ml	12	0	6	8	10
	M2	10	6	0	6	8
	M3	8	8	6	0	6
	M4	6	10	8	6	0

Layout 2		L/U	Ml	M2	M3	M4
	L/U	0	4	6	8	6
	Ml	6	0	2	4	2
	M2	8	12	0	2	4
	M3	6	10	12	0	2
	M4	4	8	10	12	0

Layout 3		L/U	Ml	M2	M3	M4
	L/U	0	2	4	10	12
	Ml	12	0	2	8	10
	M2	10	12	0	6	8
	M3	4	6	8	0	2
	M4	2	4	6	12	0

Layout 4		L/U	Ml	M2	M3	M4
	L/U	0	4	8	10	14
	Ml	18	0	4	6	10
	M2	20	14	0	8	6
	M3	12	8	6	0	6
	M4	14	14	12	6	0

References

[1] Bilge U, Ulusoy G. A time window approach to simultaneous scheduling of machines and material handling system in an FMS. Operations Research 1995;43(6):1058–1070.

[2] Zhou M, DiCesare F, Desrochers A. A hybrid methodology for synthesis of Petri net models for manufacturing systems. Robotics and Automation, IEEE Transactions on 1992;8(3):350–361.

[3] Viswanadham N, Narahari Y. Performance modeling of automated manufacturing systems. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.; 1992. ISBN 0-13-658824-7.

[4] Khilwani N, Harding JA, Mishra N. Tool selection in FMS: A hybrid sa-tabu algorithm based approach. In: Tiwari M, Harding JA, editors. Evolutionary Computing in Advanced Manufacturing; chap. 7. Hoboken, NJ, USA: John Wiley & Sons, Inc. ISBN 9781118161883; 2011, p. 123–150.

[5] Kagermann H. Change through digitization-value creation in the age of industry 4.0. In: Albach H, Meffert H, Pinkwart A, Reichwald R, editors. Management of Permanent Change. Springer Fachmedien Wiesbaden. ISBN 978-3-658-05013-9; 2015, p. 23–45.

[6] Schuh G, Potente T, Wesch-Potente C, Weber AR, Prote JP. Collaboration mechanisms to increase productivity in the context of industrie 4.0. Procedia CIRP 2014;19(0):51–56. 2nd CIRP Robust Manufacturing Conference (RoMac 2014).

[7] Shrouf F, Ordieres J, Miragliotta G. Smart factories in industry 4.0: A review of the concept and of energy management approached in production based on the internet of things paradigm. In: Industrial Engineering and Engineering Management (IEEM), 2014 IEEE International Conference on. 2014, p. 697–701.

[8] Maccarthy BL, Liu J. A new classification scheme for flexible manufacturing systems. International Journal of Production Research 1993;31(2):299–309.

[9] Wadhwa RS. Flexibility in manufacturing automation: A living lab case study of norwegian metalcasting SMEs. Journal of Manufacturing Systems 2012;31(4):444–454. Selected Papers of 40th North American Manufacturing Research Conference.

[10] Hamasha M, Alazzam A, Hamasha S, Aqlan F, Almeanazel O, Khasawneh M. Multimachine flexible manufacturing cell analysis using a markov chain-based approach. Components, Packaging and Manufacturing Technology, IEEE Transactions on 2015;5(3):439–446.

[11] Browne J, Dubois D, Rathmill K, Sethi SP, Stecke KE. Classification of flexible manufacturing systems. The FMS magazine 1984;2(2):114–117.

[12] Kusiak A. Flexible manufacturing systems: a structural approach. International Journal of Production Research 1985;23(6):1057–1073.

[13] Rachamadugu R, Stecke KE. Classification and review of FMS scheduling procedures. Production Planning & Control 1994;5(1):2–20.

[14] Ramaswamy S, Joshi S. Deadlock-free schedules for automated manufacturing workstations. Robotics and Automation, IEEE Transactions on 1996;12(3):391–400.

[15] Lacomme P, Larabi M, Tchernev N. Simultaneous scheduling of machines and automated guided vehicles: graph modelling and resolution. Powerpoint presentation page 3 www.isima.fr/~lacomme/doc/Presentation_Tchernev.ppt; 2015. Accessed: 2015-03-17.

[16] Gerwin D. An agenda for research on the flexibility of manufacturing processes. International Journal of Operations & Production Management 1987;7(1):38–49.

[17] Gupta YP, Goyal S. Flexibility of manufacturing systems: Concepts and measurements. European Journal of Operational Research 1989;43(2):119–135.

[18] Upton D. The management of manufacturing flexibility. California management review 1994;36(2):72–89.

[19] Seebacher G, Winkler H. A citation analysis of the research on manufacturing and supply chain flexibility. International Journal of Production Research 2013;51(11):3415–3427.

