SCHEDULING OF FLEXIBLE MANUFACTURING SYSTEMS INTEGRATING PETRI NETS AND ARTIFICIAL INTELLIGENCE METHODS.

Antonio Reyes Moro.

Submitted for the degree of Doctor of Philosophy

School of Computing and Mathematical Sciences
Liverpool John Moores University

2000

Acknowledgements:

I would like to thank to my three supervisors, Professor Gerry Kelleher, Dr Hongnian Yu, and Keith Whiteley for their valuable support. Thanks to Gerry for being patient with my English-Spanish-Chinese writing and for giving me guidance and support. Thanks to all my friends and colleagues at CMS. Many thanks to Madjid for his support and advice. Thanks to Liverpool.

Gracias Diego ("El Gacho") porque es mi amigo. A Rita (Tarri) y Ramón (Tito), a Chris (notepreocupess), a Alfredo (Dete) y Cristina y a Juani por los Buenos Ratos. A los de Málaga (Antonio, José, Sonia y Mª Ángeles). A Miguel, por su ayuda. Gracias a mi familia (toda) Papa, Mama, Viqui y Alber, sin vuestra seguridad, animo y apoyo no hubiera podido conseguirlo.Gracias a ti Ana, amor mío, porque has compartido conmigo los malos momentos, y porque hiciste que no tuviera que preocuparme sino de lo principal. Tú eres maravillosa.

Liverpool, 9-12-2000

Finalmente, esta tesis esta dedicada a mi padre. Porque le admiro, porque le debo tanto, porque es para mí la referencia de tantas cosas. Porque solo podría culparle, si cabe, de que a veces, solo a veces, sienta el temor de no poder ser como él.

A mi padre, Antonio Reyes Vázquez.

Bellaterra (Barcelona) Febrero de 2001.

Abstract.

The work undertaken in this thesis is about the integration of two well-known methodologies: Petri net (PN) modelling/analysis of industrial production processes and Artificial Intelligence (AI) optimisation search techniques. The objective of this integration is to demonstrate its potential in solving a difficult and widely studied problem, the scheduling of Flexible Manufacturing Systems (FMS).

This work builds on existing results that clearly show the convenience of PNs as a modelling tool for FMS. It addresses the problem of the integration of PN and AI based search methods. Whilst this is recognised as a potentially important approach to the scheduling of FMS there is a lack of any clear evidence that practical systems might be built. This thesis presents a novel scheduling methodology that takes forward the current state of the art in the area by:

Firstly presenting a novel modelling procedure based on a new class of PN (cb-NETS) and a language to define the essential features of basic FMS, demonstrating that the inclusion of high level FMS constraints is straightforward.

Secondly, we demonstrate that PN analysis is useful in reducing search complexity and presents two main results: a novel heuristic function based on PN analysis that is more efficient than existing methods and a novel reachability scheme that avoids futile exploration of candidate schedules.

Thirdly a novel scheduling algorithm that overcomes the efficiency drawbacks of previous algorithms is presented. This algorithm satisfactorily overcomes the complexity issue while achieving very promising results in terms of optimality.

Finally, this thesis presents a novel hybrid scheduler that demonstrates the convenience of the use of *PN* as a representation paradigm to support hybridisation between traditional *OR* methods, AI systematic search and stochastic optimisation algorithms. Initial results show that the approach is promising.

Table of Contents

Chapter 1. **Introduction.** 1

1. Flexible Manufacturing Systems (*FMS*). 1
2. Scheduling of *FMS*. 2
3. Scheduling objectives. 3
4. Scheduling *FMS* based on *PN* modelling and *AI* techniques. 4
5. Thesis structure. 6

Chapter 2. **Literature review.**

1. Introduction. 9
2. Flexible Manufacturing Systems and scheduling. 9
3. Classification of scheduling methods. 10
3.1. Operational research optimisation. 11
3.1.1. Operational research methods. 11
3.1.2. Drawbacks of *OR* methods when applied to *FMS* structures. 12
3.1.3. Genetic Algorithms. 13
3.2. Heuristics and simulation 15
3.2.1. Dispatch rules. 15
3.2.2. Simulation-based systems. 16
3.3. Artificial intelligence methods. 17
3.3.1. Expert systems. 17
3.3.2. Planning. 18
4. Integrating Petri Nets and *AI*. 19
4.1. Representation and reasoningn. 19
4.2. *PN* as a representation tool for *FMS*. 19
4.3. *PN* as a state-space definition. 21
5. *AI* problem solving in manufacturing optimisation. 22
6. A review of *PN* based scheduling of manufacturing processes. 25
6.1. On-line scheduling based on *PN* simulation. 25
6.2. *PN* based heuristic search. 27
7. Summary. 39

Chapter 3. ***PN* modelling of *FMS* and scheduling based on *PN* reachability analysis.**

1. Introduction. 41
2. *FMS* formulation. 41
3. Constrained buffer Petri nets *cb-NETs*. 43
4. Scheduling *FMS* based on *PN* dynamics; heuristic search within the *PN* reachability graph. 47
4.1. Flexible manufacturing system Modelling Language *FmsML*. 49
4.1.1. A *top-down* synthesis procedure based on the *FmsML* syntax 51
4.2. *cb-NET* simulator. 55

4.3. Search algorithms based in *cb-NET* structures..... 55
4.3.1. Branch & Bound..... 56
4.3.2. Best-First *A**..... 57
4.4. The *NP-hard* nature of the *FMS* scheduling problem..... 59
5. Case studies...... 62
6. Summary and conclusions...... 66

Chapter 4. **Heuristics from Petri nets.**

1. Introduction...... 68
2. Search reduction based on heuristic relaxation...... 69
3. *b-NETs*...... 74
3.1. *b-NET* definitions and properties..... 75
3.2. *b-NET* generation from *FMS* definitions..... 77
4. Resource Cost Reachability Matrix *(RCR).*..... 80
4.1. Formal definition of the *RCR* Matrix..... 81
4.2. An algorithm for the computation of the *RCR* matrix..... 83
5. Generating a heuristic function from the *RCR* matrix...... 84
5.1. *RCR* based heuristic function $h_{RCR}(M)$ definition and properties..... 85
5.2. An algorithm for the calculation of $h_{RCR}(M)$..... 87
5.3. Example of $h_{RCR}(M)$ application..... 88
6. Experimental results...... 90
7. Summary...... 94

Chapter 5. **Reachability graph truncation based on *PN* dynamics and heuristics.**

1. Introduction...... 95
2. Identifying previously explored markings...... 96
2.1. Similar and more promising markings..... 96
2.2. A safe test that guarantees optimality..... 98
2.3. The complete *cb-NET A** algorithm..... 100
2.4. Experimental comparison with different check tests..... 101
3. Controlled generation of successors: *CGS*...... 103
3.1. Description of the method..... 105
3.2. *CGS* algorithm..... 109
3.3. Example application..... 112
4. Experimental results...... 113
4.1. Integration with *A**..... 114
4.2. Integration with *Branch & Bound*..... 116
4.3. Integration with an incomplete search procedure..... 117
5. Summary...... 119

Chapter 6. **PN-based hybrid heuristic search.**

1. Introduction. 120
2. Review of previous approaches. 122
3. *PN* based incomplete search. *HST* and *DWS* algorithms. 125
3.1. Heuristic selection of transitions: *HST*. 125
3.2. Dynamic window stage search algorithm: *DWS**. 130
3.3. Analysis of *HST* and *DWS*: motivations for a new Algorithm 133
4. Dynamic look-ahead stage search: *DLSS.** 137
4.1. The *breadth-search* effect 137
4.2. *DLSS** *description*. 139
4.3. Geometry of the search frame. 140
4.4. Introducing irrevocable decisions: *Dynamics of SF*. 141
4.5. Inclusion criterion. 144
4.6. Final algorithm, integration of *A**, *HST*, *CGS* and *SF policy*: *DLSS**. 146
4.7. Deadlock avoidance and *DLSS**. 146
4.8. Effects of the heuristic function on irrevocable decisions 147
5. Experimental Results. 150
5.1. Empirical study of computational costs of *DLSS**. 151
5.2. Optimality of the algorithm. 153
5.3. Comparison with previous approaches. 155
5.4. Comparison with benchmark problems 156
6. Summary. 156

Chapter 7. **An evolutionary Hybrid scheduler for *FMS*.**

1. Introduction. 159
2. Background. 160
2.1. *FMS* scheduling approaches based on decomposition. 160
2.2. *FMS* scheduling based on Genetic algorithms. 161
2.3. The proposed approach 163
3. An evolutionary hybrid scheduler based on *PN* structures for *FMS*. 164
3.1. Overview of the method 166
3.2. Truncating the problem. 167
3.3. Obtaining the initial solutions. 168
3.4. Representing sub-schedules. 168
3.5. Building larger solutions. 168
3.6. Actualising the population. 171
3.7. Joining two sub-schedules. 171
3.8. Calculating the fitness function. 173
3.9. Termination of the algorithm. 174
3.10. A case study 174
4. Experimental results. 177
5. Summary. 179

Chapter 8. **Conclusion and future work.**

1. Summary of the thesis. 181
2. Current and future work. 183
2.1. Further improvement and development of *DLSS** and *BSPC* algorithms. 184
2.1.1. A sophistication of *DLSS**. 184
2.1.2. A full *Genetic Algorithm BSPC* approach. 186
2.2. Application to real industrial systems. 188
2.2.1. Increase the complexity of the *FMS* formulation. 189
2.2.2. Integration with results in *PN* applied to manufacturing control 189
2.2.3. Adaptation of *DLSS** to on/line scheduling scenarios. 189

References. 195

Chapter 1. Introduction.

1. Flexible Manufacturing Systems.

Over the past thirty years, a new type of production system has emerged: the Flexible Manufacturing System (*FMS*) [Ranky 90] [Parrish 90]. Traditionally, transfer lines have been used for the manufacturing of large volumes of single items. In such systems changing a transfer line to the production of a different item is costly in terms of lost production. This, together with the increase of medium variety/medium volume markets, has generated concern about the low productivity of traditional production methods [Kimemia 85]. Although the flexibility to manufacture a range of products can be achieved at the expense of efficiency, the development of new automated systems, such as flexible manufacturing systems, aims to provide the flexibility of job shop with close to the efficiency of large volume manufacturing. In essence, *FMS* are hoped to provide potential for increasing the throughput rate by reducing the effect of the shortage of manufacturing resources while still being able to adapt to dynamic production demands of multiple types of products. Additional advantages include reduction in space used and work-in-process (*WIP*) inventory requirements.

In terms of organisation, an *FMS* is a network formed by versatile computer numerically controlled manufacturing machines (CNC) and storage buffers, which are linked to automated material handling systems or automated guided vehicles (AGV). Flexibility, both in production batches and resource usage, is mainly determined by the existence of machines with multiple capabilities and the minimisation of set-up times and tool changeover. This allows the use of alternative production plans for each product, and the possibility of using different resources within the same plan. In other words, parts need not be processed in batches and each part of an order can be scheduled as a separate unit and may follow any available plan [Y. D. Kim 94].

The key to this sophistication is the absence of a physical production line, or in other words: the machine layout does not correspond to the sequences of machine utilisation, thus any location on the shop floor has to be reachable from any other [Silva 89]. The automatic transport system is in charge of moving parts and tools through the facility.

But despite their promise, the design and subsequent operation of *FMS*s have proven disappointing [Hutchison 91]. One aspect of *FMS* operation that has been particularly difficult is scheduling.

2. Scheduling of *FMS*.

Productivity in a batch system devoted to the continuous manufacturing of a single product is determined by the way it is designed. This means that analysis concentrates on bottlenecks, operation times and line balancing that are determined at the design stage. The operation phase is essentially a continuous mode free of decision making.

In the *FMS* scenario the relatively low size, but variable and changing production demands, have their processing optimised by using the flexibility design of the system. This means that they compete for the use of resources that are not eternally assigned to product types. It can be said that an *FMS* is a fully reprogrammable system [Ranky 1990] and consequently, it needs to be strongly controlled. The flexibility in a *FMS* basically introduces two decision variables to the control: a) *how* things are done and b) *when* things are done. The first determines which, among the available resources, is used to perform an operation and also decides which processing plan a part will follow. The second determines the order in which different parts are processed by the resources of the system, giving the sequence of operations that achieve the objectives and satisfy the constraints of the system.

Having said that, we define the scheduling of an *FMS* as the problem of determining plans for the parts to be produced, assigning operations to machines, and sequencing these operations. Obtaining performance that justifies the high-cost of an *FMS* depends on how we answer these questions. In other words, the beneficial effects of flexibility in an *FMS* (see [Wilhelm 85] comes at a price in terms of operation complexity. Without an effective means of scheduling and controlling *FMS*, it is clear that no reasonable economic returns can be expected from them [Harmonoski 91].

The problem has challenged both academic and industrial researchers during the last three decades and it is far from being given a definitive solution mainly because scheduling a *FMS* is amongst the hardest combinatorial problems. Not only is it NP-hard [Tzafestas 93], but even among members of the latter class it belongs to the worst

in practice. The real complexity of the problem can be illustrated by the history of three Job Shop Systems (*FMS* without routing flexibility) test examples published in 1963 [Muth 63] The optimal solution of the 20x5 problem was obtained in 1975 and the 10x10 problem was solved 26 years after the publication of the problem data [Carlier 89].

3. Scheduling objectives.

In most situations some schedules are regarded as being "better" than others. In other words schedulers may be compared with respect to the achievement of various objectives.

Three measures of schedule performance are common in the literature:

- Efficient utilisation of resources: completion time or makespan.
- Rapid response to demands: Mean completion time, flow time or waiting time.
- Close conformance to prescribed deadlines: Mean/maximum tardiness.

Makespan has received the most attention in the scheduling literature and it is the measure we will concentrate on in this thesis. However, there is an increasing recognition for the need to develop production scheduling methods based on economic consequences rather than on measures such as flowtime and makespan which are often of secondary importance [Bistline 98]. It is increasingly common to assess schedules against multiple criteria rather than just one criteria [Chan 97] [O'Grady 87] [Chryss 91]. From the point of view of the research developed here, this is formally a separate issue to the problem of scheduling as it relates to the comparison of schedules rather than the creation of schedules themselves. As we will see, however, the distinction is blurred in practice as some approaches (for example, those based on heuristics) commonly employ measures of schedule quality within the scheduling algorithm itself.

4. Scheduling *FMS* based on *PN* modelling and *AI* techniques.

The scheduling of an *FMS* needs two things: a) a representation paradigm that models the essence of the system and how it works and b) a means of reasoning about its behaviour, i.e., to develop methodologies that solve the scheduling problem based on the model.

In the following chapter, we will present a review of how this issue has been addressed in the literature. As we will see, two main groups of approaches will be identified. The first one, known as operation research methods *(OR)*, pays less attention to the modelling and concentrates on producing optimal schedules.

OR is characterised by a reduction of the problem into mathematical programming or network model formulations [Rogers 91], and subsequent solution by formal algorithmic methods. Although they can provide efficient solution techniques for small problems, the main disadvantage is that such models do not naturally reflect the complex structure of *FMS* and hence make it difficult to employ domain-specific knowledge to achieve afordable and effective schedulers [Noronha 91], although disjunctive graph models are better [Rogers 91]. [Bona 90] states that it is common practice in mathematical programming to decompose the whole set of constraints into subsets of easy (primary) constraints and complicating (secondary) ones. Primary constraints can be dealt with by building a model of the production facility, (typically sequence and technological constraints) which result from the relaxation of secondary constraints (for example, buffer policy and *WIP* constraints) which are not easily dealt with by *OR* algorithmic techniques. Consecuently, their results are difficult to translate to the industrial level.

The second well-known group is the on-line/real-time methods, which propose representation paradigms based on simulation tools that are more able to capture the complexity of *FMS*. The problem is that the scheduling techniques that they offer often resort to heuristic techniques that do not map well on to the operational complexity of *FMS*. [Hutchinson 91] suggested that the effects of routeing flexibility are more beneficial in high-cost optimal scheduling schemes than in cheap heuristics ones; and [Shaw 92] states that, because of the tighly-controlled environment in which they operate, *FMS*'s present schedulers with several operating problems that were not encountered in earlier conventional manufacturing systems.

As [Bensana 88] states scheduling is a field of investigations where advances in theory and solving techniques have not produced the expected results at the industrial level due to specific features of case-studies which prevent the general application of the approach. This criticism is often found in the literature, particularly when the *OR* general methodologies are confronted with real-time scheduling [Bistline 98] [Benjaafar 92] [Tamura 89]. As almost every review of production scheduling states [Harmonosky 91] [Rodammer 88] [Ammons 88] [Maccarthy 93] [Basnet 94] [Sukla 96], literature is sadly lacking the consideration of important *FMS* factors such as buffer policies, transportation systems, pallets, product stability constraints and slots in the tool magazines, etc. This is due to the complexity of modelling these constraints with traditional formulations of the problems. [Liu 97] states that very few models have tried to address all the aspects of *FMS* descriptions.

Consequently, there is a growing need to investigate new approaches to manufacturing scheduling that are capable of addressing the issues raised by these sophisticated systems. As [C.Y. Lee 97] points out, the new trend in scheduling theory is to extend results of classical algorithms to models that are more closely related to real problems. Even though many results may not be applicable immediately, these new models are at least motivated by industrial problems and have a greater potential for application. Finally as [Zhou 93] states, it is highly desirable for researchers, system analysts and production engineers to have a unified representation model for modelling, analysis, simulation, planning and the final control of manufacturing systems, i.e., a shared paradigm that covers all the stages of the life cycle. The Petri net *(PN)* formalism is a strong candidate to achieve these goals.

The potential of *PN* to solve the *FMS* scheduling problem is not only due to the fact that *PN* are a powerful modelling tool for the simulation of production processes and, consequently, they have been extensively employed in the on-line control of *FMS*. What it is of real interest, is that the *PN* formalism may also be viewed as a definition of a state-space structure. In other words, a modelling and simulation tool widely used in on-line/real-time scheduling is at the same time a possible representation paradigm for high-level scheduling strategies based in Artificial Intelligence *(AI)* methods. We believe this represents a promising scenario that has the potential to reduce the gap between advances in solving techniques and the industrial level by integrating *AI* search technology with simulation-based feasibility checking.

Although this integration has already begun, research in this area is relatively recent, immature and mostly unexplored [Onaga 91].

Work such as [Sakamoto 94] seems to corroborate this affirmation, and has highlighted the weak state of the art when search procedures are adopted within the *PN* community. To our knowledge, the first serious results were the work of [Lee 94] and since then, initial interesting results have been achieved. However, in our opinion, the main problem with all these approaches is that the complexity nature of the scheduling problem is not adequately dealt with. This resulted in poor results whose immediate consequence is that they obscure an important *PN* attibute: their ability to guide an *AI* based optimisation strategy in a search space that naturally models the behaviour of a system.

This thesis develops a technology based on *AI* search and *PN* that overcomes these problems and fully exploits the value of *PN* as a modeling tool for *FMS* scheduling.

5. Thesis structure.

The thesis begins by explaining how a *FMS* is modelled using a *PN*, and ends with a description of scheduling algorithms based on *PN* theory. Since each chapter deals with a different aspect of the problem, the background for each topic will be given there, as well as a full and detailed critical review of how the matter has been treated by previous work in the area.

Chapter 2 covers relevant literature and begins with a comprehensive review of the different families of *FMS* scheduling methodologies. We concentrate on reporting the findings and drawbacks of each of the disciplines. This will serve as the background for justifying the integration of *PN* and *AI* problem solving methods. The chapter will provide then the background on *PN* and *AI* search applied to manufacturing optimisation needed to understand their integration. The rest of the chapter is a comprehensive review of works that have integrated *PN* with *AI* based search methods. A description and review of each of these works will be given.

Chapter 3 starts by defining what is interesting about *FMS*, i.e., which are the main features that distinguish an *FMS* among other manufacturing organisations. The chapter provides the *FMS* formulation that we will use.

To demonstrate that *AI* techniques are relevant to *PN* models of *FMS* and vice versa, requires the definition of a scheduling architecture and each of its components. First, a new class of *PN,* the *cb-NET* is presented, which enhances the modelling techniques for manufacturing systems with features that are typically considered difficult to model, in particular bounded buffer residence times. Second, the integration of well-known heuristic search algorithms within the search space provided by the *PN* model is explained.

The chapter also presents an automated modelling methodology based on the parsing of textual specifications of *FMS* descriptions using a formal language that we have defined: *FmsML* or Flexible manufacturing system Modelling language. Finally, we discuss and reason about the complexity problem and how it may be allieviated by exploiting *PN´s* capabilities.

Chapter 4 studies *PN* structures and their dynamics to help the systematic search process within the *PN* reachability graph. The new heuristic function developed tries to give a theoretical lower bound for the minimum makespan between two production states of a *FMS*. We first review problems with previous approaches. A relevant heuristic function is presented and its properties explained. We will show that this heuristic function is admissible, and is very effective if the *FMS* is designed to exploit concurrency and avoid high machine idle time. Empirical evidence will show the superiority of the heuristic function.

Chapter 5 analyses the search space defined by a *PN* model of a *FMS* and how the knowledge acquired can be employed to reduce the search effort by avoiding the exploration of partial solutions that are known not to lead to a better schedule. The exploitation of *PN* reachability analysis in the terms mentioned, has received little attention in the works consulted, despite its potential in terms of search effort reduction. We first analyse procedures to identify partial solutions that have already been reached at the cost of maintaining a history record of the search process. Then we present a novel technique that avoids the generation of scheduling alternatives that do not yield to better schedules. Empirical analysis will show that the method ensures optimality and

considerably reduces the search effort, thus overcoming the drawbacks of previous methods.

Chapter 6 presents a search algorithm to be applied to medium and large *FMS* scheduling problems. We first discuss the literature that attempts to obtain affordable *PN* based heuristic search algorithms. We present a novel algorithm that overcomes the drawbacks observed in previous efforts, i.e, it allows the application of a *pure PN* based heuristic function, the search effort does not grown exponentially and it maintains the *best-first/backtracking* philosophy of *A**. Experiments will show interesting results, both in terms of optimality and comparison with other algorithms.

Chapter 7 integrates the results achieved in the previous chapters– heuristic search based on state space representation and structural analysis- with several successfully employed techniques – genetic algorithms optimisation ideas, and splitting up approaches – in a hybrid scheduling paradigm. The approach supposes an improvement and generalisation of the preliminary results based on *PN* truncation previously proposed. Experimental results show that its performance is close with the current works integrating *PN* and heuristic search. Compared with genetic algorithms approaches it presents advantages over global methods by identifying good structures in a progressive schedule building methodology. The preliminary scheduler demonstrates the benefits of a *PN* based implementation by allowing easy truncation analysis, immediate application of heuristic dispatching rules and state space search methodologies.

We conclude with a review of the contributions of the thesis and suggestions for further work.

Chapter 2. Literature review.

1 Introduction.

This chapter provides the context for the work reported in this thesis. It reviews:

a) Previous work on the scheduling of *FMS*.
b) The relevance of *PN* to the scheduling of *FMS*.
c) The relationship between *PNs* and *AI* problem solving methods.
d) *AI* approaches to *FMS* scheduling problems.
e) *PN* based scheduling and the use of *AI* with *PN*.

Given the wide ranging nature of the research and the rich and varied literature available on relevant topics, it is impossible to offer even a brief description of all the main methodologies that have been employed. We will concentrate instead on reporting the findings and drawbacks that each of the main disciplines present. This will serve as the background for justifying the integration of *PN* and *AI* problem solving methods.

An individual review and description of work that is directly related with such an integration will be provided in detail. The technological background for each topic will be given in each chapter, as well as a critical review of how the matter has been dealt with previously and the major contributions of each approach.