[20] Sethi A, Sethi S. Flexibility in manufacturing: A survey. International Journal of Flexible Manufacturing Systems 1990;2(4):289–328.

[21] Gerwin D. Manufacturing flexibility: A strategic perspective. Management Science 1993;39(4):395–410.

[22] Beach R, Muhlemann A, Price D, Paterson A, Sharp J. A review of manufacturing flexibility. European Journal of Operational Research 2000;122(1):41–57.

[23] Zhou M, Venkatesh K. Modeling, simulation, and control of flexible manufacturing systems: A Petri net approach. Series in intelligent control and intelligent automation; World Scientific; 1999. ISBN 9789810230296.

[24] Stecke K. Design, planning, scheduling, and control problems of flexible manufacturing systems. Annals of Operations Research 1985;3(1):1–12.

[25] Stecke K, Raman N. FMS planning decisions, operating flexibilities, and system performance. Engineering Management, IEEE Transactions on 1995;42(1):82–90.

[26] Xiong HH, Zhou M. Scheduling of semiconductor test facility via Petri nets and hybrid heuristic search. Semiconductor Manufacturing, IEEE Transactions on 1998;11(3):384–393.

[27] Tuncel G, Bayhan G. Applications of Petri nets in production scheduling: a review. The International Journal of Advanced Manufacturing Technology 2007;34(7-8):762–773.

[28] Xie C, Allen TT. Simulation and experimental design methods for job shop scheduling with material handling: a survey. The International Journal of Advanced Manufacturing Technology 2015;:1–11.

[29] Liu J, MacCarthy BL. The classification of FMS scheduling problems. International Journal of Production Research 1996;34(3):647–656.

[30] Sabuncuoglu I, Hommertzheim DL. Dynamic dispatching algorithm for scheduling machines and automated guided vehicles in a flexible manufacturing system. International Journal of Production Research 1992;30(5):1059–1079.

[31] Qiu L, Hsu WJ, Huang SY, Wang H. Scheduling and routing algorithms for AGVs: A survey. International Journal of Production Research 2002;40(3):745–760.

[32] Mejia G, Montoya C. Scheduling manufacturing systems with blocking: a Petri net approach. International Journal of Production Research 2009;47(22):6261–6277.

[33] Korbaa O, Benasser A, Yim P. Two FMS scheduling methods based on Petri nets: A global and a local approach. International Journal of Production Research 2003;41(7):1349–1371.

[34] Kim HJ, Lee JH, Lee TE. Non-cyclic scheduling of a wet station. Automation Science and Engineering, IEEE Transactions on 2014;11(4):1262–1274.

[35] Kim HJ, Lee JH, Lee TE. Noncyclic scheduling of cluster tools with a branch and bound algorithm. Automation Science and Engineering, IEEE Transactions on 2014;PP(99):1–11.

[36] Kim HJ, Lee JH, Lee TE. Time-feasible reachability tree for noncyclic scheduling of timed Petri nets. Automation Science and Engineering, IEEE Transactions on 2014;PP(99):1–10.

[37] Kim H, Choi J. An efficient one-step lookahead A* algorithm for PM-CT scheduling problems. The International Journal of Advanced Manufacturing Technology 2014;72(9-12):1481–1489.

[38] Chu F, Chu C, Desprez C. Series production in a basic re-entrant shop to minimize makespan or total flow time. Computers & Industrial Engineering 2010;58(2):257–268. Scheduling in Healthcare and Industrial Systems.

[39] Wikborg U, Lee TE. Noncyclic scheduling for timed discrete-event systems with application to single-armed cluster tools using pareto-optimal optimization. Automation Science and Engineering, IEEE Transactions on 2013;10(3):699–710.

[40] Dawande M, Geismar H, Sethi S, Sriskandarajah C. Sequencing and scheduling in robotic cells: Recent developments. Journal of Scheduling 2005;8(5):387–426.

[41] Lee DY, DiCesare F. Scheduling flexible manufacturing systems using Petri nets and heuristic search. Robotics and Automation, IEEE Transactions on 1994;10(2):123–132.

[42] Balogun OO, Popplewell K. Towards the integration of flexible manufacturing system scheduling. International Journal of Production Research 1999;37(15):3399–3428.

[43] Zurawski R, Zhou M. Petri nets and industrial applications: A tutorial. Industrial Electronics, IEEE Transactions on 1994;41(6):567–583.

[44] Murata T. Petri nets: Properties, analysis and applications. Proceedings of the IEEE 1989;77(4):541–580.

[45] Silva M, Valette R. Petri nets and flexible manufacturing. In: Rozenberg G, editor. Advances in Petri Nets 1989; vol. 424 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. ISBN 978-3-540-52494-6; 1990, p. 374–417.

[46] Silva M. Half a century after Carl Adam Petri's Ph.D. thesis: A perspective on the field. Annual Reviews in Control 2013;37(2):191–219.

[47] Jensen K, Kristensen LM. Coloured Petri nets: modelling and validation of concurrent systems. Springer; 2009.

[48] McDuffie EL, Cristofari M, Caron F, Tronci M, Wolfe WJ, Sorensen SE. Scheduling systems and techniques in flexible manufacturing systems. In: Leondes CT, editor. Computer-Aided Design, Engineering, and Manufacturing: Systems Techniques and Applications, Volume II, Computer-Integrated Manufacturing; chap. 7. CRC Press. ISBN 9781420049947; 2000, p. 1–45.

[49] Thomas M, Szczerbicka H. Evaluating online scheduling techniques in uncertain environments. In: Baptiste P, Kendall G, Munier-Kordon A, Sourd F, editors. 3rd Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA 2007). Paris, France; 2007, p. 471–479.

[50] Sabuncuoglu I, Karabuk S. Rescheduling frequency in an FMS with uncertain processing times and unreliable machines. Journal of Manufacturing Systems 1999;18(4):268–283. Special issue on scheduling: From Research Into Practice.

[51] Sabuncuoglu I, Kizilisik OB. Reactive scheduling in a dynamic and stochastic FMS environment. International Journal of Production Research 2003;41(17):4211–4231.

[52] Hillion H, Proth JM, Xie XL. A heuristic algorithm for the periodic scheduling and sequencing job-shop problem. In: Decision and Control, 1987. 26th IEEE Conference on; vol. 26. 1987, p. 612–617.

[53] Hillion H, Proth JM. Performance evaluation of job-shop systems using timed event-graphs. Automatic Control, IEEE Transactions on 1989;34(1):3–9.

[54] Carlier J, Chretienne P. Timed Petri net schedules. In: Rozenberg G, editor. Advances in Petri Nets 1988; vol. 340 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. ISBN 978-3-540-50580-8; 1988, p. 62–84.

[55] Jensen K, Kristensen L, Wells L. Coloured Petri nets and CPN Tools for modelling and validation of concurrent systems. International Journal on Software Tools for Technology Transfer 2007;9:213–254.

[56] Camurri A, Franchi P, Gandolfo F, Zaccaria R. Petri net based process scheduling: A model of the control system of flexible manufacturing systems. Journal of Intelligent and Robotic Systems 1993;8(1):99–123.

[57] Chincholkar A, Krishnaiah Chetty O. Stochastic coloured Petri nets for modelling and evaluation, and heuristic rule base for scheduling of FMS. The International Journal of Advanced Manufacturing Technology 1996;12(5):339–348.

[58] Krishnaiah Chetty O, Gnanasekaran O. Modelling, simulation and scheduling of flexible assembly systems with coloured Petri nets. The International Journal of Advanced Manufacturing Technology 1996;11(6):430–438.

[59] Lin JT, Lee CC. A Petri net-based integrated control and scheduling scheme for flexible manufacturing cells. Computer Integrated Manufacturing Systems 1997;10(2):109–122.

[60] Viswanadham N, Narahari Y. Coloured Petri net models for automated manufacturing systems. In: Robotics and Automation. Proceedings. 1987 IEEE International Conference on; vol. 4. 1987, p. 1985–1990.

[61] Jung C, Kim HJ, Lee TE. A branch and bound algorithm for cyclic scheduling of timed Petri nets. Automation Science and Engineering, IEEE Transactions on 2015;12(1):309–323.

[62] Reyes A, Yu H, Kelleher G, Lloyd S. Integrating Petri nets and hybrid heuristic search for the scheduling of FMS. Comput Ind 2002;47(1):123–138.

[63] Mejia G, Odrey NG. An approach using Petri nets and improved heuristic search for manufacturing system scheduling. Journal of Manufacturing Systems 2005;24(2):79–92.

[64] Yu H, Reyes A, Cang S, Lloyd S. Combined Petri net modelling and AI-based heuristic hybrid search for flexible manufacturing systems-part ii. heuristic hybrid search. Computers & Industrial Engineering 2003;44(4):545–566.

[65] Russell SJ, Norvig P. Artificial intelligence: a modern approach (3rd edition). Prentice Hall; 2009. ISBN 0136042597.

[66] Xing K, Han L, Zhou M, Wang F. Deadlock-free genetic scheduling algorithm for automated manufacturing systems based on deadlock control policy. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 2012;42(3):603–615.

[67] Caballero-Villalobos JP, Mejía-Delgadillo GE, García-Cáceres RG. Scheduling of complex manufacturing systems with Petri nets and genetic algorithms: a case on plastic injection moulds. The International Journal of Advanced Manufacturing Technology 2013;69(9-12):2773–2786.

[68] Han L, Xing K, Chen X, Lei H, Wang F. Deadlock-free genetic scheduling for flexible manufacturing systems using Petri nets and deadlock controllers. International Journal of Production Research 2014;52(5):1557–1572.