2 Flexible Manufacturing Systems and scheduling.

Although there is great diversity in the type of *FMS*, an unchanging feature is the existence of flexibility in job routing. Typically, the formulation of an *FMS* is an extension of a Job Shop System (*JSS*). In a standard *JSS* each part type has its own individual flow pattern, or route, through the machines that must be adhered to [Pinedo 95]. In a *JSS* there is little flexibility in the use of resources, and scheduling may often be formulated as a sequencing problem. The *JSS* definition may be extended so that several (different) resources can be used to perform an operation. These are usually referred as *JSS* with alternative.

The second type of flexibility in routing is usually encountered when the chronological order among operations is not fixed. These *FMS* are more difficult to formulate, since it requires a third decision variable to specify which route among the alternatives is to be followed. Only when this decision is made, are precedence constraints between tasks fixed. Whichever plan is chosen, the existence of different resource alternatives is also considered.

The *FMS* formulation terms that we will consider here consists of a set of *Jobs* to be produced. Several alternate *plans* for each *job* can be given, as a result of flexibility in technological constraints. We assume that each *plan* is described as a sequence of *tasks* to be achieved. Each *task* can be performed by different resources, each of these possibilities is called an *operation*. Each *operation* needs the use of one or more *resources*. The scheduling of an *FMS* can be described as the problem of determining plans for the parts to be produced, assigning operations to machines, and sequencing these operations.

3 Classification of scheduling methods.

A useful taxonomy separates scheduling approaches in two approximations which are dominated by two research communities which have produced a collection of substantially different methodologies:

a) Static scheduling, off-line scheduling or long to medium term scheduling. This refers to scheduling operations of available jobs for the entire scheduling period.

b) On line scheduling. This addresses issues of real-time control of *FMS* and concerns the detailed control of the processes as they occur.

It should be noted, however, that there is lack of clarity in these terms in relation to *real-time scheduling, rescheduling* and *dynamic scheduling*. Although real time scheduling is conceptually closer to *on-line* scheduling, a real time system can employ an off-line method. If off-line scheduling methods are utilised, the scheduling process becomes scheduling and rescheduling; whereas, according to the on-line scheduling approach, the scheduling decision is made when the state of the system changes, such as job completion or arrival of parts. There are advantages and disadvantages to both. Scheduling can be a very tedious task with off-line methods due to both the difficulty in

generating the schedule, and updating it frequently in a dynamic environment. On the other hand, scheduling decisions made by on-line methods may be sub-optimal due to a myopic system view.

A second classification scheme is based on the methodology followed. The following review attempts to classify the *FMS* operations methodologies into three main groups, which have a strong connection with the off-line / on-line division. Although the following discussion is a general classification of production scheduling, references are restricted, whenever possible, to works related to *FMS*. *PN* based work is excluded from this review. A complete review of *PN* methods is presented in section 3.

3.1 Operational research optimisation.

3.1.1 Operational research methods.

Mathematicians and operational researchers have typically considered off-line static scheduling and have emphasised optimisation. The problem is represented mathematically (mathematical programming, queuing, and network models) where a set of decision variables need to be fixed. The feasibility of the schedule is represented by a set of mathematical constraints. The problem is then solved optimality using a complete enumerative method such as *Branch & bound* [Spachis 79] [Brucker 94] [Demeulemeester 92] [Berrada 86] [Ram 90] [Christofides 87] [Jeng 93b]. The main discussion between different works is how fast the algorithms are and how deeply into the natural constraints of the *FMS* they go. Unfortunately, the goal of optimality is only relevant for small problems due to the combinatorics at the enumerative process.

A wish to address larger problems lead to the use of heuristic algorithms providing near-optimal solutions in a reasonable amount of time (see [Ahn 93] and [Hutchinson 91] for example). Basically, the following conceptual methodologies can be found:

a) Incomplete enumerative methods based either on incomplete enumerative algorithms (*truncated branch & bound* [Chu 92], *beam search* [Ow 88]) or heuristic algorithms [Adams 88].
b) Methods based on the relaxation of constraints in order to solve an affordable model analytically and then progress to a feasible solution (lagangrian relaxation) [Chen 98] [Chen 96] [Hwan 89] [Banaszak 90].

c) Stochastic exploration of the search space, and iterative optimisation, such as neighbourhood search [Chu 98], simulated annealing [Lo 91] [Mamalis 96] [Kruger 98], *tabu* search [Logendran 94] [Logendran 97] and genetic algorithms, Comparison between these methods are provided in [Tadei 95] [Glass 96] [C.Y. Lee 97].
d) Splitting the problem into separate yet linked sub-problems [Ashour 67] [Adams 88] [Yamamoto 77].

The last two methodologies represent the background for chapter 7 of this thesis and section 3.1.3 aims to briefly give the necessary background on genetic algorithms to manufacturing scheduling.

3.1.2 Drawbacks of *OR* methods when applied to *FMS* structures.

Many *OR* techniques have their origin in the solution of mathematical models of the *JSS* problem and need to be adapted for *FMS* structures. For example, the famous *shifting bottleneck* [Adams 88] is based on a decomposition approach where the sequencing problem concerning a single machine is solved in an iterative algorithm. The approach assumes fixed machine operation assignments and cannot be directly applied to *FMS* since machines are not assigned to operations a priori. Local neighbourhood methods such as *operation pairwise reordering* [Werner 95] [Chu 98] or splitting approaches based on workload assigned to machines [Chu 92] suffer from the same difficulty. This fact and the need for managing the complexity of the problem have lead many authors to divide the *FMS* operation problem into three identifiable stages [Kimemia 83]; *part selection, loading* and *sequencing*. Part selection determines the part types to be produced in the *FMS* out of the total production requirements. Loading is concerned with the assignation of resources to operations, (sometimes is referred to as pre-processing [Stecke 83]). Once the routes for each part type are fixed, the problem becomes a *JSS*, and traditional, effective and well-known methods can be applied. This hierarchical decomposition approach is considered in many papers (see [Liu 97] [Das 97] [Liu 89] [Sarin 87] [Y-D Kim 94] [Sawik 93]).

However, some authors have pointed out the drawbacks of hierarchical multi-level decomposition and proposed an integrated or single level approach [Zhang 93] [Khoshnevis 89] [Chryssolouris 85]. [Vasquez 91] states that the solutions obtained for

one sub-problem may inadvertely impose constraints on the next sequenced sub-problem. Hence, the approach may be incomplete as it potentially eliminates feasible regions. In contrast the concurrent (single level) approach can work over the entire feasible region by constantly considering the criticality of the resources. [Kim 99] states that a *loading then sequencing* approach results in many problems due to conflicting objectives, inability to communicate the dynamic characteristics of a shop or abnormal situations. When process planning (loading plus part selection) and scheduling (sequencing) are performed separately in time, it contradicts the aim of flexibility and adaptability of a *FMS* [Nasr 90]. An integration of these activities may result in a much more effective production control.

The bulk of published literature on *FMS* takes the operational research optimisation (OR) approach, and represents a first step in the analysis and control on the operations of production processes. However, they have severe limitations in terms of modelling and computational effort involved. The NP-complete nature of the problem and mathematical complexities force many assumptions. In order to formulate a model, simplifying assumptions about important practical issues such as buffer policies, set-ups, tool change-over and part stability constraints are made. These assumptions may not hold in practice. As a result, the solution of the *OR* models is usually unsatisfactory [Kusiak 89b]. This reduces the attractiveness of traditional analytical methods to produce practical problem solutions.

Although complexity reduction improvements can be made, some authors suggest that this cannot change the situation radically and is difficult to envisage an exact or even a good approximate algorithm suitable for a large class of scheduling problems. As a consequence, heuristics, such as priority dispatching rules, are often the only way to handle real-life scheduling problems [Shakhlevich 96]. We will review these in section 3.2.

3.1.3 Genetic algorithms.

Genetic algorithms (*GA*) belong to the class of iterative optimisation methods based on stochastic exploration of the search space. They represent an alternative to systematic deterministic search methods when these fail in search spaces which are in some respect too complex. Neighbourhood search, simulated annealing, and tabu-search also belong to this class, but differ from *GA* in that they are based on manipulating one feasible solution, whereas *GA* considers a population of feasible solutions.

Roughly speaking, a genetic algorithm aims to produce a near-optimal solution inspired by the principles of natural genetics and evolution. They operate through a simulated evolution process on a population of data structures, each of which represents a possible solution in the search space. Evolution occurs through a selection process and genetic recombination of selected high fitness strings to produce better solutions for the new generation.

Genetic algorithms were initially developed by John Holland [Holland 81] in the seventies, and since then, they have been extensively applied to search, optimisation and machine learning problems [Goldberg 89]. The application of *GA* to the scheduling of manufacturing processes has a more recent history. They have been applied to different production scenarios, from general resource-constrained project scheduling [Cheng 98], batch sequencing problems [Jordan 98], to Single machine scheduling [Lee 95], [Herrmann 95] flow shop systems [Reeves 95], [C-L. Cheng 96] [Murata 96] [Murata 96b], [Sikora 96], [I. Lee 97], [Aytug 98], *JSS* [Yamada 1992] [Bean 94], [Herrmann 95b], [Kopfer 97], [Biegel 90], [Dorndorf 95], [Della Croce 95], [Kobayashi 95], [Nakano 91], [Falkenauer 91], [Ombuky 98], [Ulusoy 97], [Maturana 97], [Lee 97] and machine centers [Sakawa 96] *FMS* [Bagchi 91], [Jain 97], [Uckun 93], [Holsapple 93], [Fujimoto 96], [Jawahar 98].

When *GA* are applied to manufacturing scheduling, three components must be specified.

a) Chromosome representation. The chromosome is a string of symbols that in general terms, specifies a single point of the problem space, (or individual). Each symbol in the chromosome is known as a *gene*. Both the position of the *gene* in the chromosome (*locus*) and the value of the symbol (*allele*) specifies the individual candidate solution.

b) Schedule builder. The schedule builder is in charge of obtaining a schedule from the information (usually decision variables) expressed by the genes of the chromosome. Depending on *how much* of the problem space represents the chromosome, the schedule builder will be a simulation or a more complex heuristic algorithm. We will deal with both the representation and the schedule builder in the following section. For a complete tutorial review of chromosome representations for production scheduling problems see [R. Chen 96]. The schedule builder also

determines the *fitness* of the individual, an indication of how good the schedule is with respect to the scheduling objectives.

c) *GA* operators. The general *GA* is composed of three operators: *selection, crossover and mutation. Selection* models a mating of individuals. When two (or more) individuals are selected (generally based on probability and the fitness function), *crossover* produces a new individual that inherits the characteristics of both parents. The combination of *selection and crossover* is also referred as *reproduction.* The last operation *mutation* is the random alteration of the value of a gene. The idea is to perform a random exploration of the problem space with the aim of assuring against premature loss of good scheduling decisions or strategies (which may lead to falling into local minima/maxima).

3.2 Heuristics and simulation.

Heuristic dispatching rules are different from heuristic algorithms proposed by OR in the sense that the scheduling is performed in a *dynamic* environment using a discrete event simulation of the system. Petri nets, a powerful tool for analysis and simulation, have been widely employed in the area of on-line simulation based scheduling. This work will be discussed in section 6.1. Usually the concepts *scheduler* and *scheduling* are substituted by the terms *decision module* and *decision making* respectively. Based on the role of simulation in the decision module, heuristic approaches can be divided into those based on dispatching rules and those using simulation.

3.2.1 Dispatching rules.

In this method, simulation is used as a tool to analyse the performance of a heuristic decision rule; which is later implemented in the real system. Usually, two heuristics levels are considered: one for deciding the next task to be processed and a second to choose the resources to perform the operation. A single policy may govern all the system [Iwata 80] [Ahn 93] [Chan 90] or dedicated heuristic rules for each resource may be considered, but this approach encounters co-ordination problems when multiple resources are needed to perform a single task (which is common in *FMS*). To increase the decision power several methodologies have been proposed. Multi-objective, multi-level heuristics are considered in [Norbis 96]. [Chan 97] proposes a fuzzy approach to

operation selection. Hierarchically organised dispatching algorithms are proposed in [Sabuncuoglu 88] [Sabuncuoglu 92] and dispatching rules in loading/sequencing decomposition are studied in [Shanker 85]. [Co 88] investigates the effect of buffer queue-length on five traditional dispatching in rules *FMS*. In [Donath 88] the real-time scheduling and routing of multiple products of a Flexible assembly system is performed by a heuristic approach.

These decision making rules are fast and easy to implement and the solutions obtained for traditional systems are usually acceptable. In many situations this is all that is needed, and so their use is justified. A major shortcoming of heuristic rules is that they embody a strong local decision making process [Khoshnevis 89]. In addition, the investigation of priority rules may give improved schedules for a given *FMS* but it does not lead to effective algorithms of wide applicability [Doulgeri 87] since the general consensus is that no single rule is the best under all possible conditions. On the other hand, its computational cost is fixed (the heuristics are hard-wired) preventing the decision module using extra computational time and improving the quality of decisions. As said in the introductory chapter, the complexity of *FMS* systems requires effective schedulers to fully exploit their potential.

With the rapid increase in the speed of computing and the growing need for efficiency in scheduling, it becomes increasingly important to explore ways of obtaining better schedules at some extra computational cost, short of going all the way towards the usually futile attempt of finding a guaranteed optimal schedule [Adams 88]. A first approach to this is the use of performance evaluation based on simulation. As [Kazerooni 97] states, a typical *FMS* has a high investment cost which justifies the use of computer simulation support.

3.2.2 Simulation-based systems.

Whenever a scheduling decision arises in a system, a simulation model of the *real-system* is built and initialised to the exact current state of the factory. Then, parallel simulations are carried on to evaluate the performance of a set of plausible dispatching rules over a short planning horizon. The rule or combination of rules that performs best in the planning horizon is then applied to the physical system. This approach is

sometimes referred as a *multi-pass heuristic* and is implemented in [Yung 88] and [Wu 89]. Other approaches based on simulation of candidate dispatch rules can be found in [M.H. Kim 94] [Ishii 91] [Ishii 96]. Fuzzy decision making via a combination of rules in a multi-criteria decision making is studied in [Kazerooni 89].

These works are an improvement on fixed dispatch rules. Normally, depending on the system deadline for decision response, computational effort can be controlled by increasing the horizon or varying the number of dispatch rules that are simulated. However, [Shuckla 96] states that the main problem with this approach is the need for multiple simulation replication due to the stochastic (and complex) nature of *FMS* if one seeks to obtain relatively accurate results based on statistical analysis. Unfortunately, multiple replications consume computational time and possible solutions, such as parallel computation, quickly increase the cost in terms of hardware.

Nevertheless, the interest of these works is that they provide the opportunity for reasoning in a *what-if* scenario that is based on a simulation of the real system. [Chryssolouris 88] suggests that a modular system can be built which, on the one hand, will utilize the rigourous analysis of scheduling and decision-making theory while, on the other hand, will allow the utilization of some AI techniques such as rule-based systems.

3.3 Artificial intelligence methods.

AI based methods are particularly well suited to solving *FMS* scheduling problems as *AI* has addressed similar problems involving large search spaces where human expertise can find reasonable solutions relatively quickly. [Steffen 86] has identified two main research directions based on *AI* techniques for *FMS* scheduling:

3.3.1 Expert systems.

Expert systems applied to *FMS* normally attempt to emulate expert human knowledge and are typically applied to real-time scenarios [Bistline 90]. Usually an expert system architecture is formed by:

- A structured representation of the problem [Sauve 87] [Bu 88]

- Knowledge sources about how to solve the problem such as: dispatch rules [De 88], simulation [Bruno 86] [Yung 88], multi-criteria decision making [Chryss 91], OR optimisation heuristic search [Bona 90] [Kusiak 89b].
- A control strategy that may be based on co-operation of multiple knowledge sources [Bensana 98] or agents (blackboard systems) [O'Grady 88].
- A knowledge adquisition module from past decisions [Shen 88], [Maley 88]. Some approaches employ neural networks [Li 97], decision trees [Park 89] [Shaw 92], and genetic algorithms [Aytug 98] or a combination of these [Lee 97] [Jones 95].

An expert system represents a paradigm of knowledge organisation and control based on scheduling techniques (knowledge sources) rather than a scheduling methodology per se. Many of these architectures *ISIS* [Fox 84], *OPAL* [Bensana 88] and *KBSS* [Kusiak 89] employ *AI* problem solving and planning as such knowledge sources (a collection of this work is presented in section 5).

3.3.2 Planning.

Planning usually involves the application of goal directed search techniques based on a state representation of the problem.

One can distinguish four main steps towards a successful application of *AI* based search methods:

- Provide a representation of the problem in terms of state, operations and constraints that describes the *FMS* environment.
- Understanding the problem domain: It is easy to find a first solution? What is the quality of this first solution?, how much can this solution be improved? At what cost? It is possible to identify futile decisions? What kind of knowledge can be used that is problem specific?
- Understanding the size of the problem in terms of search effort. Usually, the problem will not admit an optimum algorithm. The existing techniques consider 1) splitting approaches, 2) limited work based on job arrivals or 3) the whole problem but devise a solution-space pruning strategy if this becomes unwieldy.
- Develop a search paradigm capable of exploiting this analysis to effectively control the combinatorics of the underlying search space.

4 Integrating Petri Nets and *AI*.

4.1 Representation and reasoning.

Whatever approach is considered, but especially from the point of view of *AI* problem solving, the scheduling of an *FMS* requires two fundamental things to be done: representing the system and reasoning about its behaviour.

In the following sections, we will justify why *PN* are a good representation formalism and how AI aproaches can be used to reason with this representation.

4.2 *PN* as a representation tool for *FMS*.

A Petri Net (*PN*) [Murata 89] is a mathematical formalism and graph tool that provides a uniform environment for the modelling, formal analysis and design of discrete event systems. Petri Nets have been widely used in industrial applications [Zurawsky 94]. The performance of production systems, involving simple production, job shops, robotic assembly cells and flexible manufacturing systems, have been extensively studied by the *PN* community [Proth 96] [Silva 89] [Zhou 93] [Zhou 92] [Zhou 93] [Dicesare 91]. As a graphical tool, a *PN* works like a flow chart and provides a visualisation of a dynamic system. As a mathematical tool, a *PN* model can be described by a set of linear algebraic equations, which allow the possibility of formal checking for properties related to the behaviour of the underlying system.

A Petri net can be defined as a bipartite directed graph formed by three types of entities: places, transitions and directed arcs connecting places to transitions and transitions to places. The network structure represents the static part of the system. In order to study the dynamic behaviour of the net, in terms of states and changes, each place may potentially hold zero or more tokens, which are usually represented by solid dots. A distribution of tokens in the places of the *PN* is called a *marking* of the *PN*. The initial state of the system is called the *Initial Marking* or M_0..

Starting from the initial state M_0, two simple rules associated to transitions, the *enabling* and *firing* rules, are used to govern the flow of tokens in the net, which simulates the dynamic behaviour of the modelled system.

In order to study the evolution of dynamic systems the concept of time needs to be included in ordinary Petri nets. Time will typically model the duration of operations. Apart from simulation based schedulers that may model stochastic behaviour, deterministic behaviour is typical of off-line and some on-line schedulers.

Deterministic timed *PN* are sufficient for the modelling of traditional production processes and simple control of real systems. In some contexts that contain complex commands and constraints the ordinary *PN* needs to be extended. Extensions such as *Coloured PN* [Cossins 92] [Feldmann 98], *Object Oriented PN* [Adamou 93] [L-C. Wang 96] [Wang 98], *High Level PN* [Chang 97] *Hierarchical Time-Extended PN* [Ramaswamy 97] *Priority Nets* [Raju 93], and *Dedicated Petri Nets* [Zimmermann 99] amongst others, have been employed in the modelling and analysis of *FMS*.

*PN*s are a useful tool for modelling *FMS* since they have been widely recognised as appropriate tools for describing phenomena such as concurrency, synchronisation, mutual exclusion and conflict, which are typical features of distributed environments such as *FMS*. *PN*s allow distributed system states to be modelled naturally and through their approach to state change can capture both the static and the dynamic characteristics of real systems. Hence, *PN*s are:

a) Capable of modelling the characteristics and natural constraints of *FMS* systematically in a single coherent formulation, this makes unnecessary the discussion about loading/sequencing problem decomposition previously mentioned. Further a *PN* model can explicitly and easily characterise features such as multiple lot sizes, finite buffer sizes and part stability constraints encountered in a practical manufacturing environment, (recall that mathematical programming techniques have formulation difficulties with these features, in fact, the constraints on the storage of *WIP* are normally not included in these formulations, even though they are critical for practical *FMS* scheduling problems [Liu 97]).

b) Can provide information (by well understood mathematical analyses) that can be used to guide the scheduling process

c) Can support a modular and hierarchical construction approach [Narahari 85] [Zhou 92] [Jeng 93] to automatically synthesise *PN*s from, for example, *FMS* specification languages or formal definitions [Camurri 93] [Xue 98].

d) A single family of *PN* based tools can serve at different levels, from design to implementation. For example, *PN* are employed for the design and verification of Manufacturing System Control Software in [Zurawski 94b] [Heiner 99] [Kochikar 93]., for supervisory purposes [Caramihai 98], and decision/monitoring support for production engineers in [Murata 86] [Sahraoui 87].

e) The feasibility of the schedule obtained is guaranteed, (provided the mapping between the *PN* and the modelled system is valid) and the sequence of transitions is directly applicable by a control module that shares the *PN* representation [Chang 97] [Uzam 98].

These features justify the use of *PN* as a modelling paradigm in the reasoning process involved in the solution of the *FMS* scheduling problem.

4.3 *PN* as a state-space definition for *AI* search techniques.

A Petri net is a definition of a state-space structure in terms of state and operators that are partial functions that map states into states which together with the definition of an initial and final state, transforms the scheduling problem into a *problem state* [Newel 72] to which traditional *AI* search algorithms can be applied. The definition of *search space* is equivalent to the *PN* concept of *reachability* tree [Murata 89], which is the enumeration of all possible markings reachable from a given state of the system. Starting with an initial state representation, M_0, one can track all the possible behaviours of the system by firing all possible transitions enabled in all possible markings reachable from the initial marking M_0. The generation of the reachability tree takes exponential time for the general case [Murata 89]. An interesting empirical study of the complexity metrics of *PN* with the reachability tree as the target

problem can be found in [Soo 92]. An overview of methods to manage coverability (reachability) graph constructions can be found in [Coves 98]

On the other hand, given a *PN* the reachability problem is defined as finding a sequence of transitions that reach a final marking from an initial marking. It is not surprising that the connection between *PN* and *AI* problem solving has already been studied. [Zhang 90] [Zhang 92], [Yu, 97] and [Yim 94] transform a propositional planning problem into a class of timed *PN* and apply the *A** algorithm within the *PN* reachability graph. The reverse also applies, i.e. the *PN* defines a problem-space model as a propositional planning problem [Fikes 71]. When time information is included in the *PN* model, the reachability problem can be extended to a search with optimisation, thus establishing a parallel with the *FMS* scheduling problem.

5. *AI* problem solving in manufacturing optimisation.

If a *PN* model is a description of an *FMS*, the scheduling problem may be translated into a search problem within the state space defined by the *PN* reachability tree. A range of algorithms exists for solving such problems. From blind enumerative search methods such as *depth-first* and *breadth-first* search, to informed search methods such as *A** or *Branch & Bound* (*B&B)* approaches (see [Pearl 84]).