[69] Huang B, Sun Y, Sun YM. Scheduling of flexible manufacturing systems based on Petri nets and hybrid heuristic search. International Journal of Production Research 2008;46(16):4553–4565.

[70] Huang B, Sun Y, Sun YM, Zhao CX. A hybrid heuristic search algorithm for scheduling FMS based on Petri net model. The International Journal of Advanced Manufacturing Technology 2010;48(9-12):925–933.

[71] Piera M, Music G. Coloured Petri net scheduling models: Timed state space exploration shortages. Mathematics and Computers in Simulation 2011;82(3):428–441.

[72] Kwok YK, Ahmad I. On multiprocessor task scheduling using efficient state space search approaches. Journal of Parallel and Distributed Computing 2005;65(12):1515–1532.

[73] Sinnen O. Reducing the solution space of optimal task scheduling. Computers & Operations Research 2014;43(0):201–214.

[74] Chan FT, Bhagwat R, Chan HK. The effect of responsiveness of the control-decision system to the performance of FMS. Computers & Industrial Engineering 2014;72:32–42.

[75] Baruwa OT, Piera MA. TIMSPAT –TIMed State space Performance Analysis Tool for colored Petri net-based scheduling of discrete event systems: An application to flexible manufacturing systems; 2015. Submitted for publication in Computers & Industrial Engineering.

[76] Russell SJ, Norvig P. Artificial Intelligence: A Modern Approach. Pearson Education; 2003. ISBN 0137903952.

[77] Zhou R, Hansen EA. Breadth-first heuristic search. Artificial Intelligence 2006;170(4-5):385–408.

[78] Baruwa OT, Piera MA, Guasch A. Deadlock-free scheduling method for flexible manufacturing systems based on timed colored Petri nets and anytime heuristic search. Systems, Man, and Cybernetics: Systems, IEEE Transactions on 2015;45(5):831–846.

[79] Moro A, Yu H, Kelleher G. Advanced scheduling methodologies for flexible manufacturing systems using Petri nets and heuristic search. In: Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on; vol. 3. 2000, p. 2398–2403 vol.3.

[80] Baruwa OT, Piera MA. Identifying FMS repetitive patterns for efficient search-based scheduling algorithm: A colored Petri net approach. Journal of Manufacturing Systems 2015;35(0):120–135.

[81] Baruwa OT, Piera MA. Anytime heuristic search for scheduling flexible manufacturing systems: a timed colored Petri net approach. The International Journal of Advanced Manufacturing Technology 2014;75(1-4):123–137.

[82] Vadlamudi S, Gaurav P, Aine S, Chakrabarti P. Anytime column search. In: Thielscher M, Zhang D, editors. AI 2012: Advances in Artificial Intelligence; vol. 7691 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. ISBN 978-3-642-35100-6; 2012, p. 254–265.

[83] Zhang W, Korf RE. Performance of linear-space search algorithms. Artificial Intelligence 1995;79(2):241–292.

[84] Zhang W. Truncated and anytime depth-first branch-and-bound: A case study on the asymmetric traveling salesman problem. In: AAAI 1999 Spring Symposium on Search Techniques for Problem Solving under Uncertainty and Incomplete Information. Stanford, CA; 1999, p. 148–153.

[85] Malone B, Yuan C. A depth-first branch and bound algorithm for learning optimal bayesian networks. In: Croitoru M, Rudolph S, Woltran S, Gonzales C, editors. Graph Structures for Knowledge Representation and Reasoning; vol. 8323 of *Lecture Notes in Computer Science*. Springer International Publishing. ISBN 978-3-319-04533-7; 2014, p. 111–122.

[86] Cavalieri S, Mirabella O, Marroccia S. Improving flexible semiconductor manufacturing system performance by a coloured Petri net-based scheduling algorithm. In: Emerging Technologies and Factory Automation Proceedings, 1997. ETFA '97., 1997 6th International Conference on. 1997, p. 369–374.

[87] Baruwa OT, Piera MA. A colored Petri net-based hybrid heuristic search approach to simultaneous scheduling of machines and automated guided vehicles; 2015. Revised manuscript submitted for publication in International Journal of Production Research.

[88] Jun T, Piera MA, Ruiz S. A causal model to explore the ACAS induced collisions. Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering 2014;228(10):1735–1748.

[89] Jun T, Piera MA, Nosedal J. Analysis of induced traffic alert and collision avoidance system collisions in unsegregated airspace using a colored Petri net model. SIMULATION 2015;.

[90] Tang J, Piera MA, Baruwa OT. A discrete-event modeling approach for the analysis of TCAS-induced collisions with different pilot response times. Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering 2015;.

[91] Nosedal J, Piera MA, Ruiz S. A causal model to schedule efficient ground delays in present air traffic management systems modeling and simulation for complex networks management. In: Proceedings of the 2013 Summer Computer Simulation Conference. SCSC '13; Vista, CA: Society for Modeling & Simulation International. ISBN 978-1-62748-276-9; 2013, p. 1–8.

[92] Zuñiga CA, Piera MA, Baruwa OT. Pre-tactical trajectory de-confliction algorithm for air traffic management. In: 1st International Conference on Application and Theory of Automation in Command and Control Systems ATTACS2012. IRIT Press. ISBN 978-2-917490-20-4; 2012, p. 233–237.

[93] Nosedal J, Baruwa O, Piera MA. Concurrent and distributed systems analysis using colored Petri nets. In: Actas de XXXIV Jornadas de Automática. Terrassa, Spain; 2013, p. 538–544.

[94] Petri nets tool database. http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/db.html; 2015. Accessed: 14-Feb-2015.

[95] Basile F, Carbone C, Chiacchio P. Simulation and analysis of discrete-event control systems based on Petri nets using PNetlab. Control Engineering Practice 2007;15(2):241–259.

[96] Kounev S, Buchmann A. SimQPN-a tool and methodology for analyzing queueing petri net models by means of simulation. Performance Evaluation 2006;63(4-5):364–394.

[97] Julvez J, Matcovschi M, Pastravanu O. MATLAB tools for the analysis of Petri net models. In: Emerging Technology and Factory Automation (ETFA), 2014 IEEE. 2014, p. 1–12.

[98] Mahulea C, Barsan L, Pastravanu O. Matlab tools for Petri-net-based approaches to flexible manufacturing systems. In: F.G. Filip ID, Iliescu S, editors. 9th IFAC Symposium on Large Scale Systems LSS 2001. 2001, p. 18–20.

[99] Davidrajuh R, Lin B. Exploring airport traffic capability using Petri net based model. Expert Systems with Applications 2011;38(9):10923–10931.

[100] Aized T. Modelling and performance maximization of an integrated automated guided vehicle system using coloured petri net and response surface methods. Computers & Industrial Engineering 2009;57(3):822–831.

[101] Aized T. Modelling and analysis of multiple cluster tools system with random failures using coloured Petri net. The International Journal of Advanced Manufacturing Technology 2010;50(9-12):897–906.

[102] He X, Wu Z. Deadlock-free assignment of wafer processing in photolithography equipment - by using a CPN model. Transactions of the Institute of Measurement and Control 2011;33(3-4):422–434.

[103] Westergaard M, Evangelista S, Kristensen L. ASAP: An extensible platform for state space analysis. In: Franceschinis G, Wolf K, editors. Applications and Theory of Petri Nets; vol. 5606 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. ISBN 978-3-642-02423-8; 2009, p. 303–312.

[104] Westergaard M, Verbeek HE. Efficient implementation of prioritized transitions for high-level Petri nets. In: International Workshop on Petri Nets and Software Engineering; vol. 723. Newcastle upon Tyne, UK; 2011, p. 27–41.

[105] Gaeta R. Efficient discrete-event simulation of colored petri nets. Software Engineering, IEEE Transactions on 1996;22(9):629–639.

[106] Liu F, Heiner M. Computation of enabled transition instances for colored petri nets. In: 17th German Workshop on Algorithms and Tools for Petri Nets (AWPN); vol. 643. 2010, p. 51–65.

[107] Evangelista S, Pradat-Peyre JF. An efficient algorithm for the enabling test of colored Petri nets. In: Fifth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools. 570; 2004, p. 137–156.

[108] Mota MM, Piera MA. A compact timed state space approach for the analysis of manufacturing systems: key algorithmic improvements. International Journal of Computer Integrated Manufacturing 2011;24(2):135–153.

[109] van der Aalst W. Interval timed coloured Petri nets and their analysis. In: Ajmone Marsan M, editor. Application and Theory of Petri Nets 1993; vol. 691 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. ISBN 978-3-540-56863-6; 1993, p. 453–472.

[110] Lakos C, Petrucci L. Modular state space exploration for timed Petri nets. International Journal on Software Tools for Technology Transfer 2007;9(3-4):393–411.

[111] Christensen S, Kristensen LM, Mailund T. Condensed state spaces for timed Petri nets. In: Colom JM, Koutny M, editors. Applications and Theory of Petri Nets 2001; vol. 2075 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. ISBN 978-3-540-42252-5; 2001, p. 101–120.

[112] Mujica M, Piera MA, Narciso M. Revisiting state space exploration of timed coloured Petri net models to optimize manufacturing system's performance. Simulation Modelling Practice and Theory 2010;18(9):1225 – 1241.

[113] Narciso ME, Piera MA, Guasch A. A time stamp reduction method for state space exploration using colored Petri nets. Simulation 2012;88(5):592–616.