Rather than concentrate on problem solving techniques, many approaches that can be classified as *AI* approaches are relevant from the point of view of the representation paradigm adopted. Typically, this work represents a collection of different traditional *AI* based problem representations of *FMS*'s. For example, [Chakravarty 97] proposes an *Object Oriented* model of *FMS* and employs *Branch & Bound* to solve the sequencing problem once the loading problem has been solved, however, this work is mainly focused on the analysis of the *FMS* domain. A Knowledge Based approach using frames is studied in [Kusiak 89] where the schedule is obtained with a multi-level heuristic algorithm. Although not specifically for *FMS*, the work proposed in [Zamani 97] implements a version of the *LRTA** (learning real time A*) of [Korf 90] for general resource constrained problems (which can model simple *FMS* scenarios). The interest of this work is that it identifies the need for defining a system state that includes time, and state transition operators that models resource and

precedence constraints. The fact that this work fails to provide a formal definition of this representation paradigm, reinforces the idea of employing a *PN* as a state-transition representation which is well defined. [Fox 82] [Fox 84] represents the problem as a constraint-satisfaction problem and also employs beam-search for the scheduling of resources. [Cheng 96] also presents a methodology based on constraint satisfaction problem solving. Although the problem domain is not an *FMS*, the interest of this work is that they demonstrate that CSP scheduling techniques can provide a basis for developing high-performance approximate solution procedures and the representational assumptions underlying CSP models allow these procedures to naturally accommodate the constraints of most real-world applications. [De 90] represents a *FMS* environment using frames and production rules which are exploited for progressing search and which implement heuristic knowledge to solve conflicts while filtered beam search is employed to prune the search space.

A second class of approach adapts *AI* search algorithms to manufacturing scheduling. For example [Lee 98] adapts *A** to a modelling paradigm based on a concept called schedule elements. A schedule element acts like a decision variable that decides how a stage of a part is processed. The main criticism is that this is not a model of the system but a representation format of feasible schedules.

It is interesting to mention that work in this line usually requires feasibility checking, i.e, it needs to determine whether a planning/schedule decision is feasible according to the constraints. This problem is not observed if a representation based on *PN* is adopted, since typical constraints of the domain are modelled ruling the behaviour of the model.

As an attempt to solve the unfeasibility problem, some works start by proposing modelling paradigms that get close to *PN* fundamentals. For example, [Liu 89] devised a representation paradigm called a *work-graph* and applies *A** search. In this work, the sequence and precedence constraints are modelled, but not mutual exclusion for the use of resources. A similar approach is presented in [Liu 92]. This work introduces the idea of a dynamic state of the *work-graph* using tokens to represent the sequence status for each job, which starts to resemble to a *PN*.

From the point of view of search algorithms and schedule generation methodologies interesting work is reported in [Shaw 88] [Shaw 88b] [Shaw 89]. A non-

linear planning strategy that employs a knowledge-based representation of the *FMS* first decompose the overall problem, applies *A** to each sub-problem, and finally applies a plan revision procedure. *A** is used to obtain the best route for a single job, based on a representation of the entities using frames. Interestingly, the authors highlight the progression to a schedule by means of state-space transitions, which differs from conventional scheduling methods. Also, this scheduling methodology is interesting due to its integration of planning methodologies and splitting up approaches. We will return to this work in chapter 7.

The major contribution of this work by the time of publication was the demonstration that *AI* problem solving techniques based in state space representation are a powerful tool for *FMS* scheduling problem. More importantly, it confirmed the potential of a representation paradigm that is both a state space representation and a powerful and well-known tool for the analysis of discrete event systems. For example: *A** is used to determine the best (shortest) route for a Job. However if a *PN* is employed, this problem is simply reduced to a search for the shortest path in the graph structure represented by the *PN*. In addition, to identify conflicts among operations, they define the concept of critical sections. These are sequences of operations that require a shared resource. To define a mutual exclusion among critical sections (and avoid pre-emption), they suggest the use of semaphores. In other words, they provide a solution that includes the concept of concurrence, sequence and mutual exclusion. But a *PN* results in a much simpler and intuitive representation to model these concepts. A very similar *AI* non-linear planning application to scheduling *FMS* is described in [De 88].

An alternate approach to fight the complexity problem is the use of incomplete enumerative methods, which are based on the partial exploration of the search space by, for example, the search in parallel of different alternatives as a result of the heuristic selection of the next action to apply. *Beam-Search* remains the most used methods both for on-line and off-line [Chang 89] [Holsapple 93] [Ow 88] [Chryss 91] [Karabuck 93]. A description of *Branch & Bound, A** and *Beam Search* will be given in chapters 3 and 6 since they are the basis of the scheduling algorithms developed as part of this research.

The translation of state-space generation algorithms to on-line scenarios has resulted in the application of look-ahead simulation and *game search* influenced algorithms from *AI* to real-time scheduling [Rajan 90] [Chryss 94]. A more recent

example is [Dario 96] that presents a methodology called *Chess*. Basically the approach tries to minimise the contentions produced by a single strategy. Whenever a conflict arises, a look-ahead simulation of heuristics expands the search tree to a limit. The next decision is made accordingly with the most favourable simulation.

The interest of work in this line is that it seems to confirm what was stated in the work of [Doulgeri 87] where a simulation-based method that uses a simple look-ahead to evaluate conflicting activities was proposed. The algorithm is rather simple since it employs fixed dispatching rules. Although no representation paradigm for the problem is explained, they conclude that the decision-making process could be improved through a highly accurate, short-term look-ahead, as opposed to a less-accurate long-term look-ahead, where the noise induced by look-ahead inaccuracy could overshadow the marginal differences between decision paths.

These results are of interest to us in terms of dealing with the complexity of the scheduling problem as we will see in chapter 6. Also, a discussion about the translation of the search algorithm developed in this work to *on-line* scenarios will be presented in chapter 8.

6 A review of the use of *PN* in scheduling manufacturing optimisation.

The application of *PN* in the modelling, analysis and simulation of production processes has been widely studied. Intelligent planning and scheduling over *PN* models, which is the topic of this research, will be reviewed in detail in this section. The literature about Petri Nets as a scheduling tool for *FMS* can be divided into two groups, defined by their different scheduling methodologies and the time horizon considered.

6.1 On-line scheduling based on *PN* simulation.

As *PN*'s are a powerful modelling tool for the simulation of *FMS*, it is not surprising that the major application of *PN* in scheduling and control of production processes is the on-line control of *FMS* using dispatching rules, simple try-and-test simulation schemes or more elaborated optimisation approaches. Although the majority

of this work does not assume an integration of *PN* with *AI* technologies, a review of recent work that employs *PN* theory for the scheduling of *FMS* is needed for the understanding of the context of the work reported here.

The integration of *PN* modelling and conflict resolution based on dispatching rules is the common theme. [Chang 97] [Malo-Tamayo 98] [Colombo 97] [Lin 97] [L.C. Wang 96] [Hu 95] [Huang 92] [Yim 93] [Yim 94] [Chincholkar 96] [Kuo 98] [Schmidt 92] all describe dispatch rule based approaches to scheduling. Sometimes, new *PN* extensions are developed to incorporate such control policies into the *PN* formulation [Raju 93].

The simulation based scheduling methods described in section 3.2 have been implemented over *PN* models of *FMS*. For example, selection of a combination of different dispatching rules based on simulation is studied in [Lin 95]. Dedicated machine dispatching rules are proposed in [Liu 93] and *PN* simulation to evaluate the effect on performance of different decision alternatives is studied in [Ravichandran 86]. Heuristic algorithms have also been applied to *PN* models of *FMS* systems, for example the heuristic approach of [Donath 88] is employed in [Chetty 96] and the *Shifting Bottleneck* procedure [Adams 88] is adapted in [Chang 97].

All these *PN* approaches can be described as *on-line heuristic-simulation-methods*. Their main contribution is the use of *PN* modelling power to reduce the gap between *OR-based* scheduling and practical applications, but they inherit the drawbacks associated with simulation and heuristics dispatching algorithms. For example, in a more complex simulation approach, [Peng 98], a parallel simulation of dispatching rules in an ordinal optimisation approach is employed, but the authors argue that further research is needed to find a more sophisticated scheduling algorithm.

The integration of *PN* and *AI* may have its starting point in work that employs *PN*s as a representation paradigm for knowledge based architectures to produce schedules. For example, [Martinez 87] suggests the application of rule-based knowledge as a control strategy to a system modelled with a *PN*. Other work exploits the simulation capabilities of a *PN* model to implement expert systems based on hierarchical rules that observe performance parameters from the *PN* simulation [Tamura 89] [Hatono 91]. Higher control architectures are proposed in [Yan 97] [Yan 98], where high-level *PN*s are proposed for the control of *FMS* based on an expert system employing dynamic re-scheduling rules and a blackboard organisation.

However, work in this line focuses on descriptions of scheduling architectures and the integration of techniques within a coherent framework but pays little attention to algorithmic techniques based on *PN*.

For example, in [Camurri 93], a high level description of *FMS* as a knowledge base is transformed into a coloured *PN*. The work proposes two scheduling schemes: a monte-carlo simulation based scheduling scheme and simple/complex dispatching rules [Camurri 91]. As described in 3.2, these are not powerful enough to couple with the complexity of *FMS*. The same can be said of [Tsukiyama 92] where a scheduling tool based on human-computer cooperative problem solving is proposed. The system proposes *PN* as a simulation tool over which state-dependent decision rules and dispatching rules are in charge of build an initial schedule. The schedule obtained can be modified by human editing.

In [Martinez 89] the basis of a hierarchical control architecture is proposed for the real-time control of production systems. They represent the problem by means of *PN* models. As part of the decision module, they suggest the application of look-ahead search (heuristic Beam Search), however no algorithm description or results were given.

It seems clear that these papers suggest the benefits of use *PN* representation for scheduling paradigms which are related to *AI* methods, but lack the definition of *AI* based search algorithms that use and exploit this *PN* representation. Such an affirmation justifies the aim of this research to provide effective *PN* based scheduling algorithms as decision modules which can be integrated within these control architectures.

6.2 *PN* based Heuristic search.

The discussion above suggests that the combination of *PN* modelling and *AI* problem-solving methods may be a promising way to solve Flexible Manufacturing Systems (*FMS*) scheduling problems. Although this integration has already begun, this research direction is relatively recent and immature.

The following, is a review of the existing literature that transforms a *PN* model of a *FMS* into a state-space problem, and applies *AI* space search algorithms. It includes other approaches of interest that are relevant from the point of view of modelling or integration of technologies. *Table 1* gives a classification of each work in terms of the manufacturing system characteristics considered and the scheduling strategy applied.

Unless specified, approaches consider minimisation of the makespan as the scheduling objective.

Paper	**System considered**	**Buffer constraints**	**Search Method**	**Heuristic Information**
[Hillion 87] [Hillion 89] [Hillion 98]	Cyclic *JSS*	Infinite	Heuristic algorithm	
[Proth 96b] [Proth 98]	Pseudo-Cyclic *FMS*		Decomposition, mathematical programming and heuristic algorithms.	
[Damasceno 98]	*JSS*	Limited buffer.	Splitting up and mathematical programming	
[Liu 93]	*FMS*		Dispatching rules.	
[Tanida 92]	Cyclic *JSS*		Approximation algorithm based on priority list dispatching	
[Gusikhin 93]	*JSS* with alternative operations	Infinite	Heuristic guided backtracking.	
[Sakamoto 94]	*JSS*		Depth first Search within the *PN* reachability graph.	
[Dutilleul 98]	Hoist scheduling problem	Null-buffer	Branch & Bound	Heuristic based on "process the maximum number of products at the same time".
[Lloyd 95]	*JSS* with assembly/ disassembly.	Infinite	Branch & Bound with search reduction	Longest machine working time
[Shen 92] [Chen 93] [Cheng 94]	Parallel processing with synchronisation		Truncation of the *PN* model. Branch & Bound with search reduction.	Lower bound based on longest remaining Job
[Abdallah 98]	*JSS*	Null-buffer	*B&B* guided by a heuristic dispatching rule.	*PN* based structural information, heuristics

Paper	System considered	Buffer constraints	Search Method	Heuristic Information
[Shih 91]	*JSS* with alternate operations.	Limited buffer	Look ahead approach based in beam search	Best first.
[Lee 92] [Lee 94]	*FMS* with alternate operations	Infinite	*A**	Non admissible heuristic function based on the number of operations remaining.
[Sutdhiraksa 96]	Robotic assembly operations		Reachability graph analysis based on heuristic methods	Lee 94 heuristic, plus consideration of robot idle time
[Lee 94b]	*FMS* with AGV transportation	Limited buffer.	*A**	[Lee 94] heuristic but tuned using problem information .
[Sun 94]	AGV control in *FMS*		Staged search *A**	[Lee 94] heuristic.
[Yim 96]	*FMS* with alternate operations	Infinite	*A**	Several tuned heuristic function based on problem information
[Chen 95]	*FMS* with alternate routing	Infinite	Irrevocable depth-first search technique	Heuristic estimation based on the *PN* state equation.
[Jeng 96] [Jeng 98] [Jeng 99]	*FMS* with alternate routing	Infinite	*A**	Heuristic estimation based on the *PN* state equation.
[Jeng 98 b]	*FMS* with alternate routing	Limited buffer.	*A** with limited backtracking.	Heuristic estimation based on the *PN* structure plus longest remaining job
[Xiong 98]	*JSS*	Infinite	Hybrid search based in *A** and depth first.	*A** employs longest remaining job.
[Xiong 97]	*JSS*	Limited buffer/ Null-buffer	Hybrid search based in *A** and depth first.	Longest machine working time
[Chiu 97]	*FMS*	Infinite	Embedded Genetic Algorithm	
[Inaba 98]	Assembly Systems	Infinite	Staged search *A**. Search reductions.	Completion time estimation based on the mean operation cost

Paper	System considered	Buffer constraints	Search Method	Heuristic Information
[Boutet 98]	*FMS* in Preliminary System Design		Reachability graph analysis.	

Table 1: Summary of PN based works.

[Gusikhin 93] states that heuristic based simulation remains the more widely used computer-based scheduling technique for *PN*. In this work *FMS* with alternative routing is considered. An infinite buffer policy is assumed between any two operations. The *PN* model is a composition of models of both *FMS* facilities and technological processes. Time is deterministic and assigned to transitions. The main contribution of this paper is the clear view of the combination of *PN* traditional simulation techniques and *AI* systematic search within the reachability graph. The algorithm proposed employs simple chronological backtracking guided by heuristic knowledge when deciding the next operation to apply. The main problem is that no heuristic function or procedure is given and no case study is presented.

[Sakamoto 1994] considers the scheduling of a *JSS* (no alternate routing). *PN*'s are said to be extended to consider deterministic time (although it is difficult to see that is anything other than standard timed-*PN*). The scheduling approach lies in reachability tree generation. Being aware of the NP-hard nature of the problem, they suggest a depth-first approach and a control mechanism for duplicate markings. Although the paper does not offer anything new, its main contribution seems to be the application field: *Chemical Plants*. However, the paper has an elegant description of how the simulation of time is conducted in a *PN* model. This type of analysis is missing in much other work.

[Hillion 87, 89, 98] consider the problem of scheduling a *JSS* with *n* different parts and *m* machines. The number of parts of each type are manufactured according to a given production mix and follow repetitive demands in steady state. The *PN* model follows a hierarchical approach: A first sub-net, called the *Processing Circuit* models the precedence constraints. This circuit is said to be decision free (no possible alternate routing is considered). A second circuit: *the Command Circuit* models resources and the

machine utilisation conflicts among operations. This organisation is commonly found in analytical approaches using disjunctive network graphs.

The scheduling algorithm is based on effective algorithms obtained for *Flow-shop* systems scheduling approaches. It is based on the idea of determining the critical circuit containing the bottleneck machine, which determines the productivity of the system. The algorithm employed is heuristic, identifying the bottleneck machine and trying to guarantee full utilisation of this machine, whilst minimising WIP.

Although interesting, these papers are a clear example of previous results that are non-extendible to *FMS*. Notice that in a *JSS*, the assignment of operations to machines is fixed, and hence, given an input rate, it is possible to identify the bottleneck machine. In application to *FMS* this has two problems: the random arrival of production demands and, more criticaly, the unfixed assignment of operations to machines (due to flexibility).

[Proth 96, 98] consider *JSS* with alternative operations and assembly processing. The scheduling approach is based on continuous part processing, in the sense that it is assumed that the number of parts of each type to be produced is large. To solve this problem, a traditional decomposition approach based on *loading then sequencing* is proposed. The loading or *short term planning* is solved as a mathematical program. The scheduling is obtained by first reaching a steady state (a minimum initial marking) that reduces work in process and allows the system to continue by firing the transitions as soon as they are enabled (Earliest Operation Mode).

The main problem with this approach is the assumption of constant part rates. It would be interesting to study its response to dynamic job arrivals, which are usually considered in *FMS* environments.

[Damasceno 98] also address the problem of *JSS* (no alternate routes) with limited buffers by exporting traditional OR research methods to *PN* models. Due to NP-hardness, they propose a scheduling method based on Job arrival decomposition. Jobs are scheduled one after another using dynamic programming taking into account already scheduled operations. No *PN* based search is described. In addition the paper criticises previous heuristic search approaches such as best first A^* in [Jeng 98] claiming that it seems difficult to obtain reasonable bounds for partial solutions and it is improbable that

the *A** search method can solve large scheduling problems. Such a conclusion seems to confirm the immature state of the integration of *PN* and *AI*.

[Liu 93] also suggest the application of *AI* to the control of *FMS* modelled by *PN*. The paper analyses *buffer overflow and deadlock* problems and suggests the possibility of scheduling by studying all the possible combinations of firing policies for each machine. The use of the reachability tree is proposed as a means of detecting deadlock situations. However, since they are constrained to a small case study, there is no mention of the difficulty of extending this method to larger *FMS* formulation.

[Tanida 92] consider cycled *JSS* with no alternate operations. Scheduling is performed in terms of the construction of priority lists of transitions, which are ordered by *measures*. The paper is particularly opaque, specifically the algorithms lack an intuitive interpretation. As far as we can determine, the approach followed is simply the use of dispatching rules to determine the next transition to apply.

[Lloyd 95] considered a *FMS* with product assembly/disassembly and infinite buffer capacity. However alternative routing seems not to be considered. A typical *PN* modelling procedure is employed where time is assigned to places. A branch & bound method is proposed which includes a method for search reduction by comparing previously explored markings. As a branch & bound approach, it is only appropriate for small problems.

[Dutilleul 98] also employed a branch and bound search algorithm for the *Hoist* scheduling problem. The major characteristic of such scheduling problem is that they include operations where the processing times are included between a minimum and maximum value. This particular property of operations may be found in several kinds of chemical industry (for example, electroplating). The main interest of this paper is that the authors indicate that the usual timed *PN* are not able to model, amongst other things, a no-wait machine. A *PN* extension known as P-Time *PN* [Khansa 96] is proposed to model these constraints. The advantage is that the constraints for the scheduling algorithm are automatically extracted from the model. The heuristic function employed by the branch and bound maximises the production rate by processing the maximum number of products simultaneously.

[Shen 92] propose a splitting up approach based on *PN* structures for the affordable scheduling of parallel processing in industry. The *PN* is truncated into sub-nets that are solved using Branch & Bound procedures. A lower bound based on the longest job is used to prune non-promising markings. Sub-schedules are joined together for the final solution.

[Chen 93, 94] is an extension of the previous paper. The problem domain is not *FMS*, but parallel processing with synchronisation such as CPU task scheduling and industrial manipulators. The branch & bound approach uses no heuristic information to select the next node to expand, although a comparison test is proposed to find previous reached markings. The branch and bound algorithm, when solving a sub-net, employs information about synchronisation with previously generated schedules. Both this work and the previous one are interesting since they show how *PN* structural and property analysis are useful in truncating larger problems and guiding search methodologies.

[Abdallah 98] modelled a *JSS* where the buffer policy is a NULL-buffer, in the sense that parts can wait in the machines but the machine is not released until the part has been transferred to the next stage. Although no alternative operations are allowed, an operation may required multiple resources. Since null-buffering is considered deadlocks must be avoided. A *PN* model of such systems possesses structural properties that can be exploited to generate deadlock avoidance heuristics. A branch and bound procedure that makes use of this information is used. The branching scheme, however, is depth-first guided by a two-level dispatching rule. They report successful computational results with large problems. However, the problem domain is strongly constrained. There is no flexibility and the null-buffer policy leads to many dead markings, which are pruned. This suggests that the dispatching rule is likely to find a very good first solution. A pruning method based on comparison with markings that actually exist in the candidate path for the optimum solution is employed.

[Shih 91] considered the on-line scheduling of cycled *JSS* with alternative machines and limited buffer capacity. The target optimisation parameter is earliness/tardiness. The system is modelled with a deterministic *PN* with time assigned to transitions. The main result of the paper is that it demonstrates the benefits of employing *AI* techniques (*Beam*

Search) as a short term look-ahead instead of general dispatching rules such as FIFO, priorities based on *dynamic slack* and the number of remaining operations. This confirms the possibilities of *PN* reachability analysis based on simulation. However, no heuristic information extracted from the *PN* model seems to be employed to guide the search. Chapter 6 will analyse the implication of this in terms of the quality of the solution obtained.

[Chen 95] stated that *FMS* characteristics such as resources, sharing concurrency, routing flexibility and mutual exclusion, lot sizes are difficult to describe using conventional tools such as mathematical methods equations and proposes a *PN* modelling of the system. Infinite buffer policies are considered by assuming that buffering is a shared resource that allows enough space for all the unfinished parts. This obviously avoids deadlock situations. Alternate routes for each job are considered. Two types of systems are considered, *symmetric and asymmetric* depending on wether alternate plans have or do not have the same number of operations. The scheduling methodology is based in an irreversible strategy (notice that no deadlocks can occur) guided by a heuristic estimation of the goodness of the marking. In other words, all the transitions for the current markings are fired, obtaining a set of successors out of which, a single marking is considered, discarding the rest. In this sense the procedure can be considered as a *Beam Search* with *Beam* width of one, or simply a *Hill Climbing* strategy. The selection is based on a heuristic estimation of the minimum total cost of operations needed to complete the jobs. In fact they consider the *PN* as a mathematical formulation and try to solve a relaxation of the original minimisation problem based on the *PN* state equation. The interest of this work lies in the use of the *PN* definition to obtain heuristics that can guide the search process. However, the main problem with this approach is that it implements an irrevocable strategy and that the heuristic function is an estimation of the total processing cost of remaining operations, and not the makespan, which is the scheduling objective.

[Lee 92, 94] presents an excellent synthesis procedure for *FMS* with alternate operations, and multiple resource utilisation. To avoid deadlock situations, intermediate storage is allowed between any two operations. Models for limited buffer situations are described, although not applied in the scheduling experiments. They propose a heuristic search algorithm *L1* which is an adaptation of the well-known *A** algorithm. The

authors are aware of the NP hard nature of the problem and a comprehensive analysis of the performance of the *A** algorithm based on the heuristic function is provided. The main problem identified is handling the complexity of large problems. To make the *A** algorithm progress quickly to a solution (any leaf in the reachability tree is a solution), they propose a heuristic function that makes the algorithm prefer markings that are closer to the solution. The main problem with this heuristic function is that it does not contain any information about the current state of the system and that it is difficult to tune. This leads to unforeseen and weak results as we will analyse in chapter 6.

Nevertheless, the main contribution of this work is the clear suggestion of the combination of heuristic search and *PN* modelling, leaving the door open for an effective integration of both technologies. Many researchers have considered this work (not least this thesis) as a starting point for further research. As an example, [Lee 92b] adapts the previous work for the periodic scheduling of *FMS*. *L1* is modified by the inclusion of a restriction on the number of times each transition can be fired to complete a cycle.