[114] Lee J, Lee JS. Heuristic search for scheduling flexible manufacturing systems using lower bound reachability matrix. Computers & Industrial Engineering 2010;59(4):799–806.

[115] Li C, Wu W, Feng Y, Rong G. Scheduling FMS problems with heuristic search function and transition-timed Petri nets. Journal of Intelligent Manufacturing 2014;:1–12.

[116] Zhou R, Hansen EA. Beam-stack search: Integrating backtracking with beam search. In: Biundo S, Myers KL, Rajan K, editors. ICAPS. AAAI. ISBN 1-57735-220-3; 2005, p. 90–98.

[117] Ulusoy G, Sivrikaya-Serifoglu F, Bilge U. A genetic algorithm approach to the simultaneous scheduling of machines and automated guided vehicles. Computers & Operations Research 1997;24(4):335–351.

[118] Shih H, Sekiguchi T. A timed Petri net and beam search based online FMS scheduling system with routing flexibility. In: Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on; vol. 3. 1991, p. 2548–2553 vol.3.

[119] Lee DY, DiCesare F. Integrated scheduling of flexible manufacturing systems employing automated guided vehicles. Industrial Electronics, IEEE Transactions on 1994;41(6):602–610.

[120] Jeng M, Chen S. A heuristic search approach using approximate solutions to Petri net state equations for scheduling flexible manufacturing systems. International Journal of Flexible Manufacturing Systems 1998;10(2):139–162.

[121] Jeng MD, Chen SC. Heuristic search based on Petri net structures for FMS scheduling. Industry Applications, IEEE Transactions on 1999;35(1):196–202.

[122] Jeng M, Lin C, Huang Y. Petri net dynamics-based scheduling of flexible manufacturing systems with assembly. Journal of Intelligent Manufacturing 1999;10(6):541–555.

[123] Elmekkawy TY, Elmaraghy HA. Efficient search of Petri nets for deadlock-free scheduling in FMSs using heuristic functions. International Journal of Computer Integrated Manufacturing 2003;16(1):14–24.

[124] Abdallah I, El Maraghy H, El Mekkawy T. An efficient search algorithm for deadlock-free scheduling in FMS using Petri nets. In: Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on; vol. 2. 1998, p. 1793–1798 vol.2.

[125] Huang B, Jiang R, Zhang G. Comments on "heuristic search for scheduling flexible manufacturing systems using lower bound reachability matrix". Computers & Industrial Engineering 2014;67(0):235 – 236.

[126] Huang B, Jiang R, Zhang G. Search strategy for scheduling flexible manufacturing systems simultaneously using admissible heuristic functions and nonadmissible heuristic functions. Computers & Industrial Engineering 2014;71(0):21–26.

[127] Luo J, Xing K, Zhou M, Li X, Wang X. Deadlock-free scheduling of automated manufacturing systems using Petri nets and hybrid heuristic search. Systems, Man, and Cybernetics: Systems, IEEE Transactions on 2014;PP(99):1–1.

[128] Huang B, Shi XX, Xu N. Scheduling FMS with alternative routings using Petri nets and near admissible heuristic search. The International Journal of Advanced Manufacturing Technology 2012;63(9-12):1131–1136.

[129] Jensen K, Kristensen LM, Mailund T. The sweep-line state space exploration method. Theoretical Computer Science 2012;429(0):169–179.

[130] Korf RE, Zhang W, Thayer I, Hohwald H. Frontier search. J ACM 2005;52(5):715–748.

[131] Della Penna G, Intrigila B, Melatti I, Tronci E, Venturini Zilli M. Exploiting transition locality in automatic verification of finite-state concurrent systems. International Journal on Software Tools for Technology Transfer (STTT) 2004;6:320–341.

[132] Lamborn P, Hansen EA. Layered duplicate detection in external-memory model checking. In: Proc. of the 15th international workshop on Model Checking Software. SPIN '08; Berlin, Heidelberg: Springer. ISBN 978-3-540-85113-4; 2008, p. 160–175.

[133] Mujica M, Piera MA. Hybrid search algorithm to optimize scheduling problems for TCPN models. In: Proceedings of the 2010 Summer Computer Simulation Conference. SCSC '10; San Diego, CA, USA: Society for Computer Simulation International; 2010, p. 461–468.

[134] Mujica Mota MA, Piera Eroles MA. Time line search for the state space-based optimization algorithm for timed coloured Petri nets. In: Management and Control of Production and Logistics. ISBN 978-3-902661-81-4; 2010, p. 144–151.

[135] Mujica Mota M, Piera MA. An improved time line search algorithm for manufacturing decision-making. International Journal of Production Research 2014;52(4):1116–1132.

[136] Christensen S, Jensen K, Mailund T, Kristensen LM. State space methods for timed coloured Petri nets. In: In Proceedings of 2nd International Colloquium on Petri Net Technologies for Modelling Communication Based Systems. 2001, p. 33–42.