[Lee 94b] follows the approach of [Lee 94] and extends the *PN* model to consider the application of AGV and limited capacity intermediate storage. Two types of AGV layouts are considered: Centralised and distributed. Also, limited buffer capacity is modelled. The main contribution of this work is an improvement of the heuristic function proposed in [Lee 94]. Being aware of the difficulties of tuning the heuristic function proposed there for each problem, they present an alternative that takes into account average operation cost. However, this is still far from being a *PN* based heuristic function and provides only small improvement to [Lee 94]

[Sun 94] also follows [Lee 94] in integrating AGV modelling and control in an application concerned with vehicle collision and traffic jam problems. The *PN* model, consequently, is not the usual approach. The algorithm is a version of a staged search algorithm *Limited expansion A**. The algorithm is the same proposed in [Lee 94] but with a limitation on the number of markings as candidates to be explored, which prevents exponential explosion. However, the heuristic function employed is the same as in [Lee 94]. This is a typical example of work that both uses heuristic functions with a depth-first search component and an irrevocable decision strategy. This has clear drawbacks as we will see in chapters 4 and 6. For example, the work [Yim 96] tries to

overcome the tuning difficulties of [Lee 94] with a heuristic function by including information about the specific system. Again, It is interesting to note that this paper introduces irrevocable decisions (as in [Sun 94]), by limiting the number of markings to be considered during the search. They conclude that they do not observe any difference if such limitation is employed. This suggests the need to separate control of the search effort from the application of *PN* based heuristics to direct the search.

[Sutdhiraksa 96] does not study *FMS* descriptions but considers the problem of robotic assembly. A single case study consisting of two robots arms assembling two pieces of a product with three assembly steps, is modelled as a timed *PN*. Three scheduling approaches based on reachability graph analysis are proposed for comparison. The first two are based on a combination of simple splitting up techniques and the entire generation of the reachability graph. To improve the results, the *A** approach of [Lee 94] is proposed although no algorithm description is given. They report unsuccessful results for medium problems due to computational expense. Apart from the interest of applying *A** in a rolling horizon manner, an interesting aspect of the work is the use of heuristic knowledge about the specific problem domain to reduce search.

[Jeng 98] (see also [Jeng 96], [Jeng 99]) follows [Lee 94] in using *A** but proposes the use of the heuristic function of [Cheng 95]. The *FMS* considered allows alternate routing of symmetric and asymmetric types. The buffer policy, is (as in [Lee 94]) unlimited storage. The heuristic function *h(m)*, is based on the minimisation of the total operation cost of the remaining operations from *m* to the goal marking, constrained by the *PN* state equation. This is an important advance on [Lee 94] since the heuristic estimation now takes into account global information on the system state represented by the *PN*. The heuristic, nevertheless, tries to give a value for the minimum sum of total operation costs, but not the makespan. As a result, the heuristic function also makes the algorithm progress quickly to a first (although not optimum) solution.

The paper also presents a pruning procedure with the objective of pruning paths that represent permutations of concurrent transitions, which yield equivalent schedules. However, the amount of space that is pruned is determined by a comparison with [Lee 94]. These results are problematic, since a proper comparison must be made using exactly the same algorithm and heuristic function with and without the pruning method. It is interesting to note that the algorithm settings for [Lee 94] are not specified.

The main contribution of this paper is the use of a heuristic function that depends on the state of the system and is *PN* based, but the system described has the same problems as [Lee 94] when fighting the combinatorial explosion.

[Jeng 98b] tries to improve the work of [Jeng 98]. The *PN* heuristic function in [Jeng 98] was presented as a solution of a linear system of equations based on the *PN* state equation. In [Jeng 98b] they become aware of a simplest way of obtaining the heuristic in [Jeng 98], which is similar to the *PN* analysis that will be presented in chapter 4. However, some of the heuristics appear to have limited the applicability to *FMS* in which the batch size for each job is restricted to one. The second aspect of interest in the approach is that the backtracking capability of *A** is limited. This supposes a compromise between pure *A** and the backtracking free algorithm of [Cheng 95]. Criticism of this approach will be discussed in chapter 6. Here we simply note that they consider a limited backtracking depth of four, which is a conservative setting but necessary to avoid exponential explosion. The problem with this approach is the so-called deadlock recovery capability (limited buffer capacity is considered). It is not completely clear how this can be achieved as they propose an arbitrary pruning of paths.

[Xiong 98] proposes the combination of best first (BF) and backtracking (BT) search. Although, the problem considered is a simple *JSS* with no alternative operations and infinite buffer policies, the approach is easily adaptable to *PN* models of alternative routings and buffer constraints. This is because the reasoning mechanism is not based on a mathematical formulation but on the *state-space* description of the *PN*. The proposed algorithms can thus be directly applied to *PN* models with typical *FMS* features such as alternate routings or plans. This contrasts with the approaches followed by [Hillion 88, 87, 98] were the introduction of flexibility leads to major changes in the scheduling procedure proposed. A major problem with the approach is the absence of heuristic information to guide the *BT* (Branch & Bound) phase, so results are not completely satisfactory. We will return to this approach in chapter 6.

Although published earlier, [Xiong 97] is an extension of the previous work [Xiong 98] but with the aim of testing the capability of the algorithm to avoid deadlocks. Infinite buffer, no buffer, and limited buffer are considered, as well as the modelling of material handling devices. The main conclusion is that Petri Nets provide an explicit way to

represent deadlock states, and traditional backtracking approaches can explore alternate paths based on this information. The scheduling algorithm employed is the BF/BT strategy of [Xiong 98]. For the BF (A^*) stage, the heuristic function proposed is the maximum sum of operation times of those remaining operations for all jobs which are planned on each machine. Although the function is admissible for a pure *JSS*, it cannot be applied if alternate routing is considered since machines are not assigned to processes.

[Chiu 97] propose an embedded Genetic algorithm approach for the scheduling of *FMS* problems. Although no *PN* model is described or explained, the authors seem to be employing a timed place *PN*. The model of the system is decomposed into a *transportation model* and the *process-flow* model.

To reduce search, the methodology is based on a time-decomposition approach, where parts are scheduled in a rolling horizon manner. A limited *WIP* limit is initially defined, then the total schedule is generated segment by segment, each segment being the result of a *GA* search.

The main problem with this approach is that the *PN* are employed only as a simulation tool, the *PN* state is transformed into a chromosome representation, the *GA* obtains a solution (chromosome), which must be transformed into a feasible solution for the *PN* model.

This work is based on a previous paper [Liu 92] that employs the same method but substitutes the *GA* algorithm by A^*. Unfortunately, this work does not make use of *PN* modelling, and it is merely descriptive from the point of view of A^*.

In a recent work, [Inaba 98] considers a flexible formulation of mechanical assembly systems. They consider a *PN* model that integrates path selection and machine selection. All parts are loaded from a common stack where infinite is assumed. All operations are performed using a single machine. The A^* approach of [Lee 94] is adopted, and includes the limitation of the number of markings to be explored proposed in [Yim 96] and [Sun 94]. The heuristic function proposed is basically the same as [Lee 94b].

This work again demonstrates the potential of *PN* reachability analysis based scheduling in the sense that previous results from *FMS* domains are easily extended to other scenarios such as Flexible Assembly systems with minor variations. From the point of view of scheduling methodology, the novelty of the approach is the inclusion of

a technique to identify repetitive processes. This captures a repetitive good strategy if large quantities of parts are considered. The procedure identifies previous decision scenarios in terms of machine status. For example, if a previous similar status of resources is found along the path from the initial state to the current marking, one can repeat the actions previously taken and progress in a decision free manner. If this happens, the *A** algorithm is re-started with the new marking as the initial marking.

In our opinion, this is an interesting idea to be considered in the future, but the paper fails in addressing it clearly and many questions arise. For example, what is the chance of finding a repetitive process? Since once a repetitive process is identified, the current path is considered as irrevocable, how would you know that you were repeating something that was good? The experimental results, based on a single simple problem instance, are not compelling.

Finally, [Boutet 98] considers the *FMS* scheduling problem in terms of Preliminary System Design of *FMS*: the static part of the *FMS*: resources, jobs and routes are modelled with timed *PN*, but they consider interval membership constraints on operation duration. Constrained Predicate Nets [Kubek 95] are proposed to include global constraints such as due dates. Although no specific algorithm is given, the paper concludes that studying the relationship between conventional graph-oriented scheduling methods from OR and *AI* with *PN* reachability analysis methods should lead to theoretically and practically useful results.

7. Summary.

The scheduling of an *FMS* requires two fundamental things to be done: a) define a representation paradigm that captures the essence of the system and how it works and b) reason about its behaviour, i.e., to develop methodologies that solve the scheduling problem based on the formal expression that it is the model.

The first part of this chapter reviewed how this issue has been addressed in the literature. Two main groups of approaches have been identified. The first one (*OR* methods), pays less attention to the modelling and concentrates on producing optimal schedules. Unfortunately, the main disadvantage is that such models do not naturally reflect the complex structure of *FMS* and their results are difficult to translate at the

industrial level. The second well-known group are the on-line/real-time methods, which propose representation paradigms based on simulation tools that are better qualified to capture the full operation complexity of *FMS* but the scheduling techniques that they offer often revert to heuristic techniques that can not deal with the operational complexity of *FMS*.

The interest of *PN* to the solution of the *FMS* scheduling problems is not only due to the fact that *PN* are a powerful modelling tool for the simulation of production processes and, consequently, that they have been extensively used in the on-line control of *FMS*. What it is of real interest, is that the *PN* formalism may also be viewed as a definition of a state-space structure in terms of state and operators. This is a representation paradigm very familiar in *AI* planning. We believe this represents a promising scenario that has the potential to reduce the gap between advances in the theory of *AI* search techniques and the industrial level [Jianjun 92] by the integration of both *AI* search based on domain heuristic knowledge supported by the powerful analysis and simulation tool that is the *PN*.

Although this integration has already begun, this research direction is relatively recent and immature. We will provide a better insight on this matter in the following chapters.

Chapter 3. *PN* modelling of *FMS* and scheduling based on *PN* reachability analysis.

1 Introduction.

This chapter is about providing *PN's* and associated techniques that allow *PN* to be used to schedule *FMS*. As explained in the previous chapter, *AI* techniques are potentially relevant to solving problems based on finding sequences of operations in *PN* based models of *FMS*. First, we give a definition of the *FMS* model that we will use in the thesis. Second, we describe how a *PN* can represent the constraints and dynamics of a discrete event system. This chapter does not attempt to describe a complete modelling methodology for *FMS* using *PN*s, but we will introduce a new class of *PN,* the *cb-NET* that is intended to allow us to represent manufacturing systems with features that are typically considered difficult to model. Next step we to define an architecture in which *PN* and *AI* search algorithms are integrated. Finally we present an automatic synthesis procedure for *PN* models from *FMS* specifications. A basic language for specifying *FMS* descriptions is given, and the top-down methodology implemented by the parser for this language is outlined. Application examples are given at the end of the chapter.

2 *FMS* formulation.

In this section, we propose a problem formulation that captures what we believe to be the major characteristics of a general *FMS* layout. To consider all the aspects of an *FMS* layout would be too great a task [Basnet 94]. Bear in mind that we would be required to deal simultaneously with routing flexibility, part transportation, buffer spaces, tool slots, machine availability and pallets. Few authors consider all these characteristics simultaneously, and in the cases that do there are serious concerns over the ability to actually define solutions for general FMS systems [Basnet 94].

The algorithms and work developed in this research may be described as studying novel scheduling approaches applied to a general *FMS* description which are validated with random instances that attempt to capture the essence of the problem

domain. We will consider an *FMS* defined as a set of different product types to be produced in the system. A job type (or product type) is defined in terms of a number of different paths or plans that describe how to produce that part. Each plan is formed by an ordered sequence of operations or tasks. The term operation applies to any loading/unloading operation, part transportation or machine processing. Due to machine flexibility, an operation can be carried out by different resource sets. After the completion of an operation, parts may be transferred to intermediate buffer facilities where capacity and residence time constraints apply. The scheduling objective is to minimise the makespan or total completion time.

The description above may be defined more formally as:

Definition 1: *FMS* notation.

An *FMS* consists of:

1. m available resources $\{R_1, R_2, \ldots\ldots R_m\}$.
2. n jobs to be processed $\{J_1, J_2, \ldots\ldots J_n\}$.
3. Each job J_i has p_i process plans
4. Each plan P_{ij} has a sequence of q_{ij} temporally related tasks $\{T_{ij1}, T_{ij2} \ldots\ldots T_{ij}q_{ij}\}$ ordered by the technological constraints[1].
5. Each task can be processed in several ways. C_{ijk} stands for the number of different possibilities (choices) for achieving task T_{ijk}. S_{ijkl} is the set of resources needed for choice l of task T_{ijk}. h_{ijkl} is the processing time for task T_{ijk} using resource set S_{ijkl}.
6. The processing of task T_{ijk} using resource set S_{ijkl} is termed an operation, (O_{ijkl}).
7. Intermediate part buffers may exist between tasks. They may be constrained in capacity and time.
8. The following standard assumptions apply:
   - Machines are always available and never break down.
   - Each machine can process at most one task at a time.
   - Any task T_{ijk} can be processed using at most one resource set S_{ijkl} at any time. Each task consumes a single subpart of the previous unit and produces only a single subpart. This means that no assembly/disassembly procedures are allowed.

[1] The indicies refer to: *i-th job, j-th plan, k-th task, l-th choice.*

- Operations are non pre-emptive, i.e, once an operation has started, it cannot be interrupted.
- Set-up times are independent of the schedules and are included in processing times.
- Processing times, due dates, ready time and technological constraints between tasks are deterministic and known in advance.

The following section will present the *PN* formalism as a tool for modelling such scenarios.

3 Constrained buffer Petri nets: *cb-NETs*.

Having defined such a generic problem formulation for an *FMS* it is now the time to describe how such scenario can be modelled as a *PN* structure.

Although we assume the reader familiar with basic *PN* theory (see [Murata 89]) the following sections cover basic background.

FMS buffer constraints have also not been considered in the relevant work reviewed in the previous chapter (section 6.2) since they employ deterministic time *PN* [Murata 89] which are not sufficient for modelling these constraints. To handle them, we have defined a class of *PN* that we have called *constrained buffer nets* or *cb-NETs*. Extending the definition of ordinary *PN* has been a traditional solution independently carried out by authors. For example, [Khansa 96] and [Caramihai 98] extend the *PN* definition to include a time limit for the firing of transitions. These models have been recently applied to the *hoist* scheduling problem in [Calvez 98] and [Dutilleull 98]. However, integration of both models to effectively model residence time constraints is needed.

It is not our intention to define a complete modelling paradigm based on *PN* for production processes. Our aim is to demonstrate that *PN* integration within *AI* problem methodologies is straightforward (for example recently [Konstas 98] has proposed a rule-based *PN* definition that allows the modelling of time constraints). The situation appears to be radically different from traditional modelling approaches, such as mathematical models or disjunctive graph models, where inclusion of constraints other

than sequence dependencies usually leads to major revisions of the search algorithms involved.

Definition 2: A *cb-PN* is a tuple

$$N=(P,B,T,I,O,\gamma,\tau,\varphi,M_0)$$ where:

- $P=\{p_1,p_2,\ldots..p_m\}$ is a finite set of places.
- B is a subset of P that represents intermediate storage buffers. We call a place $p\in B$ a *buffer-place.*
- $T=\{t_1,t_2,\ldots\ldots t_n\}$ is a finite set of transitions with $P \cup T \neq \varnothing$ and $P \cap T = \varnothing$.
- $I: P \times T \Rightarrow N^+ \cup \{0\}$ is an input function that defines directed arcs from places to transitions, i.e., if $I(p_i, t_j) >0$, then p_i is said to be an *input* place of t_j and we include an arc from p_i to t_j. If $I(p_i, t_j) = k$, then we label the arc k.
- $O: P \times T\Rightarrow N^+ \cup \{0\}$ is an output function that defines directed arcs from transitions to places, i.e., if $O(p_i, t_j) >0$, then we include an arc from t_i to p_i. Besides, p_i is said to be an *output* place of t_j. If $O(p_i, t_j) = k$, then we label the arc k. Additionally, $C: P \times T \Rightarrow N^+ \cup \{0\} = [c_{ij}]$ where $c_{ij} = O(p_i,t_j) - I(p_i,t_j)$ is called the incidence matrix.
- $\gamma : P \rightarrow R^+ \cup \{0\}$ is the delay function assigned to a place.
- $\tau : T \rightarrow R^+ \cup \{0\}$ is the firing time function.
- $\varphi: B \rightarrow R^3$ is a 3-tuple associated with *buffer-places* that contains the constraint information which is interpreted in the following way: the first component $\varphi^1(p)$ refers to the maximum number of available tokens that must reside at the place (buffer), the second $\varphi^2(p)$ is a constraint on the minimum residence time of a token in that *buffer-place.* $\varphi^3(p)$ is the maximum residence time for that token. It is noted that the intermediate storage capacity is measured in terms of the number of units, not the physical size of storage units.
- $M: P\Rightarrow N^+ \cup \{0\}$ is the m-component marking vector that defines the state of the *PN* and whose *i-th* component $M(p_i)$ is the number of tokens in the *i-th* place. M_0 is the initial marking.

Normally, the concept of time is not assigned both to transitions and places. Two main models have been favoured in the literature, timed-place *PN* and timed-transition *PN* and it can be demonstrated that both models are conceptually equivalent [Murata

89]. However, timed-transition *PN* produce simpler models and also carry an implicit concept of time limit, by fixing an exact duration of the operations. In chapter 5, we will require both these properties.

To show how *cb-nets* work, we start by defining a simple robotic example depicted in *fig. 1*: An *FMS* consists of a robot and a cutting machine. Pieces from the unprocessed parts buffer are transferred to the cutting machine by means of the Robot. Before pieces are processed, they wait in the machine's input buffer, and once processed they remain in the machine's output buffer until they are transferred to the completed parts buffer. The operation cost for the robot to load/unload a part is always 1. For illustrative purposes we assume that robot movements without parts have negligible cost. The processing time of the cutting machine is 2. Both the machine input and output buffer have a maximum capacity of 5 parts and the maximum time a part can stay in these buffers is 15 time units.

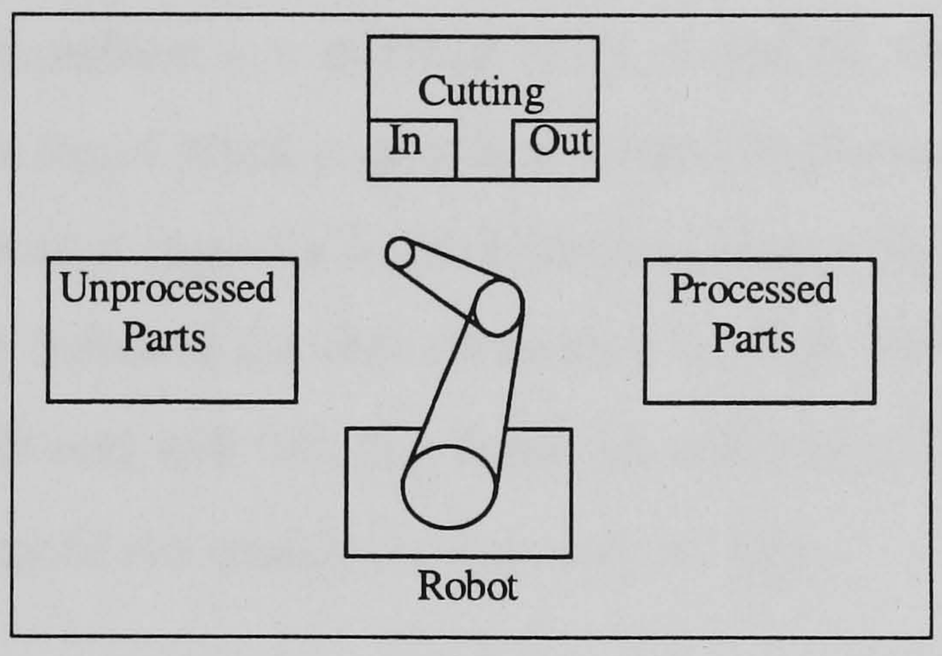


Fig. 1: FMS layout

To introduce the graphical concept of *PNs*, consider *fig. 2* which shows the *cb-NET* model for the system layout of *fig. 1*.

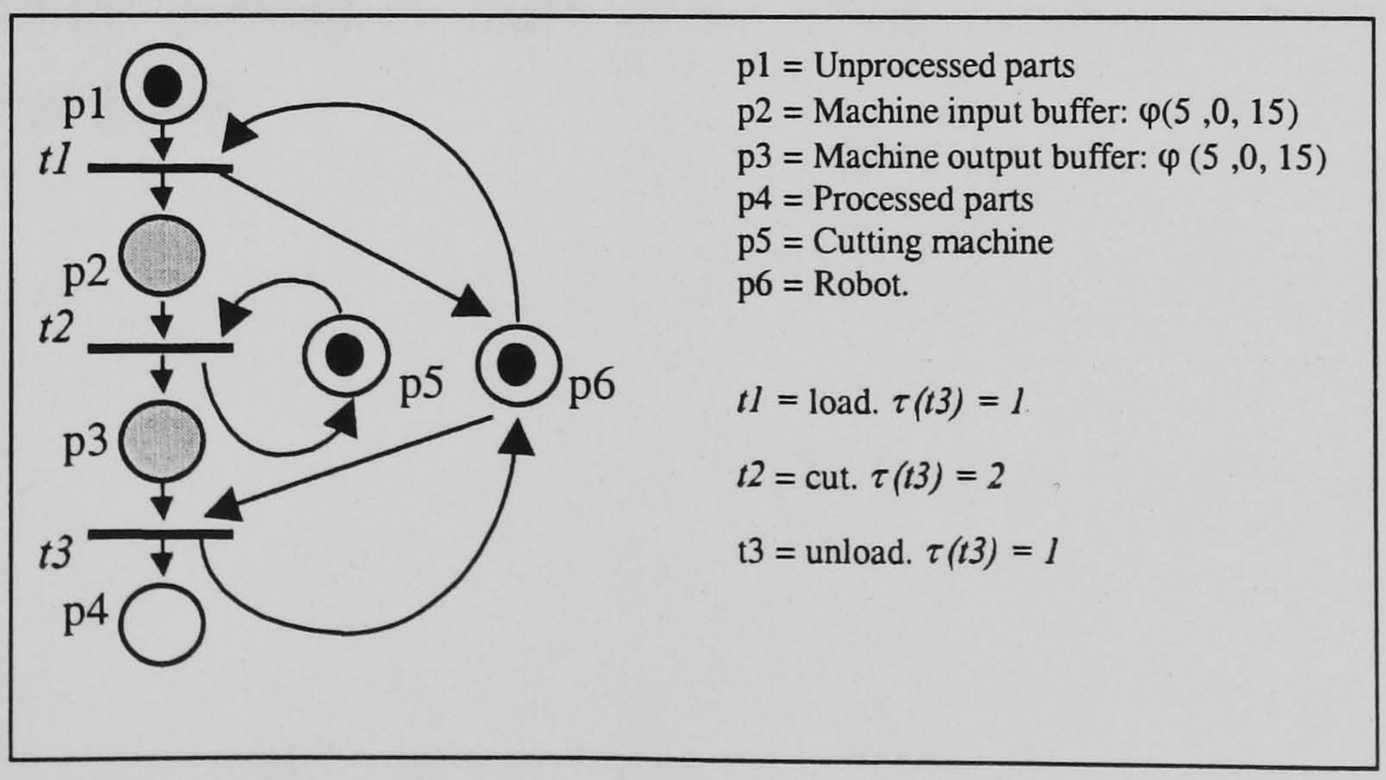


Fig. 2: cb-NET model of FMS example of fig 1.

In our example, places *(circles) p5* and *p6* represent the status of the resources, *p2* and *p3* represent buffers and finally *p1* and *p4* represent unprocessed and finished parts respectively. *Transitions (bars)* represent machine operations (but also events such as starting or stopping of operations). Places and transitions are connected by *arcs* as defined by the *cb-NET* input and output function. Shadowed places represent the cutting machine input and output buffers, which are constrained in the number of *tokens* (parts) they can hold and *token's* residence time. In the *cb-NET* of the figure, the initial marking (state) of the system is given by placing a token in the unprocessed part buffer and placing tokens in places representing resources indicating their availability.

Once a system state (marking) is defined, the question is: how does a *PN* work? The dynamics of the *PN* are explained in terms of marking evolution defined through a pair of rules, informally named the *token game* [Silva 89]:

1) *Enabling rule.* A transition t becomes *enabled* at M if each input place p of t contains as many *available* tokens as the arc weight.
2) *Firing rule.* Once transition t is enabled at M, it can be fired. The firing starts by removing from each input place p as many tokens as the weight of the directed arc connecting p to t. It also deposits in each output place p the number of tokens equal to the weight of the directed arc that connects t with p. The newly deposited token will stay *unavailable* and will become ready (*available*) after the time associated in p plus the firing delay of the transition; i.e., $\tau(t) + \gamma(p)$.

Given this definition, a marking M is said to be reachable from an initial marking M_0 if there exists a *firing* sequence of transitions that following the enabling and firing rule, transforms M_0 into M.

Figure 3 shows the marking evolution of the *cb-PN* resulting from the following firing sequence: *t1,t2* and *t3*.

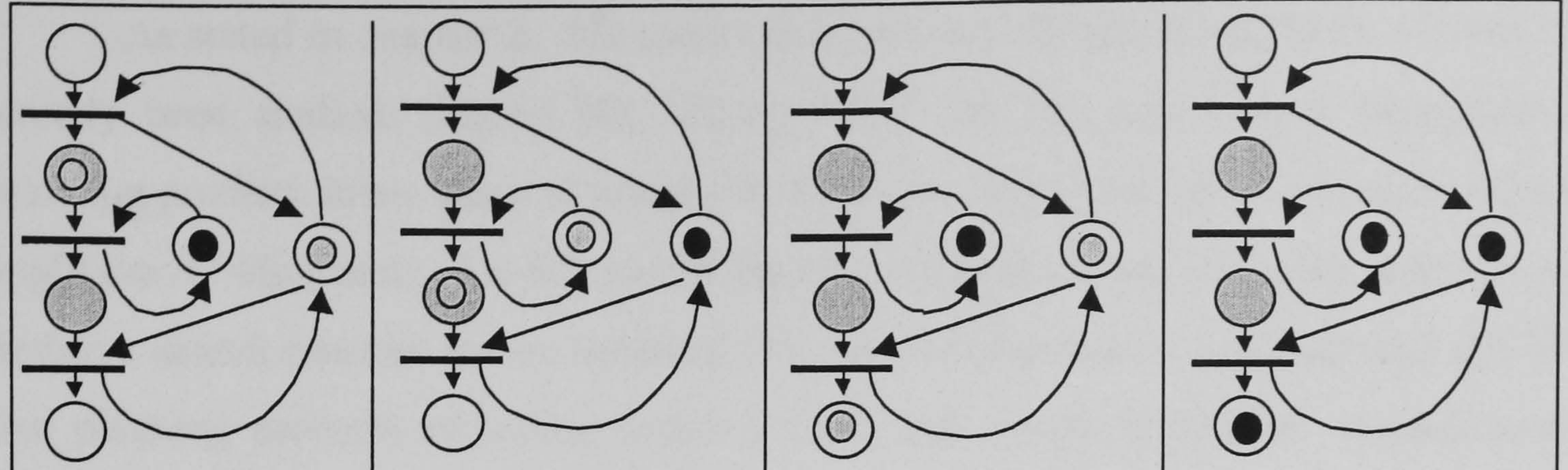


Fig. 3:Marking evolution for the cb-NET model of FMS example of fig 1.

4 Scheduling *FMS* based on *PN* dynamics and heuristic search within the *PN* reachability graph.

So far we have given a definition of the *FMS* problem domain and given a *PN* extension that able to model the *FMS* formulation given in section 2. The *cb-NET* model contains the constraints of the system and the *PN* theory provides the rules that define its dynamics. However, *PN* theory does not provide any conflict-solving method, since the firing rule does not specify, for example, which of the transitions among those enabled should be fired or when. In other words, the control of the system is not determined from the point of view of decision making. This is where *AI* based search methods may be employed.

A Petri net (and consequently a *cb-NET)* can be seen as a definition of a state-space structure. A state-space structure is described in terms of a formal definition of states and operators. Operators are partial functions that map states into states [Newell 72]. The problem space representation is a classic representation formalism used by *AI* problem solving methodologies. The problem is defined by the initial world model, the set of available operators and their effects on world models, and the goal statement [Fikes 71]. The solution of a planning/scheduling problem is defined as the problem of finding a sequence of actions that achieves a goal state from an initial state, often optimising some performance criteria.

As stated in chapter 2, this connection between *PN* and *AI* problem solving has already been studied. [Zhang 90] [Zhang 92] [Yim 94] transform a propositional planning problem into a class of timed *PN* (Predicate-Transition Nets) [Genrich 81] and apply the *A** algorithm. [Yu 97] model the classic blocks world planning problem and devise a search strategy for its solution. The parallels between a propositional *STRIPS* like planning problem including temporal costs and timed *PN* can be adapted to the *FMS* scheduling problem and are summarised in *table 2* [Zhang 92].

Propositional planning/scheduling.	***PN* based scheduling.**
Elements of a state description: *machine, buffers and other resources*	places in the *cb-NET*
A state description	A marking
Initial and goal states	Initial and goal markings
Operations: preconditions → Actions.	A transition: enabling rule + Satisfaction of buffer constraints→ firing rule.
State space	*cb-NET* reachability tree.
Plan (schedule) and its temporal cost (makespan)	Firing sequence and time associated to a marking

Table 2.Parallels between prepositional planning and a PN model.

In the previous table, the problem search space is identified by a concept known as the *reachability* tree. The reachability tree [Murata 89] associated with a *PN* is a graph in which each node represents a marking reachable from M_0 and each arc represents the firing of a sequence of transitions. The reachability tree for a *FMS* description modelled with a *cb-PN* is finite[2].

Consequently, if the *cb-NET* model is a description of the *FMS*, the scheduling problem is translated into a search problem within the state space defined by the *cb-NET* reachability tree. An optimal schedule can, in general, be obtained by generating all the solution paths. The problem is that, for the general case, the generation of the reachability tree takes exponential time [Murata 89][3]. The minimum architecture that enables a *PN* based scheduling with an appropiate *PN* definition (*cb-NETs* in our case) is depicted in *fig. 4*.

---

[2] Since *cb-PN* are bounded and irreversible *PN* (see chapter 4).

[3] An interesting empirical study of the complexity metrics of *PN* with the reachability tree as the target problem can be found in [Soo 92].

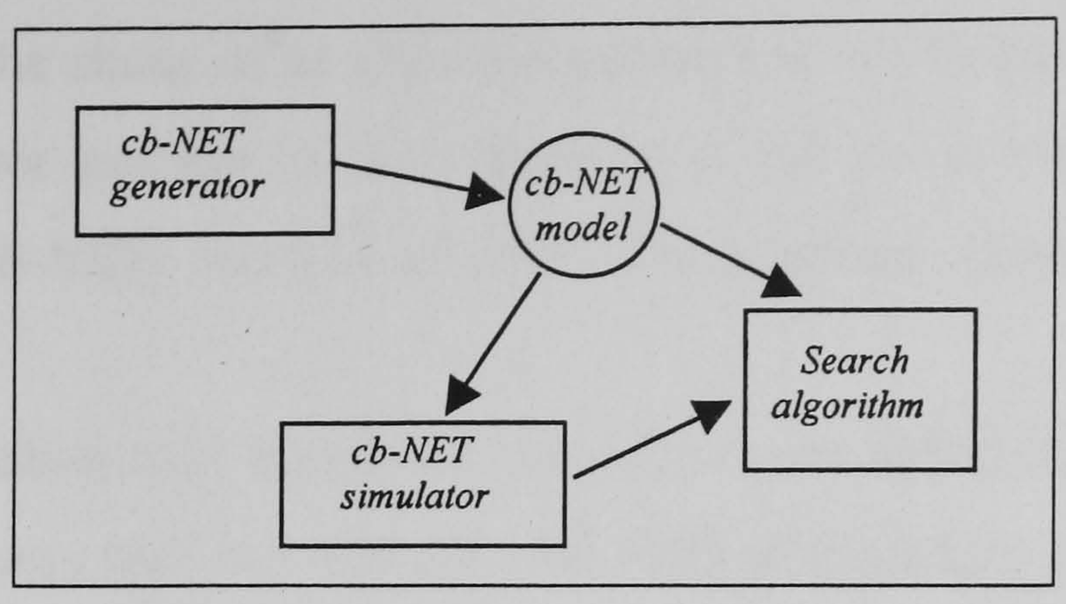


Fig.4: PN based scheduling architecture.

Apart from the *FMS* model in terms of a data structure that contains the *cb-NET* mathematical expression, the architecture depicted in *fig. 4* contains three elements. (1) a *cb-NET* simulator, that is in charge of the dynamics of the model (*enabling*, *firing* rule and *buffer-places* constraint satisfaction). (2) a search algorithm that explores the *cb-NET* reachability graph with the objective of finding an optimum solution. (3) an automatic generator of *PN* models from textual *FMS* specifications following the *FMS* formulation given in definition 1.

Both the simulator and the search algorithm make use of the *cb-NET* model. The former as an input to perform the *token game* and the latter as a mathematical expression to obtain information about the system to guide the search process. The search algorithm is in charge of constructing the reachability tree (or search graph) by generating feasible sequences of transitions. To determine the operations (transitions) that can be applied and to obtain new states (markings) the search module makes use of the simulator. Each of these modules is described next.

4.1. A Flexible Manufacturing System Modelling Language *FmsML*.

Recently, several works have addressed the synthesis of coloured *PN* for *FMS* task specification [Villarroel 88] [Xue 98] [Arjona 96] [Camurri 93] [Zimmermann 99]. These works are interesting from the point of view of developing methodologies and experiences of automatic *PN* synthesis rather than in providing formal languages for the specification of production processes. To obtain any practical benefit, the results must be integrated with *PN* theory for large system analysis and *PN*-independent commercial manufacturing system's description languages such as WITNESS [Witness].

However, for the shake of an effective study, test and validation of the methods and algorithms that we propose in this thesis, the tedious and error prone task of manually construct *cb-NETs* models of *FMS* formulations needs to be practically supported[4].

The *FMS* definition tool developed here allows the definition of resource types, buffers, and jobs in a way that is oriented to the *FMS* formulation given in definition 1. The prototype implemented consists of a formal language called *FmsML* (*Flexible manufacturing system Modelling Language)* in which the *FMS* is encoded. The encoding is parsed generating source code in *C* that contains a sequence of functions that construct the *cb-NET*. When the code is executed, it generates the data structure that is the *cb-NET* model (depicted in *fig 4*). Figure 5 shows the data flow for this process.

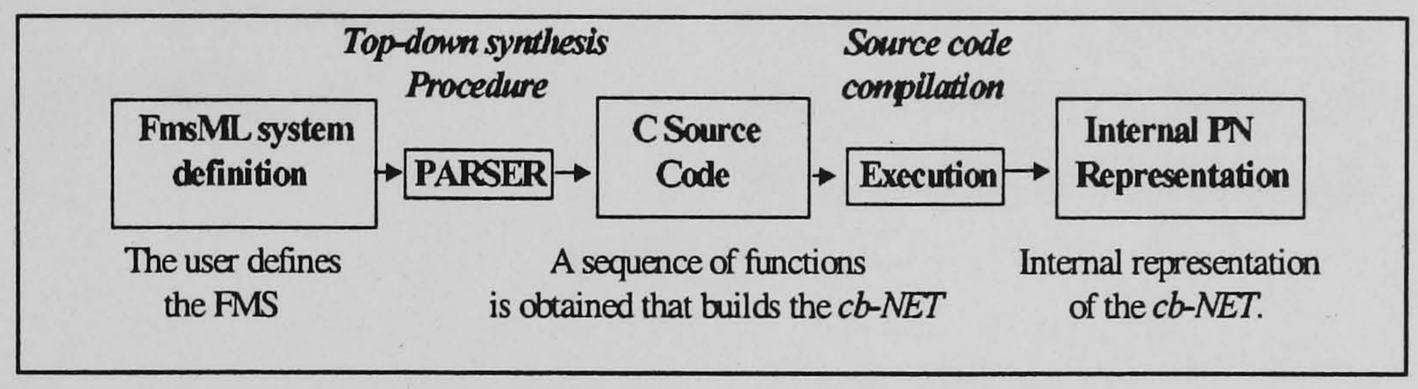


Fig. 5. Data flow

The lexical and syntactical specifications for the grammar of *FmsML* are coded in standard *BNF* format for *lex & yacc*. *Lex & yacc* [Mason 90] were originally designed for *UNIX* as automatic generators of lexical and syntactical analysers. The user gives the specification of the grammar in *BNF* and the semantic procedures (coded in *ANSI C*) to be called when the statements of the grammar are found (semantics of the grammar). *Lex & yacc* generate source code in *ANSI C* with the lexical analyser and merge the syntactical analyser with the semantics defined, thus obtaining a complete parser that implements the synthesis procedure presented in the following section.

[4] For example, a typical *FMS* description consisting of ten jobs and eight resources presented in [Lee 94], assuming infinite buffer policy (which simplifies the model), has a total of 63 places, 91 transitions, and 318 arcs. The code for defining this *FMS* description in a textual language for creating *cb-NETs* is approximately 700 lines long. This language, named *PnML,* has been developed as part of this research, but its description is beyond the scope of this thesis. A brief description of it can be found in [Reyes 98].

4.1.1 A *top-down* synthesis procedure based on the *FmsML* syntax.

The parsing of an *FMS* description is a *top-down* synthesis directed by the *FmsML* syntax[5]. The procedure considers each job as a subsystem that competes for the use of resources. First, resources are created as places. Then there is a stepwise refinement of *sub-PN* (that can also be viewed as complex transitions). The upper level is the definition of a job and the lower level is formed by the *cb-NET* model of operations. Through the following description, we will make use of the *FMS* notation given in *definition 1.*

1st Step: Resources and jobs.

```
RESOURCE_CONSTRUCT:                    BUFFER_CONSTRUCT:
#resource SYMBOL                       #buffer SYMBOL
      [units  = INTEGER ;]                   [capacity  = INTEGER ;]
      [set_up = REAL ;]                      [min_waiting = REAL ;]
#end ;                                       [max_waiting = REAL ;]
                                       #end ;

JOB_CONSTRUCT:     #job SYMBOL PLAN_CONSTRUCT* #end ;
```

Two types of resources can be defined: resources such as machines, robots, AGV, etc., and buffers. A resource is defined by a name, a number of units of the same resource type (parallel processors) and (optionally) a set-up time can be specified. A buffer is defined by a name and the constraints applied on the buffer. Each job is defined by the different plan descriptions to achieve the parts of this type. The parsing of such structures is as follows:

Every time the parser analyses a correct `#job` statement, it creates two new places. The first one, the *input buffer,* represents the unprocessed parts that are ready to enter the system. A second place (the *output buffer*) represents parts that have been processed. The *input(output) buffer* acts as an input (output) place for p_i sub-*PN*s that model each of the p_i plans for the job J_i. Any *#resource* or *#buffer R* construction of the

[5] See [Jeng 93] for a review of top-down, bottom-up synthesis methods and [Narahary 85] and [D-S. Yim 94] for examples of these methods.

language is modelled as a single place. If R_i is found by the parser within the `#plan` statement then R_i is an input/output place for that *plan*. *Fig. 7* shows this organisation.

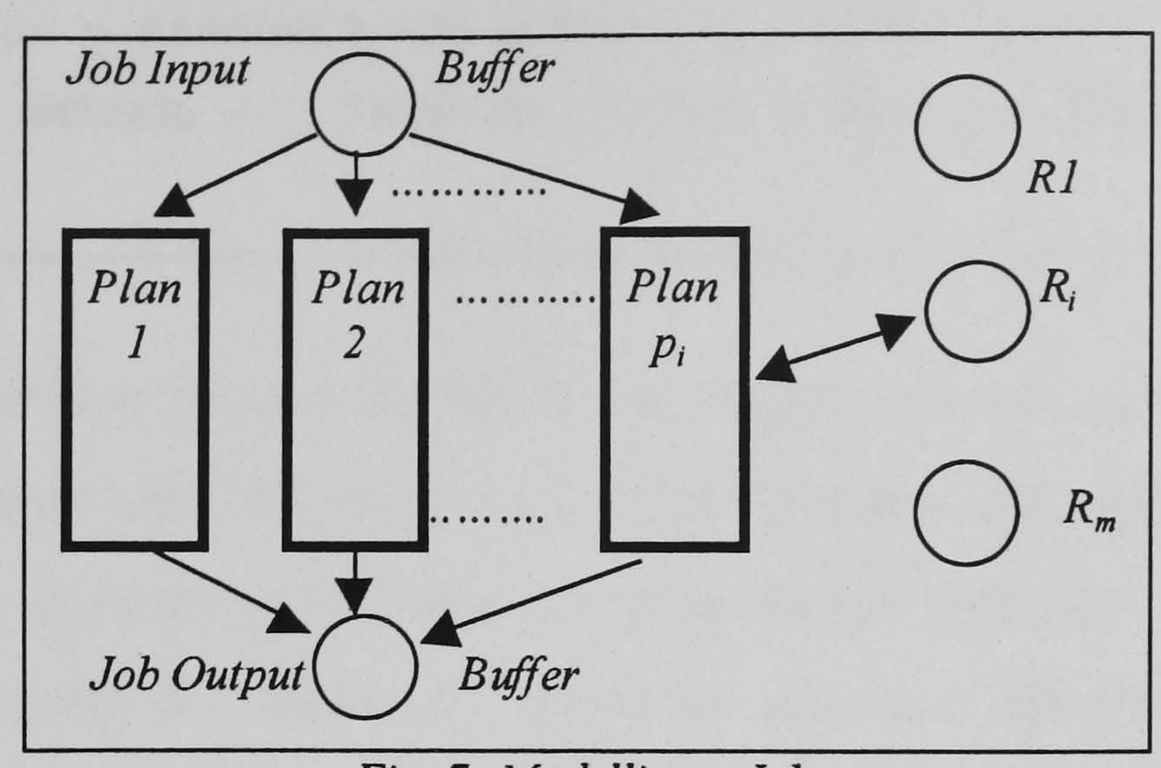


Fig. 7: Modelling a Job.

Step 2: Plans.

```
PLAN_CONSTRUCT:    #plan SYMBOL TASK* #end;
```

The j_{th} `#plan` statement that defines process plan P_{ij} for job J_i is formed by a sequence of q_{ij} *tasks* to be performed. Between any two stages or tasks (sub-*PN*s) a buffer policy is applied. These buffers are modelled by places, but the buffer specific policy is not yet known. The first (last) operation uses as a buffer the *input* (*output*) buffer defined previously (see *Fig 8*). Any place representing a buffer is an input (output) place for the next (previous) `#task`, except the *input* and *output buffers*. If any of the alternatives for achieving a task use resource R_i then the place representing the resource is an input/output place for the sub-*PN*.

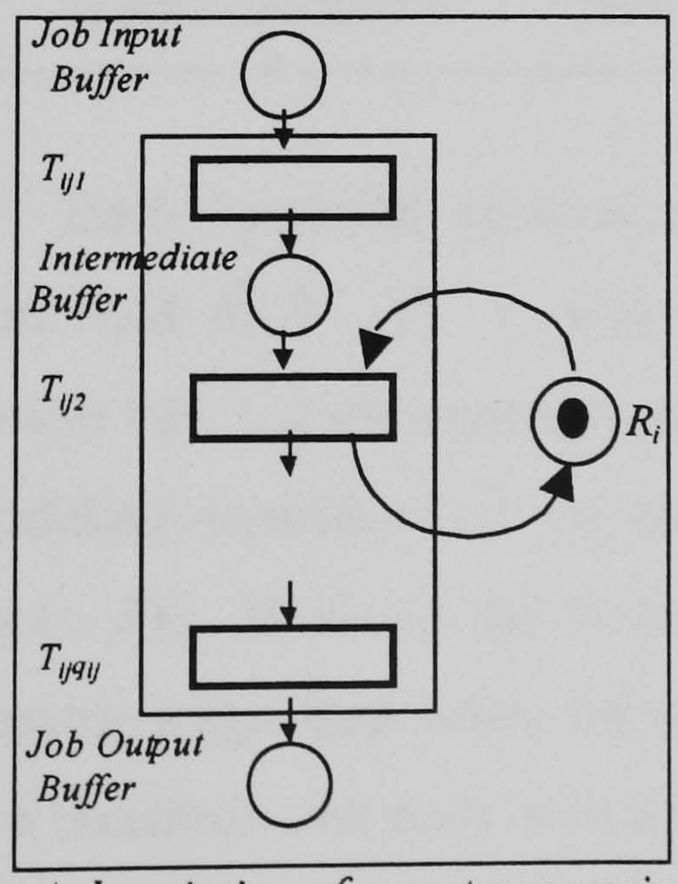


Fig. 8: A description of a part-processing plan.

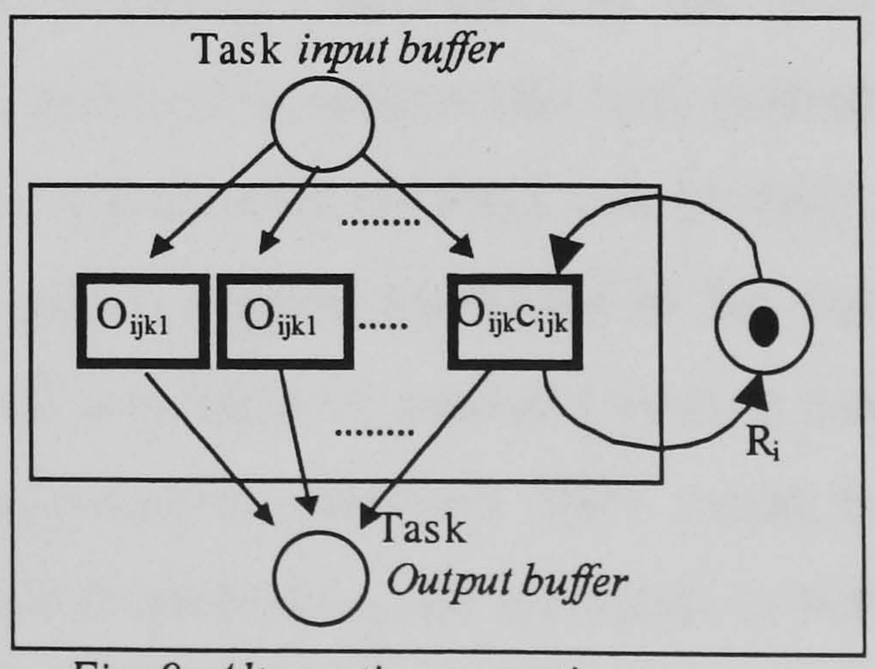


Fig. 9: Alternative operations.