[137] Kristensen LM, Mailund T. A generalised sweep-line method for safety properties. In: In Proc. of FME'02, volume 2391 of LNCS. Springer-Verlag; 2002, p. 549–567.

[138] Sun TH, Cheng CW, Fu LC. A Petri net based approach to modeling and scheduling for an FMS and a case study. Industrial Electronics, IEEE Transactions on 1994;41(6):593–601.

[139] Xiong HH, Zhou M, Caudill R. A hybrid heuristic search algorithm for scheduling flexible manufacturing systems. In: Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on; vol. 3. 1996, p. 2793–2797 vol.3.

[140] Xiong HH, Zhou M. Deadlock-free scheduling of an automated manufacturing system based on Petri nets. In: Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on; vol. 2. 1997, p. 945–950 vol.2.

[141] Moro A, Yu H, Kelleher G. Hybrid heuristic search for the scheduling of flexible manufacturing systems using Petri nets. Robotics and Automation, IEEE Transactions on 2002;18(2):240–245.

[142] Abdallah IB, Elmaraghy HA, Elmekkawy T. Deadlock-free scheduling in flexible manufacturing systems using Petri nets. International Journal of Production Research 2002;40(12):2733–2756.

[143] Mejía G, Montoya C. A Petri net based algorithm for minimizing total tardiness in flexible manufacturing systems. Annals of Operations Research 2008;164(1):63–78.

[144] Mejía G, Montoya C. Applications of resource assignment and scheduling withăPetri nets and heuristic search. Annals of Operations Research 2010;181(1):795–812.

[145] Kim YW, Suzuki T, Narikiyo T. FMS scheduling based on timed Petri net model and reactive graph search. Applied Mathematical Modelling 2007;31(6):955–970.

[146] Korf RE. Real-time heuristic search. Artificial Intelligence 1990;42(2-3):189–211.

[147] Mujica MA, Piera MA. Performance optimisation of a CNC machine through exploration of timed state space. International Journal of Simulation and Process Modelling 2010;6(2):165–174.

[148] Ganesharajah T, Hall N, Sriskandarajah C. Design and operational issues in AGV-served manufacturing systems. Annals of Operations Research 1998;76(0):109–154.

[149] Abdelmaguid TF, Nassef AO. A constructive heuristic for the integrated scheduling of machines and multiple-load material handling equipment in job shops. The International Journal of Advanced Manufacturing Technology 2010;46(9-12):1239–1251.

[150] Sawik T. A multilevel machine and vehicle scheduling in a flexible manufacturing system. Mathematical and Computer Modelling 1996;23(7):45–57.

[151] Anwar MF, Nagi R. Integrated scheduling of material handling and manufacturing activities for just-in-time production of complex assemblies. International Journal of Production Research 1998;36(3):653–681.

[152] Reddy B, Rao C. A hybrid multi-objective ga for simultaneous scheduling of machines and AGVs in FMS. The International Journal of Advanced Manufacturing Technology 2006;31(5-6):602–613.

[153] Lacomme P, Moukrim A, Tchernev N. Simultaneous job input sequencing and vehicle dispatching in a single-vehicle automated guided vehicle system: a heuristic branch-and-bound approach coupled with a discrete events simulation model. International Journal of Production Research 2005;43(9):1911–1942.

[154] Nilsson NJ. Principles of Artificial Intelligence. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 1980. ISBN 0-934613-10-9.

[155] Wang Q, Wang Z. Hybrid heuristic search based on petri net for FMS scheduling. Energy Procedia 2012;17, Part A(0):506–512.

[156] Jawahar N, Aravindan P, Ponnambalam S, Suresh R. AGV schedule integrated with production in flexible manufacturing systems. The International Journal of Advanced Manufacturing Technology 1998;14(6):428–440.

[157] Hansen EA, Zhou R. Anytime heuristic search. J Artif Intell Res(JAIR) 2007;28:267–297.

[158] Erol R, Sahin C, Baykasoglu A, Kaplanoglu V. A multi-agent based approach to dynamic scheduling of machines and automated guided vehicles in manufacturing systems. Applied Soft Computing 2012;12(6):1720–1732.

[159] Raman N, F.B. T, R.V. R. Simultaneous scheduling of machines and material handling devices in automated manufacturing. In: Stecke KE, Suri R, editors. Second ORSA/TIMS Conference on Flexible Manufacturing Systems. Elsevier; 1986, p. 455–466.

[160] Suri R, Desiraju R. Performance analysis of flexible manufacturing systems with a single discrete material-handling device. International Journal of Flexible Manufacturing Systems 1997;9(3):223–249.

[161] Khayat GE, Langevin A, Riopel D. Integrated production and material handling scheduling using mathematical programming and constraint programming. European Journal of Operational Research 2006;175(3):1818–1832.