Step 3: Tasks.

```
TASK_CONSTRUCT:    BUFFER_REFERENCE: SYMBOL
#task SYMBOL
      description = RESOURCE_LIST:REAL [| RESOURCE_LIST:REAL]*;
      [buffer = SYMBOL / ( INTEGER , REAL , REAL) / NULL / NO_WAIT ;]
#end ;
```

A task is defined by two attributes: *description* and *buffer*. *description* specifies the resource sets that may be used to perform the operation and *buffer* indicates the buffer policy applied after the finish of the operation. A shared buffer is specified by addressing the name of a previous *#buffer* construction. A dedicated buffer (or infinite buffer policy) can be specified by setting *cb-NET* values for the task output buffer defined in the previous step.

When the parser finds a *#task* statement, the first step is to determine the number of alternatives (choices) to perform the operation. This is achieved by analysis of the attribute *description*. Each alternative is defined by the set of resources needed and its temporal cost.

Fig. 9 shows how a task $T_{ijk,}$, that can be achieved by C_{ijk} alternative operations, is modelled as C_{ijk} sub-*PN* that share the input and output places of the *task*. If resource R_i is addressed in one of the alternate resource sets, then R_i is an input/output place of the sub-net representing that choice.

Step 4: Operations.

```
RESOURCE_LIST:     SYMBOL [ & SYMBOL ]
```

Each operation alternative of the *description* attribute has the following format: R_1 & R_2 & ...R_i : t , where R_1 ...R_i is the list of resources that may perform the operation and t is the operation cost. Each of these alternatives for a task (namely O_{ijkl}) is modelled dependent on the output buffer policy applied expressed in the *buffer* attribute. *Fig. 10* shows the three basic *cb-NET* structures or modules used to model a processing stage, dependent on which storage policy is employed. Each model begins with a transition and ends with a place. This structure realises the connectivity between the basic modules.

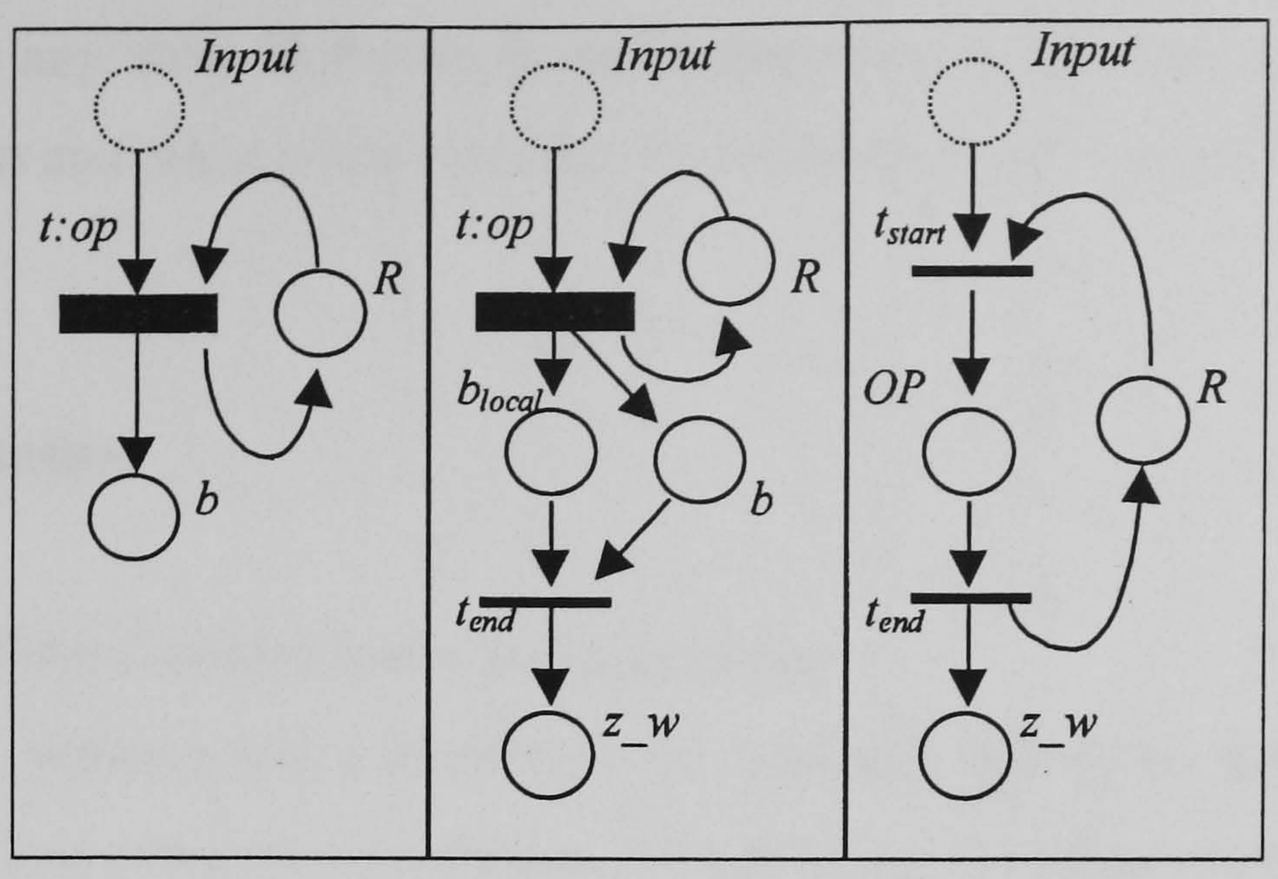


Fig. 10: cb-NET basic modules.

Figure *10 (left)* models a *Buffer policy with a dedicated buffer:* This basic module models the existence of a limited buffer dedicated to a single stage of a part plan (i.e, not shared by other processing stages). The operation is represented by a transition *t*, the firing delay associated with the transition models the operation cost *op*. A *buffer-place b* represents the buffer. Depending on the constraints, it can model an infinite, finite, or zero-wait buffer policy. The dynamic sequence is as follows: *(1) t* becomes enabled, *(2) t* is fired, releasing an unavailable token in *b, (3)* the token becomes available after *op* time units.

Figure *10 (center)* models a *Buffer policy with common buffer.* When a central storage facility (or input/output buffers located at the machines), exist, job stages compete for the use of the buffer. In other words, the buffer is a shared resource. *t* models the operation that once fired, releases a token into b_{local} and *b*. b_{local} models the part status; whilst *b* is a shared resource limited by the buffer capacity. t_{end} is an immediate transition that releases all the resources used by the task when the next operation starts. This is followed by a *zero-wait* buffer at the end. The *cb-NET* controller is then forced to start the next operation when the part is removed from the common buffer. Time constraints are associated with $b_{local.}$ A second level of capacity constraint can be defined for $b_{local.}$, which models that a maximum of $\varphi^1(b_{local})$ parts of this specific job type can wait in *b*.

Finally, *fig. 10 (right)* models a *Null-buffer policy:* where a part can wait in the machine after being processed. The models employ a transition t_{start} that marks the beginning of the operation and consumes the resources. Place *p* models the operation time using $\varphi^2(b)$. The token becomes available and can be transferred to the next

processing stage at any time. However, it will occupy the resource until the transition t_{end} transfers the part and releases the resource. A *zero-wait* place finishes the sequence.

4.2 *cb-NET* simulator.

The *cb-NET* simulator performs two operations:

a) Given a marking and a transition, the simulator determine which transition will become enabled within the current marking. It makes use of the *cb-NET* model to check the *enabling* rule. Also, it determines if the application of the transition will violate the constraints associated with *buffer-places*.
b) Given a marking and a transition that is enabled, the simulator makes use of the *cb-NET* model to obtain a new marking as a result of firing the transition. It is assumed that the application of the transition does not violate *buffer-place* constraints, because this was checked in *a)*.

Consequently the *cb-NET* simulator actualises a marking by firing an enabled transition that does not violate any buffer constraints. The marking is actualised by token redistribution and time information. In other words, the system keeps track of the time status of tokens in the places. This information is used to determine the time cost of the schedule and to verify residence time constraints.

4.3 Search algorithms based on *cb-NET* structures.

The search algorithm explores the search space defined by the *PN* reachability tree. The objective is to find a path that is a sequence of transitions that results in an optimal schedule. The *AI* community has studied several optimal systematic search algorithms in the last decades. From blind enumerative search methods such as *depth-first* and *breadth-first* search, to informed search methods such as *A** or *Branch & Bound* approaches (see [Pearl 84]). Before presenting the two algorithms adopted in the literature for *PN* based scheduling, *table 3* gives the basic notation of graph search and its relation to the *cb-NET* reachability graph.

Concept	Meaning	*cb-NET* translation
A node in the search tree	Data structure representing the state of the system. The root node is the initial state. Goal states are leaf nodes.	*cb-NET* marking. M_0 is the initial marking *(root node).* M_F is the final marking *(leaf node)*
Expand or Explore a node	Obtain descendants for a node. The node under exploration is usually referred to as the *current* node.	Obtain a *new* marking *M'* from a marking *M.*
A branch of the search tree	Part of the search tree reached by applying a specific operation.	A new marking *M'* obtained after the firing of a specific transition *t* in *M.*
depth(n): depth of a current node *n* also node level.	Depth of the node in the search graph in terms of number of operations applied from the root node.	Number of transitions applied from M_0 to the current marking.
g(n): cost of the node	Current value of the objective function for the node.	Makespan or elapsed time of *M*

Table 3: Graph search and its relation with PN reachability analysis.

4.3.1 Branch & Bound.

The principle of a *B&B* algorithm is a depth-first search where a discrimination function is generally employed to determine the next operation to apply. A second feature is the calculation of a lower bound or estimate of the minimum total cost that can be obtained if the search continues from the current branch (node). When the first solution is found, and a process of backtrack is started, each new node generated is compared with the candidate solution by means of such a lower bound. If the theoretical solution with minimum cost solution that can be obtained following this branch is greater that the actual cost of the candidate solution, the exploration of this branch is discarded.

The *B&B* algorithm has been applied to search in *PN* models of production processes with modifications in [Shen 92][Chen 94] [Lloyd 95] [Abdallah 98]. All this work deals with *JSS* descriptions in the sense that no alternative routes or operations are considered. [Shen 92] and [Chen 94] obtained a first solution following a *blind* strategy (the next operation to apply is selected randomly among the conflicting ones). The selection of the next transition to fire in [Lloyd 95] also seems to be randomly decided.

[Abdallah 98] employs a dispatching rule based on the shortest processing time (*SPT*) and least working remaining time (*LWRT*). The main disadvantage of a *B&B* approach is that it limits the application of *PN* based information to guide the search since decisions are restricted to the next transition to fire at the current node *(depth-first strategy)*. On the other hand, a standard depth-first methodology does not keep a record of previously explored similar markings (states) which is a problem as we will see.

4.3.2 Best-First *A**.

The *A** algorithm (see [Pearl 84] [Nilsson 82]) bases its strategy on reordering the nodes not yet expanded according to increasing values of a function $f: M \rightarrow R$. $f(M)$ representings an estimate of the minimum makespan of the schedule that we might get if we follow the path from M_0 to M towards M_F. By choosing the markings with a lower value of $f(M)$ we are heuristically guiding the search towards the most promising markings.

$f:M \rightarrow R$ is calculated from the following expression, $f(M) = g(M) + h(M)$. $g:M \rightarrow R$ is the cost associated with the current marking M (in terms of makespan, it is the elapsed time associated with M.

The key to the performace of *A** lies in $h:M \rightarrow R$. $h(M)$ is an heuristic estimation of the *actual* cost of the minimal cost path between node n and the goal node (over all possible paths from n to all possible goal nodes). Given a current marking M, $h(M)$ is trying to estimate the cost of the best possible sequence of transitions to be fired to achieve the final marking M_F.

Ideally, $h(M)$ should be a solution of a relaxation of the problem; which results in a easier problem to solve. Generally, these heuristics represent theoretical lower bounds, which may never be reached in the actual problem.

To ensure optimality, i.e, to ensure that the first solution found by *A** is an optimum solution to the problem, $h(M)$ must satisfy the following condition:

Definition 3: *A heuristic function h is said to be* ***admissible*** *if* $h(m) \leq h^*(m)$.

$h^*(m)$ represents the actual optimum cost from M to M_F . If $h(m)$ satisfies this property, the estimation of the makespan from the current marking m to the final marking m_f is always lower than the actual optimum makespan that can be obtained.

The case when $h(M) = h^*(M)$ represents the ideal situation but implies the problem is already solved. The admissibility of the heuristic functions proposed in the literature and the *PN* based heuristic function developed will be studied in detail in the next chapter.

The general *A** algorithm has been adapted to *PN* structures in [Zhang 90] [Zhang 92] [Yim 94] [Lee 94], [Yim 96], [Jeng 98], [Inaba 98] and [Sun 94]. *Fig.6* shows a first definition of *A** adapted to search within a *cb-NET* reachability graph.

The algorithm can be explained as follows: The initial marking starts the search space at the root node. The list *UnExplored* contains markings whose successors have not yet been determined ("new" markings). The *Explored* list contains already considered markings. At each iteration of the algorithm (1) the marking *M* yielding the lowest value of *f(M)* is chosen for exploration by removing it from the *UnExplored* list (2).

If *M* represents the goal marking M_F the path from M_0 to *M* is recovered and the algorithm terminates (3). If not *M* is added to the *Explored* list (4). For each enabled transition *t* in *M,* a new marking *M'* is obtained (5). To determine those enabled transitions, and the new marking, *A** makes use of the *cb-NET* simulator module.

The algorithm also identifies previously reached markings in terms of the distribution of tokens. When a new marking is *similar* to a previously considered marking and the new one represents a *more promising* path (7), it is still considered for exploration. If two *similar* markings are found in the *UnExplored* list (6), the one suggesting a *more promising* path is kept for further exploration and the other is discarded. It must be noted that for complex data structures this step can considerably increase the computation time of the algorithm. The application of decision heuristics at this level, by determining the interpretation of *similar* and *more promising*, will have great implications for the performance of the algorithm. Chapter 5 of this thesis considers *PN* based techniques to optimise the interpretation of these concepts for the *FMS* scheduling problem.

Algorithm *cb-NET A*:*

Receives: M_0, M_F: The initial and final marking of a *cb*-NET *N* from *FmsML*.

Returns: Sequence of transitions representing the schedule.

Variables: *UnExplored:* List of *new* markings for exploration.

Explored: List of markings already explored.

UnExplored = M_0

Explored = ∅

(1) while *UnExplored* is not empty **do** the following

(2) Remove a marking *M* from *UnExplored* yielding the smallest *f(M).*

(3) **If** *M* matches with M_F **then**

Retrieve the path from M_0 to M_F

Exit with success.

(4) **else** Put *M* in the *Explored* List

(5) **For** every transition *t* *enabled* in marking *M* do the following:

Obtain the marking *M'* as result of firing *t* in *M*

(6) **If** *M'* is *similar* to another marking *M"* in *UnExplored* **then**

If *M'* *is more promising than* *M"* then

Put *M'* in *UnExplored*

Remove *M"* from *UnExplored*

else goto (5)

(7) **If** *M'* is *similar* to another marking *M"* in *Explored* **then**

If *M'* *is more promising than* *M"* **then**

Put *M'* in *UnExplored*

else goto (5)

Put *M'* in *UnExplored*

goto (5)

goto(1)

Fig. 6: cb-PN based A algorithm.*

4.4. The NP-hard nature of the *FMS* scheduling problem.

Unfortunately, due to the NP-hard nature of the *FMS* scheduling problem it is not possible to obtain a polynomial scheduling algorithm that assures optimality as a

function of the parameters of the problem (parts, jobs, machines, degree of flexibility, etc).

Informed search approaches such as *A** guide the search towards the optimum solution using heuristic analysis of as yet uncompleted solutions. But the property of optimality and/or accuracy is double edged. Studies have shown that the *A** approach with an admissible heuristic function spends a large amount of time discriminating between sub-schedules whose makespan's do not vary significantly from each other [Pearl 84]. In such cases, the admissibility property of *h(M)* adds a *breadth-first* search component that prevents *A** from completing the search if the search space is large. The *AI* community has been aware of this problem: [Korf 85] proposes an alternate *A** algorithm called iterative deepening *A* (IDA*)* as a way to bound memory requirements at the cost of node re-expansion by a hybrid algorithm between *B&B and A**. However, [Sarkar 91] states that *IDA** performs poorly for problems such as the *JSS* scheduling problem which has real-valued cost estimates due to excessive node re-expansions.

The immediate implication of these results is that the vast majority of work in *FMS* scheduling considers the objective of optimality unrealistic. Additionally, the application of *PN* to manufacturing has typically been concerned with real-time control and more detailed system modelling [Zurawsky 94]. It thus seems natural to move towards the development of scheduling algorithms with application to dynamic real-time scheduling scenarios where the pursuit of optimality is ill-defined and unrealistic (not only unaffordable in terms of response time but the dynamic nature of the *FMS* environment implies many uncertainties which imply planning and scheduling are a continuous process). It is our belief that the *depth-first* search guided *B&B* algorithms are less applicable to these environments and limits the applicability of heuristic analysis on *PN* structures. As stated in [Korg 90], a related drawback is that *B&B* must search all the way to a solution before making a commitment to how good a node (path) is. Actually, *B&B* finds solutions quickly basing the optimisation procedure on chronological backtracking. In other words, before a possible alternate choice at depth k is considered, the complete search sub-tree from the current state of depth k to the goal marking needs to be explored.

In the literature surveyed, two general strategies have been followed to obtain affordable *PN* based *A*-like* search algorithms that guarantee a sub-optimal but acceptable solution within a reasonable time[6].

a) Adjusting the heuristic function.

In essence, the approach depends on finding a heuristic function *h(M)* that over-estimates *h*(M)* and whose difference *h(M) - h*(M)* reduces closer to the goal marking. If *h(M)* affects *f(M)* in such a way that *f(M)* decreases as *M* is deeper in the reachability tree, then the search is performed in a mixture of depth-first – best-first search. To achieve this, tuning parameters are needed.

The main problem with this approach is that: a) If *h* is unreliable it can lead to unpredictable solutions in terms of quality. b) Parameter tuning is needed and this can be difficult, and c) the search effort of the algorithm is difficult to control.

b) Hybrid search approaches.

If one cannot afford a full *A** algorithm, various hybrid combinations of strategies can be implemented, with the aim of reducing storage requirements at the expense of narrowing the evaluation scope of the algorithm (and optimality).

These algorithms control the amount of search by adjusting one or more parameters, allowing the computational cost to be bounded. This, theoretically, makes possible the use of admissible pure *PN* based heuristics applied to the bounded search space. Stage search [Nilsson 82] and Beam Search [Ow 88] are examples of incomplete search algorithms.

It is interesting to mention that - adjusting the heuristic function and *Incomplete A** - have the same effect: they obtain an acceptable sub-optimal solution within a reasonable and controllable search space. The combination of both methods is possible, and is a common practice in the literature.

---

[6] Recall what was said in chapter 2: the traditional approach that rejects optimal search algorithms and devises more or less complex heuristic dispatching rules might not be an advisable solution. As suggested in [Hutchinson 91] the benefits of *FMS* organisation may only be obtained with high-cost optimal (near-optimal) scheduling schemes rather than cheap heuristic ones. Recall also the *game-search* approaches summarised in chapter 2, and their conclusion regarding *short-term* accurate search and how this reinforces the idea of rejecting *depth-first B&B* based approaches.

Nevertheless, the aim of this research is the exploitation of *PN* information to guide search and thus to develop accurate and well informed *PN*-based heuristic functions. However, this may well lead to unaffordable search spaces. A useful algorithm that exploits the heuristic function available from *PN*, must thus control the complexity problem.

5. Case studies.

The two case studies that we present in this section, aim to briefly demonstrate the *PN* based modelling paradigm presented in this chapter

The first shows a small example that aims to capture *FmsML* capabilities and allows easy following of the *cb-NET* automatic synthesis process. The system is a small manufacturing layout consisting of two pressing machines and a robot. The example has been taken from the company *AEM,S.A.* [Aemsa]. *Fig. 11* shows the layout of the system.

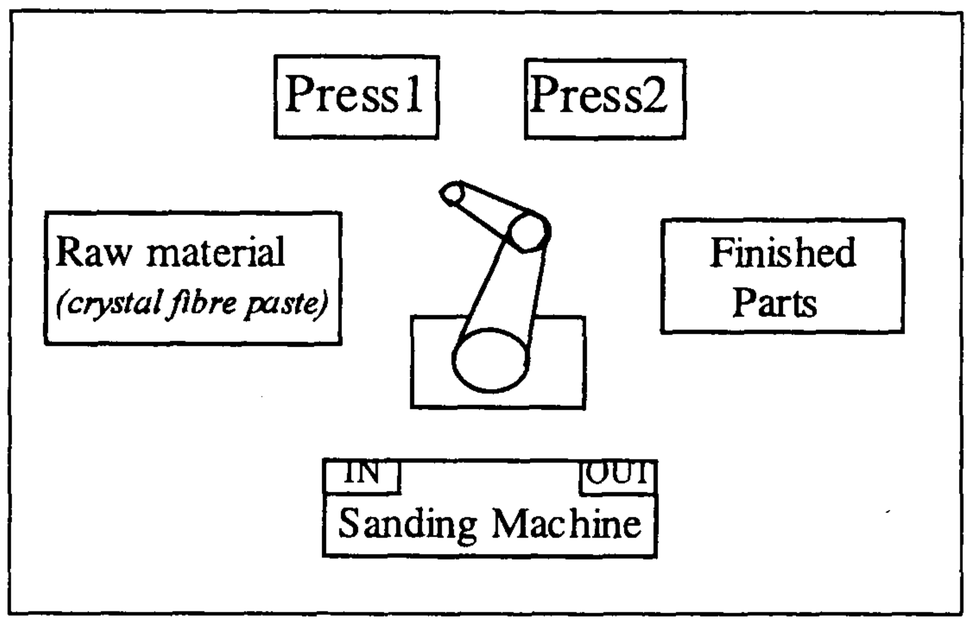


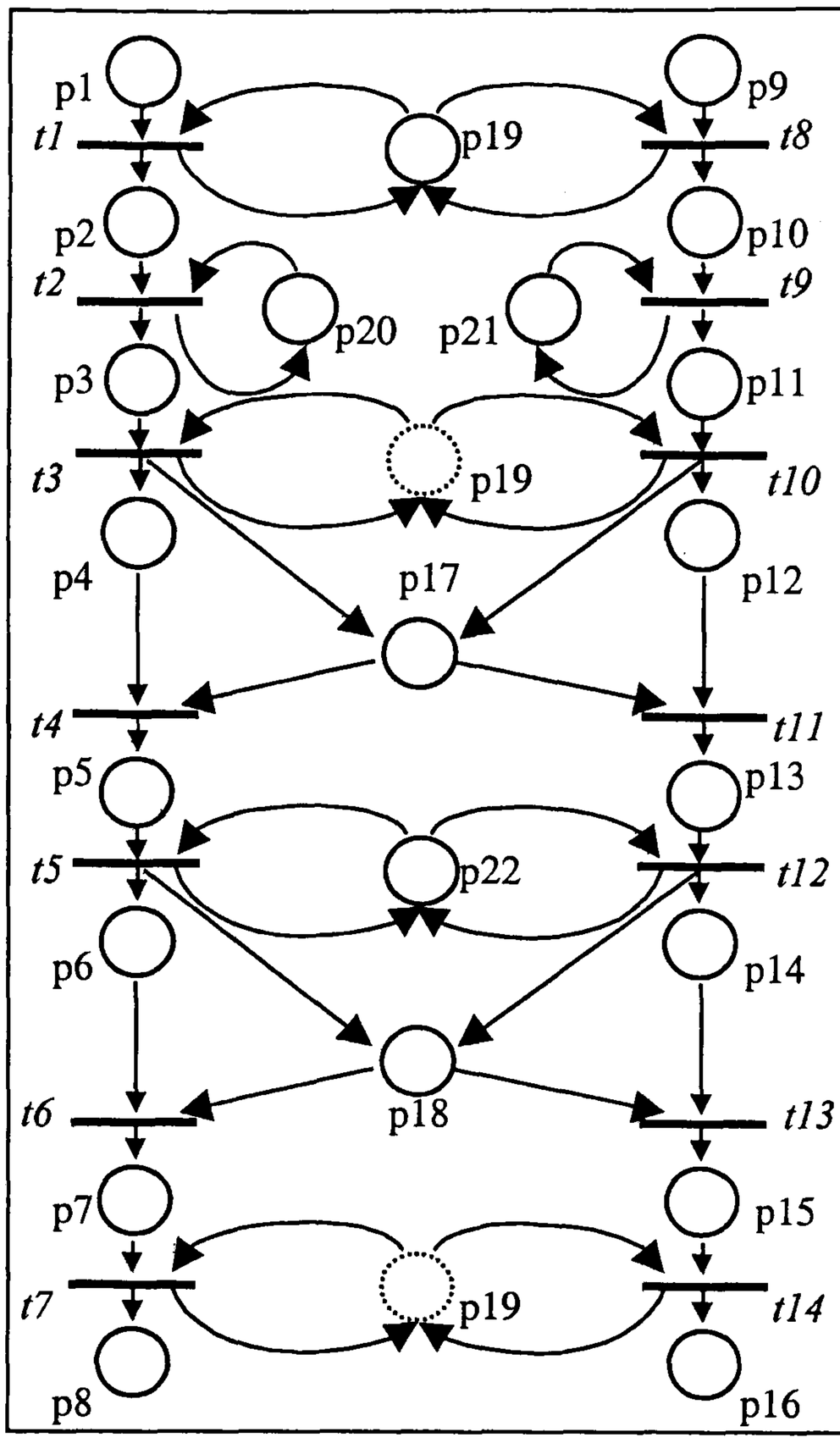
Fig. 11. System layout.

Two part types are produced in this system in two steps. First, the raw material (crystal fibre paste), is shaped by a pressing-heating machine, then a sanding machine removes excess material.

```
#system AEMSA;

#resource Robot #end;
#buffer Input
      capacity = 10;
      min_waiting = 1.0;
#end;
#buffer Output
      capacity = 10;
#end;
#resource Press1 #end;
#resource Press2 #end;
#resource Sander #end;

#job Part1
   #task Load1
      description= Robot:1.0;
      buffer = NO_WAIT;
   #end;
   #task Pressing1
      description= Press1:2.0;
      buffer = NO_WAIT;
   #end;
   #task Tranfer1
      description= Robot:1.0;
      buffer = Input;
   #end;
   #task Sanding1
      description= Sander:1.0;
      buffer = Output;
   #end;
   #task UnLoad1
      description= Robot:1.0;
   #end;

#end;

#job Part2
   #task Load2
      description= Robot:1.0;
      buffer = NO_WAIT;
   #end;
   #task Pressing2
      description= Press2:2.0;
      buffer = NO_WAIT;
   #end;
   #task Tranfer2
      description= Robot:1.0;
      buffer = Input;
   #end;
   #task Sanding2
      description= Sander:1.0;
      buffer = Output;
   #end;
   #task UnLoad2
      description= Robot:1.0;
   #end;

#end;
```

Fig. 12: FmsML code for the system example.

Product demand varies greatly from one day to another. A robot is in charge of loading and unloading the presses. The presses have no buffer since once the product is made it has to be removed immediately from the press to avoid deformation. On the other hand, raw material is kept from desiccation in a special initial buffer, so when the robot picks up raw material, it must be immediately processed. This means that there is no intermediate buffer among robot movements to/from the presses and that the robot cannot wait holding parts. This type of buffer policy is known as *zero-wait* [Lloyd 95b]. On the other hand, the sander has a input and an output buffers with a maximum capacity of 10. Since the material is hot and relatively soft after the pressing stage, cooling is needed before sanding, so it must wait in the buffer at least for one time unit. The cost of the pressing+heating is constant and equal to two units, sanding takes one time unit and pickup/release movements take one time unit. The movement of the robot when not holding parts takes negligible time. *Fig.12* shows the *FmsML* code for the system that results in the *cb-NET* model of figure *fig 13*.



p1 = Raw material for press 1 φ(+α ,0, +α)
p2 = Press1 input buffer: φ(1 ,0, 0)
p3 = Press1 output buffer: φ (1 ,0, 0)
p4 = Sander local input buffer 1 φ(+α ,1, +α)
p5 = Zero-wait buffer φ(1 ,0, 0)
p6 = Sander local output buffer 1 φ(+α ,0, +α)
p7 = Zero-wait buffer φ(1 ,0, 0)
p8 = Processed parts of type 1 φ(+α ,0, +α)
p9 = Raw material for press 2 φ(+α ,0, +α)
p10 = Press2 input buffer: φ(1 ,0, 0)
p11 = Press2 output buffer: φ (1 ,0, 0)
p12 = Sander local input buffer 2 φ(+α ,1, +α)
p13 = Zero-wait buffer φ(1 ,0, 0)
p14 = Sander local output buffer 2 φ(+α ,0, +α)
p15 = Zero-wait buffer φ(1 ,0, 0)
p16 = Processed parts of type 2 φ(+α ,0, +α)
p17 = Sander input buffer φ(10 ,0, +α)
p18 = Sander output buffer φ(10 ,0, +α)
p19 = Robot
p20 = Press1
p21 = Press2
p22 = Sander

t1, t3, t7, t8, t10, t14 = robot movements. $\tau = 1$
t2, t9 = presssing+heating. $\tau = 2$
t5, t12 = Sanding process. $\tau = 1$
t4, t6, t11, t13 = Buffer realease. $\tau = 0$

Fig. 13: cb-NET model of FMS example of fig 11/12 .

We have considered the problem of scheduling two parts of each type. The problem is solved optimally obtaining a makespan of 13 time units (*Fig. 14).* The solution obtained satisfies the buffer policies, i.e., once a raw material has been picked up, it is processed immediately. Also, when a press is finished, it is immediately emptied. Parts wait for at least one time unit in the sander input buffer before being processed. The key to this problem is a good schedule for robot movements (the bottleneck resource).

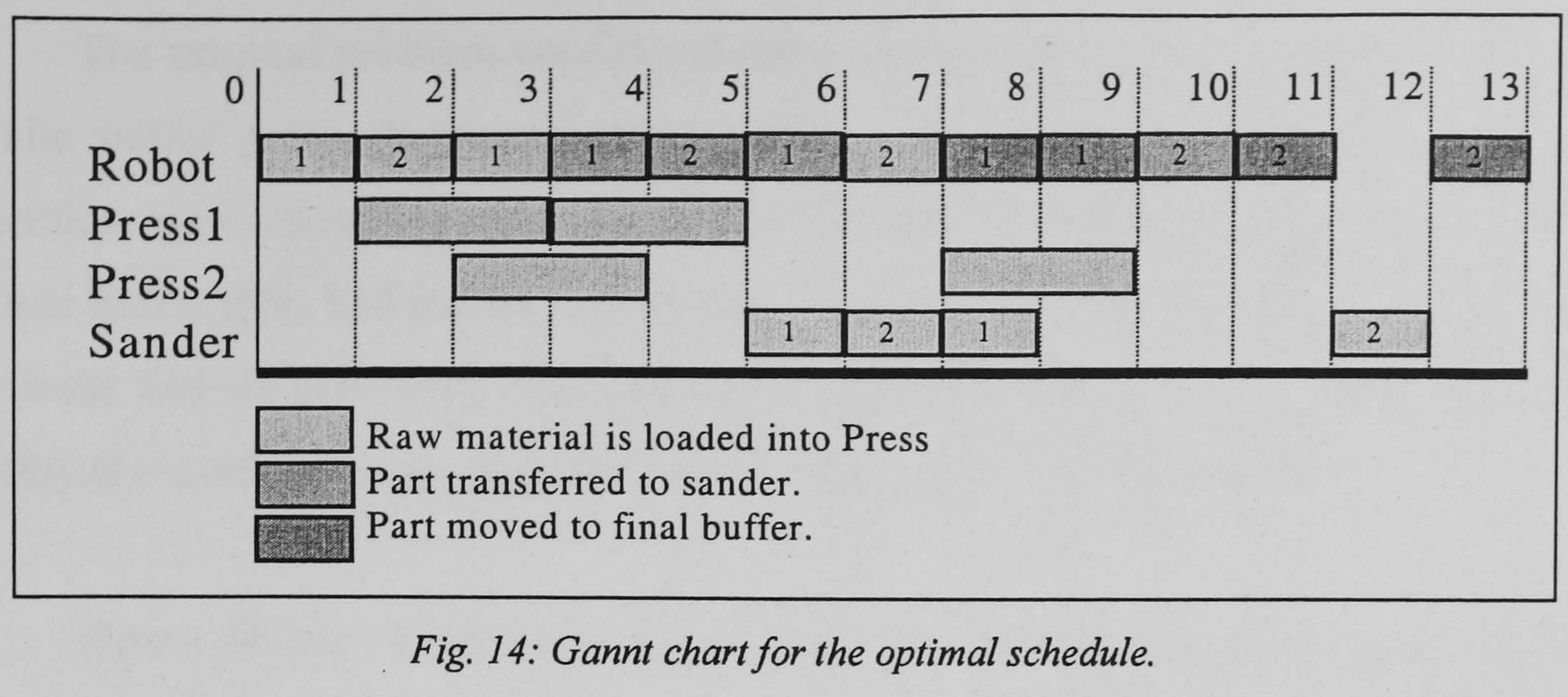


Fig. 14: Gannt chart for the optimal schedule.

The second example focuses more on how *FmsML* is employed to model academic case study problems that will form the test scenarios for the validation and comparison of the methods and algorithms presented this thesis. Additionally, it shows the modelling of alternate routes and different storage policies. The selected system is taken from a case study presented in [Lee 94] that consists of five different part-types. The *FmsML* code for the first job is given in *fig. 15* (the full specification for the problem is given in [Reyes 98]).

```
#system LEE_94;

// The FMS has three multipurpose machines....
#resource M1 #end;
#resource M2 #end;
#resource M3 #end;

// The FMS has five jobs.

#job Job_One
   #task T11 // The task can be achieved by either M1 or M3…
     description=M1: 7.0 | M3: 4.0;
   #end;
   #task T12
     description=M2: 3.0;
   #end;
   #task T13
     description=M1: 3.0 | M3: 6.0;
   #end;
   #task T14
     description=M1: 2.0 | M2: 4.0;
   #end;
#end;
```

Fig. 15: FmsML definition for the first job of [Lee 94] example.

The original problem considered ten parts for each job to be scheduled and an infinite buffer policy between any two tasks. For illustrative purposes of *cb-NET* dynamics, we studied the same problem with different storage policies [Lloyd 95]: a) Infinite buffer size, b) Limited storage size of one, c) No storage, but parts may wait in machines and d) zero-wait. The problem is too large to be solved optimally so we employed the sub-optimal algorithm *DWS** which presented in chapter 6.

Figure 16 shows the Gantt Chart of activity for each machine. It is noticeable that machine idle time increases as we constrain the buffer policy, as operations are delayed to guarantee system constraints.

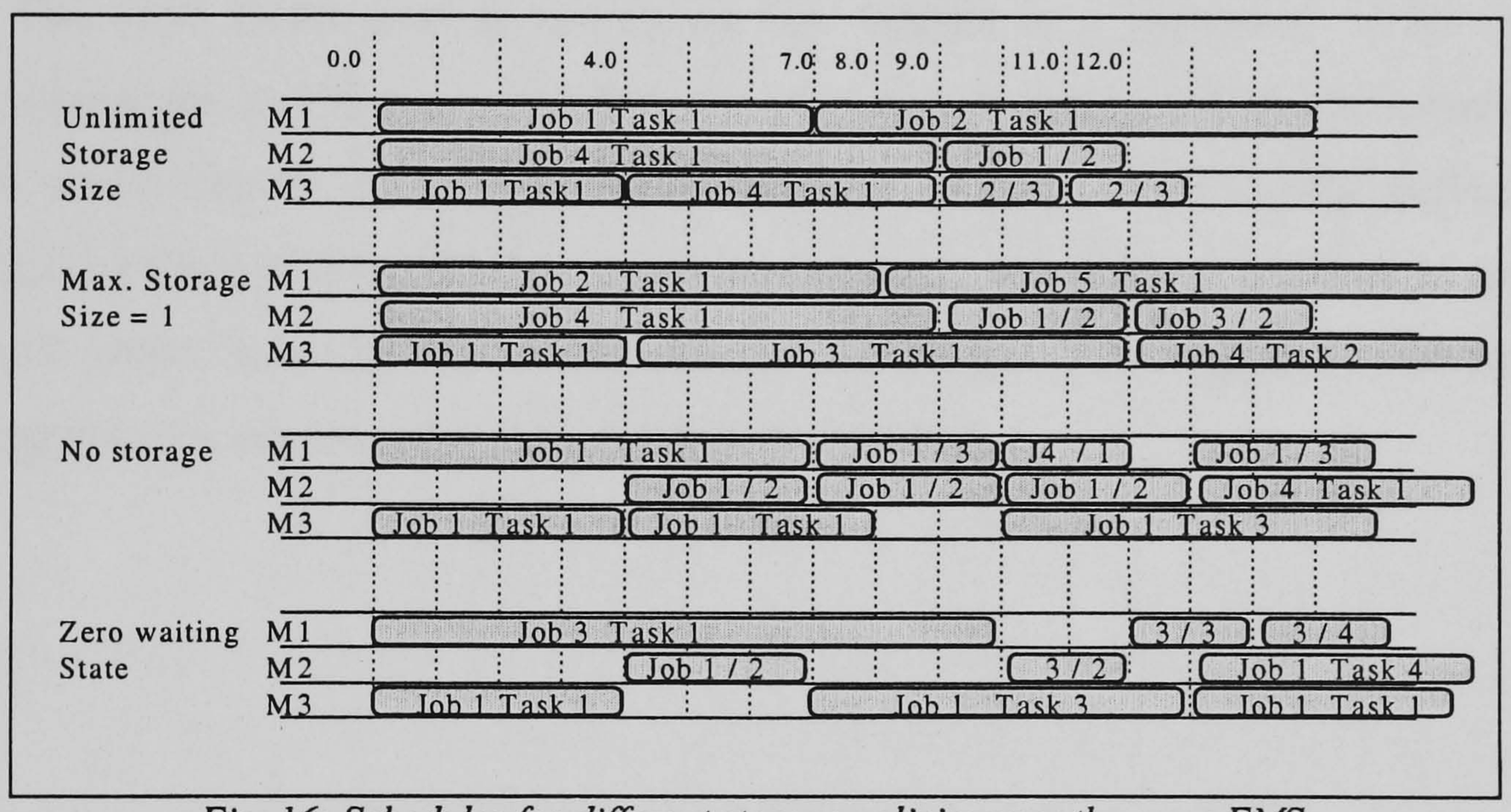


Fig. 16: Schedules for different storage policies over the same FMS.

Note that in the *zero-wait* case, when a part is processed it is transferred to the next stage immediately. Also, for the no storage-case, note that machine 3 is kept unavailable until the part of type one is processed by machine 2 at the next stage.

6. Summary and Conclusions.

In this chapter we have examined the *FMS* problem domain and presented a *PN* extension, *cb-NETS* that allows us to model *FMS* formulations and enables automatic synthesis process. The existence of bounded operation times is not observed frequently in *FMS* formulations from the literature. This is also the case with constraints associated with residence times in buffers. To the best of our knowledge, as [Lloyd 95b] affirms,

the modelling and consideration of the *no-wait* buffer policy is not found in work integrating *PN* and *AI* based heuristic search methodologies for *FMS* scheduling purposes.

A language for the definition of *FMS* specifications has been defined. This tool and the automated synthesis of *PN models* from these definitions has been motivated by: 1) the need for a systematic method that guarantees properties of the *PN* model needed for the scheduling process, 2) the need for a fast and reliable tool to obtain *PN* models of *FMS* case studies, 3) to serve as a preformatted input to generate random case studies to validate the *PN*-based scheduling algorithms.

Finally, we have reviewed the basic integration of *PN* theory and *AI* based search techniques and discussed the issue of complexity.

The work undertaken in this thesis has resulted in a collection of *PN* based hybrid algorithms and the use of *PN* based heuristic information to guide search and achieve useful search effort reduction. The next chapter will formally define the mathematical basis of this, and the algorithms to calculate it. Formal methods to reduce the search effort based on the *cb-PN* domain will be given in chapter 5. The hybrid search approaches developed will be defined in chapter 6.

Chapter 4. Heuristics from Petri nets.

1 Introduction.

This chapter is about the study of *PN* structures and dynamics to help the systematic search process within the *PN* reachability graph by means of a new *PN* based heuristic function.

One of the most interesting features of integrating *PN* theory and *AI* problem solving is the possibility of employing mathematical analysis over the problem domain that is a *PN* model. This possibility has been under exploited in previous work. This work typically either proposes a non-*PN* based heuristic, (which usually considers only limited information), or restricts the *FMS* problem domain. The heuristic function has normaly been typically employed to fight the combinatorial explosion, which masks the ability of *PN* to guide the search process, results in tuning difficulties and can lead to unexpected results. As our aim is a heuristic search algorithm, search effort reduction will be derived from the use of incomplete hybrid search algorithms that allow the application of admissible *PN* based heuristic functions.

The approach described here tries to give a theoretical lower bound for the minimum makespan. It is based on the idea of solving the problem by relaxing the resource constraints of the problem, both in terms of the use of machinery as well as buffer constraints. From this we have a problem which is only constrained by the operation sequence between jobs. With this assumption, jobs can always be done by following the path with the lowest blocking time. There is no blocking due to the availability of the resources, that is, every part is immediately processed.

In order to do so, a relaxation of the *cb-NET* model of an *FMS* needs to be obtained, which preserves those constraints related to part routing and neglects the others. Such a model defines a *PN* subclass that we have called *b-NETs*. *b-nets* are useful since they follow the same rules and perform close to the original *FMS cb-NET* scheduling formulation that we attempt to solve. Consequently good estimates of possible behaviours can be obtained. On the other hand, a *b-NET* has properties that allow the relaxed scheduling problem to be more easily solved. We will show that the optimum solution to such a relaxed problem is a good candidate as a heuristic function for the original problem.

The heuristic function developed and experimentally studied in this chapter has the following characteristics. It:

- is applicable to typical *FMS* descriptions.
- exploits *PN* based structures and dynamics.
- is admissible.
- considers both current status and global information.
- has a low computational cost without the need for mathematical relaxation.
- is very effective if the *FMS* is designed to exploit concurrency and avoid high machine idle time (bottleneck resources), which is typically a desirable design objective

The first section of the chapter reviews the efforts produced in this direction and identifies their difficulties. Section 3 formally defines *b-NETs* and explains their properties, *b-NETs* are a subclass of *PN* that satisfy structural properties which are needed to obtain a mathematical expression from where to easily obtain the relaxed solution of the problem. Section 4 and 5 describe the heuristic function obtained from a *b-NET* model of a *FMS* based on a matrix structure called *Resource Cost Reachability (RCR)*. This matrix will allow minimum calculation during the search process. Finally, section 6 presents experimental results.

2 Search reduction based on heuristic relaxation.

In the previous chapter, we stated that one of the strategies to be followed to progress to a first near-optimum solution was the relaxation of the property of admissibility of the heuristic function *h(m)*. If a *depth-first* search component is added to *h(m)*, thus making $f(m) = g(m) + h(m)$ to reflect a tendency to decrease as *depth(m)* increases within the path, the search will progress towards a first solution quicker than it otherwise might.

A simpler way to achieve this behaviour is by defining $h(m) = -w \bullet depth(m)$ which is the heuristic function developed in [Lee 94]. The rationale for this approach is: *the closer we are to the final marking, the closer we are to the optimum solution.* *w* is a

tuning parameter that specifies the degree of *depth-first* behaviour over the search. Clearly, this function is not admissible if $w>0$, for the general case.

The tuning of w is the most problematic issue to deal with. Consider a *FMS* scheduling problem defined by a *cb-NET*. Let M be the current marking under evaluation. Any marking M' which is a successor of M has an associated elapsed time $g(M')$ or makespan which will usually be greater than $g(M)$[7].

If $w=0$ then $f(M)=g(M)$ making A^* behave in a pure *Best-First* manner and consequently the next marking to explore is chosen according to the elapsed time of the marking. If $f(M)<f(M')$ then a *breadth-first* search behaviour is evident.

If we define $c(M,M')=f(M')-f(M)$ then for *depth first* search to occur $c(M,M')$ must be less than 0:

$$g(M') - g(M) - w \bullet depth(M') + w \bullet depth(M) < 0$$

$$g(M') - g(M) < w \bullet depth(M) + w - w \bullet depth(M)$$

$$g(M') - g(M) < w$$

In other words, w must be greater that the cost of obtaining M' from M for the general case. As a first approximation, one can use the following variation of $h(m)$

$$(1) \qquad h(m) = - OP \bullet depth(M)$$

where OP is the mean delay associated with operations. The work proposed in [Lee 94b] and later in [Yim 96] implements a similar approach by considering the heuristic function:

$$(2) \qquad h(m) = w' \bullet A \bullet E1 - w' \bullet A \bullet depth(m)$$

where A is the mean operation cost and E is the number of operations to schedule. In the case studies proposed any possible solution has exactly the same number of operations, hence the term $w' \bullet A \bullet E1$ is a constant that does not affect the selection process. Consequently, this expression can be reduced to

$$h(m) = - w' \bullet A \bullet depth(m)$$

Which is almost expression *(1)* except for w' which models machine concurrency. In fact, expression *(1)* does not take into account any concurrency between resources so, in the majority of cases, A is greater than $g(M')-g(M)$. If w' is set to the inverse of the number of independent machines, full concurrency is assumed. It must be

[7] $g(M)=g(M')$ if the transition fired at M was enabled, i.e, it had all its input tokens already available.

noted that the degree of concurrency between machines varies with each *FMS* formulation. As an attempt to more accurately predict the gap *g(M')-g(M),* a third heuristic function is proposed in [Yim 96]:

$$h(m) = w' \bullet A \bullet El - w' \bullet (A \pm sw \bullet S)$$

Which also takes into account the variance in operation cost. The sign of *sw* depends on the cost of the transition fired being greater than *A*. This is an incremental heuristic function in the sense that the new *h* value is added to the *h* of the current marking, and it follows that the term *depth(m)* is present in such a calculation.

The implementation and experimentation undertaken as part of this research with these heuristic functions showed a difficulty in tuning. For example, an experiment with randomly generated *FMS* problems obtained using a uniform distribution for the operation cost was generated. The setting of *w'=0.5,* meant that the algorithm did not finish in reasonable time in around 15% of problems, due to its *breadth-first* behaviour.

Two alternate heuristic function are also proposed in [Yim 96]:

a) $h(m) = - w \bullet MA_i$. MA_i is the mean operation cost for the operations performed in machine *i.* Again this is an additive function and *i* depends on the transition fired.

b) $h(m) = - w \bullet JA_i$. JA_i is the mean operation cost for the operations belonging to job *i.*

The main problem with these heuristics is that the authors do not offer a satisfactory explanation of their motivation. If *b)* was developed to take into account differences between jobs, they should have suggested an appropriate case study. The case of *a)* is less clear, it can be seen that *a)* may favour operations being performed on machines with greater mean cost than of others available ones. This is contradictory in terms of minimising makespan.