[162] Caumond A, Lacomme P, Moukrim A, Tchernev N. An MILP for scheduling problems in an FMS with one vehicle. European Journal of Operational Research 2009;199(3):706–722.

[163] Nishi T, Hiranaka Y, Grossmann IE. A bilevel decomposition algorithm for simultaneous production scheduling and conflict-free routing for automated guided vehicles. Computers & Operations Research 2011;38(5):876–888.

[164] Zheng Y, Xiao Y, Seo Y. A tabu search algorithm for simultaneous machine/AGV scheduling problem. International Journal of Production Research 2014;52(19):5748–5763.

[165] Raju KR, Chetty OVK. Design and evaluation of automated guided vehicle systems for flexible manufacturing systems: an extended timed Petri net-based approach. International Journal of Production Research 1993;31(5):1069–1096.

[166] Lacomme P, Larabi M, Tchernev N. Job-shop based framework for simultaneous scheduling of machines and automated guided vehicles. International Journal of Production Economics 2013;143(1):24–34.

[167] Abdelmaguid TF, Nassef AO, Kamal BA, Hassan MF. A hybrid GA/heuristic approach to the simultaneous scheduling of machines and automated guided vehicles. International Journal of Production Research 2004;42(2):267–281.

[168] Jerald J, Asokan P, Saravanan R, Rani ADC. Simultaneous scheduling of parts and automated guided vehicles in an FMS environment using adaptive genetic algorithm. The International Journal of Advanced Manufacturing Technology 2006;29(5-6):584–589.

[169] Chaudhry I, Mahmood S, Shami M. Simultaneous scheduling of machines and automated guided vehicles in flexible manufacturing systems using genetic algorithms. Journal of Central South University of Technology 2011;18(5):1473–1486.

[170] Kumar M, Janardhana R, Rao C. Simultaneous scheduling of machines and vehicles in an FMS environment with alternative routing. The International Journal of Advanced Manufacturing Technology 2011;53(1-4):339–351.

[171] Gnanavel Babu A, Jerald J, Noorul Haq A, Muthu Luxmi V, Vigneswaralu T. Scheduling of machines and automated guided vehicles in FMS using differential evolution. International Journal of Production Research 2010;48(16):4683–4699.

[172] Deroussi L, Gourgand M, Tchernev N. A simple metaheuristic approach to the simultaneous scheduling of machines and automated guided vehicles. International Journal of Production Research 2008;46(8):2143–2164.

[173] Hurink J, Knust S. Tabu search algorithms for job-shop problems with a single transport robot. European Journal of Operational Research 2005;162(1):99–111. Logistics: From Theory to Application.

[174] Jensen K. Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts; vol. 1 of *Monographs in Theoretical Computer Science: an EATCS series*. 2. ed., 2. corr. printing ed.; Springer; 1997. ISBN 3540609431.

[175] Jensen K. Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. Volume 2, Analysis Methods; vol. 2 of *Monographs in Theoretical Computer Science: an EATCS series*. Springer; 1995. ISBN 3-540-58276-2.

[176] Egbelu PJ, Tanchoco JMA. Characterization of automatic guided vehicle dispatching rules. International Journal of Production Research 1984;22(3):359–374.

[177] Korf RE. Artificial intelligence search algorithms. In: Atallah MJ, editor. Handbook of Algorithms and Theory of Computation. CRC Press; 1996, p. 36–1 to 36–20.

[178] Li W, Nault BR, Xue D, Tu Y. An efficient heuristic for adaptive production scheduling and control in one-of-a-kind production. Computers & Operations Research 2011;38(1):267–276. Project Management and Scheduling.

[179] Taghaboni-Dutta F, Tanchoco JMA. Comparison of dynamic routeing techniques for automated guided vehicle system. International Journal of Production Research 1995;33(10):2653–2669.

[180] Vis IF. Survey of research in the design and control of automated guided vehicle systems. European Journal of Operational Research 2006;170(3):677–709.

[181] Reveliotis S. Conflict resolution in AGV systems. IIE Transactions 2000;32(7):647–659.

[182] Yoo Jw, Sim ES, Cao C, Park JW. An algorithm for deadlock avoidance in an AGV system. The International Journal of Advanced Manufacturing Technology 2005;26(5-6):659–668.

[183] Saidi-Mehrabad M, Dehnavi-Arani S, Evazabadian F, Mahmoodian V. An ant colony algorithm (ACA) for solving the new integrated model of job shop scheduling and conflict-free routing of AGVs. Computers & Industrial Engineering 2015;(0).

[184] Vempaty NR, Kumar V, Korf RE. Depth-first versus best-first search. In: AAAI. 1991, p. 434–440.

[185] Zamani R. A parallel complete anytime procedure for project scheduling under multiple resource constraints. The International Journal of Advanced Manufacturing Technology 2010;50(1-4):353–362.