[Inaba 98] adapts the *A** search of [Lee 94] over *PN* models of *FMS* assembly systems, and proposes the following heuristic function which is an adaptation of (2)

$$h(m) = 1/N_M \bullet C_{min} \bullet r(m)$$

where C_{min} is the minimum assembly operation cost, and *r(m)* is the number of remaining assembly operations. N_M is the number of assembly machines. They are aware that as such a heuristic is admissible (and too optimistic), it leads to large search spaces. Hence, they propose $h(m) = 1/N_M \bullet C_a \bullet r(m)$, where C_a is the mean operation cost plus the mean of loading/unloading task.

The number of remaining assembly tasks *r(m)* is obtained from the number of tokens in the places indicating subassembly stages. Although the same effect as (2) is obtained, this work begin to show the possibilities inherent in analysing the current marking to obtain information.

The main criticism of a heuristic function that is based on the number of remaining operations is that it betrays the original idea of a heuristic function as an estimation of what is yet to be done. Two markings of the same depth have the same value of *h(m),* no matter the path followed. This leads to undesirable behaviours. For example, suppose two partial schedules represented by M_1 and M_2 have both scheduled exactly the same number of operations. *M1* has followed the strategy in which the fastest operations have been done first, achieving a low total operation cost but with low machine parallelism. On the other hand *M2* has achieved a quasi-full machine concurrency, the total operation cost is higher as a result of not just choosing the fastest operations. Suppose that the makespan *g(M1)* is almost *g(M2).* Notice that in *M1* the longest operations still need to be scheduled, which suggests that M_1 is less promising than *M2.* However since *depth(M1) = depth(M2)* both markings are heuristically equal.

Situations of this kind lead to unforeseen and often poor results. For example, a problem with 5 different jobs, 3 machines and ten parts per job is solved in [Lee 94] and the best makespan obtained is 426. However a simple splitting up approach obtains a solution of 370[8].

A heuristic function that is common in the literature based on *PN* reachability tree search is the following [Xiong 98]:

$$h(m) = max_i\{\xi_i(m)\ ,\ i=1,2,...N\}$$

[8] The approach solves a single part job problem optimally and then replicates the result ten times.

Where $\xi_i(m)$ is the sum of operation times of the remaining operations for all jobs which remain to be processed on the *i-th* machine for the current system state represented by marking *m*. *N* is the total number of machines. By choosing the *i-th* machine that maximises this expression, one is supposing that no idle time is produced for the machine, and the other machines finish at least at the same time as the i_{th} machine.

Although the heuristic function is admissible, it is not directly applicable to *FMS* since processing tasks are not assigned a priori to machines due to routing flexibility. Such an admissible heuristic function is employed to node selection pruning in the *B&B* approaches implemented in [Shen 92] [Chen 94] [Lloyd 95].

An adaptation of such heuristics to *FMS* scenarios is proposed in [Jeng 98b]. The case studies considered restrict the number of parts to be processed to a single part per job.

$$(3) \qquad h(M) = \max_{p \in PI} (M(p) \bullet \min_{l \in L(p)} (C(l)))$$

This heuristic is estimating the cost of reaching the final marking by calculating the minimum cost path among those possible for each of the remaining subparts. Although it is not completely clear, the cost of the path *C(l)* appears to be obtained from the sum of the minimum delay of transitions that belong to the paths. Intuitively, this is an admissible heuristic function, but it is only valid if a single part per job is considered. Although replicating *PN* structures for these jobs may solve this problem, this leads to huge *PN* models. In addition, when many parts compete for the use of resources, the assumption that the rest of the parts are done concurrently with the longest part, becomes overly optimistic.

[Jeng 98] and [Jeng 99] propose an alternative approach based on the *PN* state equation [Murata 89] that relates two markings with a firing sequence of transitions expressed as the array *u* and the incidence matrix *C*:

$$M_F = M + C \bullet u$$

It is well known that, if there exists a firing sequence from a current marking *M* to a final marking M_F, then the sequence expressed as *u* is a solution to the corresponding state equation [Murata 89]. Although the converse is not necessarily true, the solved firing count vector can serve as an estimate of the current makespan. If we

multiply each element of the vector by a delay time corresponding to each transition. Taking the sum of this seems to be a good candidate for the estimated cost[9]. This is derived in [Jeng 99] as follows:

$$(4) \qquad \text{minimise } \Sigma\, cost(t_j) \bullet u_j \quad j=1 \ldots n$$

$$\text{Subject to } C \bullet u = \Delta M$$

$$\text{and } u_j \in (N^+ \cup \{0\})$$

The problem is that the above is an integer programming problem which is typically expensive to solve. Consequently [Jeng 98] proposes an alternate linear system that is easy to solve and avoids excessive matrix computation for each marking generated during the search. This heuristic function is not admissible and adds a *depth-first* component to the search, since *g(m)* tend to increase less than *h(m)* decreases. Such behaviour is produced since *h(m)* is an estimation of the total operation cost and does not take into account concurrence between resources.

Combinations of (3) and (4) are possible. For example, [Jeng 98] fixes a depth-bound for the application of (4). The search is controlled as follows: (4) is used until a given depth is reached. The motivation for this seems to be ensuring that at least a few operations of each job are scheduled. This is due to the ability of (4) to differentiate between alternate routes better than (3). Once the first decisions are made, (3) is used for the rest of the search.

3 Buffer nets: *b-NETs.*

As described above, the heuristic function developed is based on the idea of relaxing two constraints of the system: the limited amount of resource and the time constraints imposed on materials. With these assumptions jobs can always be performed by following the fastest routes and operations are not delayed by constraints imposed on buffers or materials. As a result, maximum machine parallelism can be assumed. To achieve such a relaxed system as a *PN* model, the original *cb-NET* model of a *FMS* needs to be transformed into a simpler one, where no constraints are assigned to places

[9] Actually, [Jeng 99] proposes a place-timed *PN* model, so the operation cost of the transition is defined to be the maximum time associated with its input places.

and hence, an infinite buffer policy can always be applied. We will call such a net a *buffer NET* or *b-NET*.

A *b-NET* is technically a net subclass. A net subclass is defined exclusively by introducing constraints on the structure of ordinary nets [Silva 89]. In general, the aim is to restrict the generality of the model, allowing us to characterise fully otherwise hard to study properties such as liveness and reversibility. The definition of net subclasses is a common practice in the *PN* literature. For example, *CO-nets* [Proth 97] and *ECO-nets* [Wang 96] where introduced with the aim of defining a subclass of *PN* for manufacturing system integration.

In our case, it is from a *b-NET* that the heuristic function will be obtained. *b-NET's* are convinient since their structural properties enable a mathematical expression which results in *PN* based heuristic function with is cheap to calculate.

3.1 *b-NET* definitions and properties.

Definition 4: A timed-*PN* (P,T,I,O,M_0) is called a *b-NET* if

$$(P = R \cup Q) \wedge (R \cap Q = \varnothing).$$

where the set of places R represents the resources and the set of places Q represents the buffers, and the following three conditions are also satisfied:

(a) $I(r,t)=O(r,t)\ \forall t,r\ \ t\in T,\ r\in R$

(b) $\forall\ t\in T$, there exists a single $p\in Q : I(p,t)=1$ *and* a single $p'\in Q : O(p',t)=1$, for $p\neq p'$;

(c) The subnet $G'= (Q, T, I', O', M', h)$, where I' and O' are the restrictions of I to $(Q \times T)$ and O to $(T \times Q)$ respectively and M' is the restriction of M to P, is an acyclic graph without isolated places.

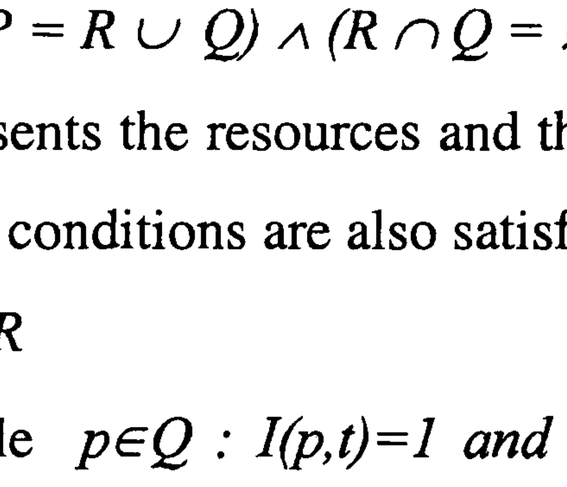


Fig. 17: A single-transition scheme constrained by (a) and (b).

Condition *(a)* ensures that all the resources *used* by a transition are released after the firing of the transition. Condition *(b)* indicates that each transition has an input (output) *buffer;* and only one input (output) *buffer* is allowed for each transition. Condition *(c)* ensures that no cycles are introduced.

It can be seen that *b-NET*s have two types of places: resource-places and buffer-places. Time is associated with transitions. From a structural point of view, a *b-NET* is any ordinary deterministic timed *PN* that can be formed using only the single-transition structure shown in *Fig.17*.

Definition 5: ***Initial and final state (markings) of a b-NET.***

For a *b-NET N*, $B_I \subset Q$ (the Input buffer set) contains all the places of Q which are not output places of any transition, i.e., $B_I=\{p\in Q \mid \neg\exists\, t \in T \quad O(p,t)>0\}$; $B_o \subset Q$ (Output buffers set) contains all the places of Q which are not input places of any transition i.e., $B_O=\{p\in Q \mid (not\ \exists)\ t \in T\ I(p,t)>0\}$. A state M_0 is an initial state (marking) for a *b-NET* if $(M(p) = 0\ \forall p\notin(B_I \cup R)) \wedge (M(p)\geq 1\ (\forall p\in R)) \wedge (\exists p\in B_I \mid M(p) \geq 1)$. A state M_F is a goal (final) state (marking) for a *b-NET* if $(M(p) = 0\ \forall p\notin(B_o\cup R)) \wedge (M(p)\geq 1\ (\forall p\in R)) \wedge (\exists p\in B_o \mid M[p] \geq 1)$.

Note that $B_I\cap B_O=0$. Two interesting properties can be observed in *b-NETs:* they are *practically* live and bounded, whatever the initial marking may be.

A *PN* is said to be live if, no matter the marking which has been reached, it is possible to ultimately fire any transition of the net by progressing through some further firing sequence. It is obvious that the definition of *final* state of a *cb-PN* does not satisfy this property, however:

Property 1: A *b-NET* is live for all $M\neq M_F$.

The proof is straightforward from *fig. 17* and the definition of a *b-NET*. It is always possible to perform any of the operations for which the system was designed whatever the decisions made in the past. This is the *practical* meaning of liveness: there always exist enabled transitions at any marking *m* that is not a final marking.

Property 2: Given an initial bounded marking M_0 the number of tokens for any marking reachable from M_0 is constant and equal to the initial number of tokens.

Again, the proof is straightforward. The *b-NET* definition ensures that any transition releases as many tokens as it consumes, i.e., no tokens are created or consumed i.e., $\forall t \in T;\ \sum I(p,t) = \sum O(p,t)\ \ \forall p \in P$, hence a *b-NET* is bounded and the number of tokens remains constant.

The following proposition follows from these properties:

Proposition 1: Given an initial bounded marking M_0 of a *b-NET* it is possible obtain a sequence of transitions that leads to any final state (marking) M_F the number of tokens in input buffers are the same as in output buffers.

Proof: Proof is straightforward from properties 1 and 2.

3.2 *b-NET* generation from *FMS* definitions.

We use the *b-NET* to model *FMS* having assumed that only an infinite buffer policy can be applied. It is worth noting that the structure of *fig. 17* corresponds to the *cb-NET* basic module presented in the previous chapter. In fact it is possible to obtain an infinite buffer *sub-net* from any of the basic modules if one ignores the constraints φ imposed on *buffer* places in a *cb-NET*. As an example, *fig. 18* shows how the *cb-NET* model of a limited shared buffer is reduced to a *b-NET* (the other basic modules follow a similar approach).

The first step removes the constraints imposed on the common buffer *b* and the *zero-wait (z_w)* buffer. This results in an unconstrained *cb-NET* with $\varphi(b)=(+\infty,0,+\infty)$ and $\varphi(z\_w)=(+\infty,0,+\infty)$. The first reduction step merges *b* with *b* local. The resulting net contains a transition that removes a token from an infinite buffer *b* and puts a token into an infinite buffer *z_w*, which has a firing delay of zero and uses no resources. Hence, the *b-NET* obtained after eliminating t_{end} and *z_w* is equivalent to the previous one from the point of view of scheduling.

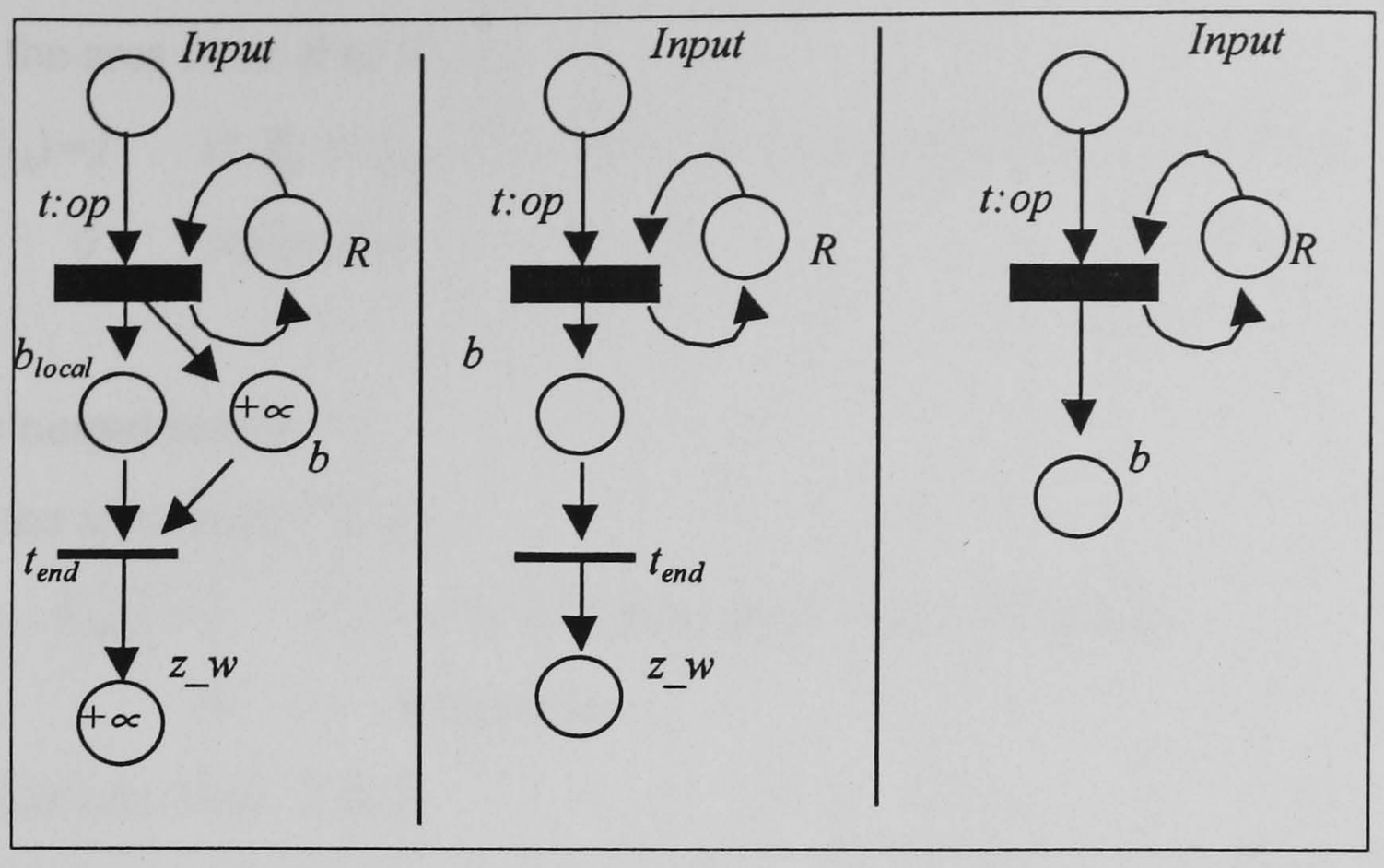


Fig. 18: Reduction of an unconstrained cb-NET to its equivalent b-NET

Although such a reduction procedure indicates that it is always possible to obtain a *b-NET* from a *FmsML* definition, by ignoring the constraints imposed on buffers, the following construction procedure formalises the methodology.

Let $N=(R,Q,T,I,O,M,\tau)$ be a timed-transition *PN* where initially R = ∅ , Q= ∅ and T= ∅. The following are the steps for the construction of a *b-NET* for a *FMS* formulation expressed in a *FmsML* that follows the formulation given in definition 1.

1. Create the set of resource places: Add the *#resource* construction R_i ($R = R \cup \{R_i\}$) for i=1 to m, where *m* is the number of resources.
2. Create the set of buffer places:
   2.1. Add the initial input buffer of the i^{th} *#job* J_i to the buffer set Q ($Q = Q \cup \{J_{i0}\}$) for i=[1 .. n], where *n* is the number of jobs.
   2.2. Add the buffer place for *#task* T_{ijk} of *#plan* P_{ij} for the *#job* J_i , $Q = Q \cup \{J_{ijk}\}$ for i=[1…n], j=[1… p_i] and k =[1…q_{ij}] where p_i is the number of different plans for job J_i and q_{ij} is the number of tasks for plan P_{ij} of job J_i.
3. Create the set of transitions: Adding l^{th} choice defined in the *description* of *task,* T_{ijk} of *#plan* P_{ij} for *#job*$_i$ T = T∪{t_{ijkl}} for i=[1…n], j=[1… p_i], k =[1…q_{ij}] and l=[1…C_{ijk}], where C_{ijk} is the number of choices for T_{ijk}.
4. Create the input matrix:

   a) Define the arcs from *Q* to *T*

$$I(J_{ijk}, t_{ijk+1,l}) = 1 \quad 1 \le i \le n;\ 1 \le j \le p_{i-1};\ 0 \le k < q_{ij};\ 1 \le l \le C_{ijk}$$

$$0 \qquad \text{otherwise}$$

b) Define the arcs from *R* to *T*.

$I(R_i,J_{ijk})=1$ if $R_I \in S_{ijk}$. . $1 \le i \le n;\ 1 \le j \le p_i;\ 0 \le k < q_{ij}\ ;1 \le l \le C_{ijk}$

0 otherwise.

5. Create the output matrix:

a) Define the arcs from *T* to *Q*

$O\ (J_{ijk}\ ,\ T_{ijkl}) = 1$ $1 \le i \le n;\ 1 \le j \le p_i;\ 0 \le k < q_{ij}\ ;1 \le l \le C_{ijk}$

0 otherwise.

b) Define the arcs from *T* to *R*.

$O(R_i,J_{ijk})=1$ if $R_i \in S_{ijk}$, $1 \le i \le n;\ 1 \le j \le p_i;\ 0 \le k < q_{ij}\ ;1 \le l \le C_{ijk}$

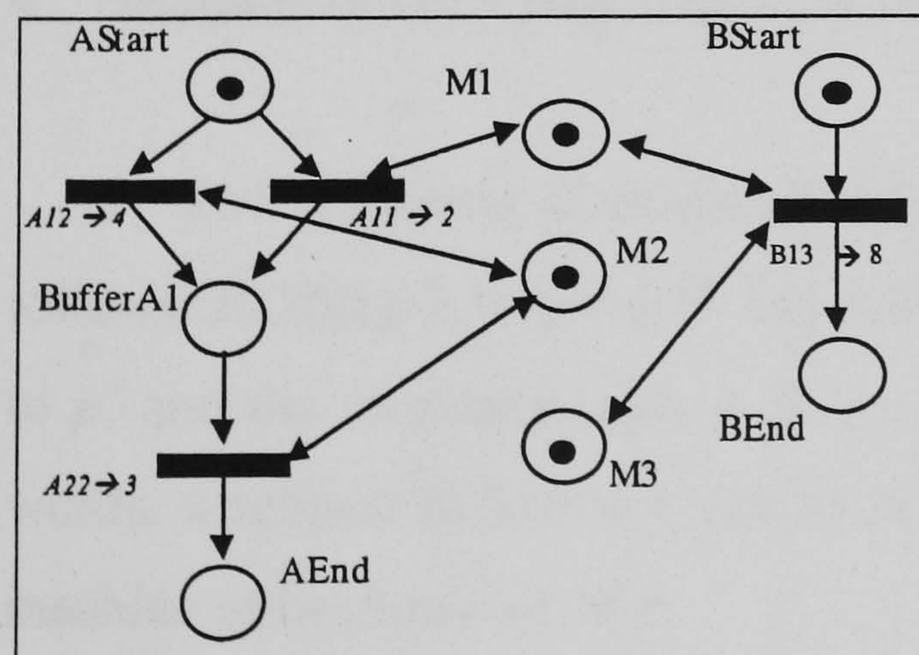
0 otherwise.

6. Create the time vector: $\tau\,(t_{ijkl}) = h_{ijkk}.\ 1 \le i \le n;\ 1 \le j \le p_i;\ 0 \le k < q_{ij}\ ;1 \le l \le C_{ijk}$

It can be easily demonstrated that the *PN* generated by following the above procedure is a *b-NET*. *Fig.19* shows the *b-NET* for a simple *FmsML* definition. The following equivalencies have been used:

- A totally unprocessed part for job $i \equiv J_{i0} \in Q \equiv J_{iStart} \equiv Ji0$
- A partial processed part at task *j* of job *i*, $buffer_{ij} \equiv J_{ij} \in Q.\ 1 \le i \le n\ ;\ 0 \le j \le p_{i-1}$ with *pi* the number of tasks for J_i.
- A completed part for Job $i \equiv J_{ipi} \in Q.\ J_{iEnd} \equiv Jipi$ with p_i the number of tasks for J_i.

Fig. 19 shows the *b-NET* model of a simple *FMS*.




```
#system FMS;

#resource M1 #end;
#resource M2 #end;
#resource M3 #end;

#job Job_A
  #task A1 description=M1: 2.0 | M2: 4.0; #end;
  #task A2 description=M2: 3.0; #end;
#end;

#job Job_B
  #task  B1 description= M1 & M3: 8.0; #end;
#end;
```

Fig. 19: b-Net model of a simple FMS.

In a way, a *b-NET* represents a relaxation of a *FMS* model. No constraints are imposed on the use of buffers. Hence, the solution to the scheduling problem defined by the *b-NET* is a lower bound on the solution to the original problem defined by the *cb-NET*. The difficulty is that solving a *b-NET* problem is as difficult as solving the whole problem itself.

Ideally, *h(M)* should be a solution of a relaxation of the problem; which results in an easier problem to solve. A *b-NET* guarantees that operations are not delayed by constraints imposed on buffers or materials. However, we need to go further by considering an unlimited resources. This will guarantee that jobs can always be done following the fastest routes and that they are not delayed by the sharing of resources. As a result, maximum machine parallelism can be assumed. Conceptually, this relaxation is introduced into the *PN* model by eliminating those places representing resources. This results in a *PN* that represents the alternate paths that a product may follow. The reduced model corresponds to the routing (processing) circuit that can be found in some modelling approaches [Hillion 98]. *PN* theory provides mathematical analysis for the solving of such a relaxed problem. However, as we have reviewed in section 2 of this chapter, this may well lead to expensive matrix calculations that require relaxation of the original heuristic function [Jeng 99]. Being aware of this, our approach bases its methodology in the calculation a priori of a matrix called the ***Resource Cost Reachability*** (***RCR***) matrix, which is obtained once the *b-NET* model is given. The *RCR* matrix will later be employed to calculate the heuristic function for every marking using a cheap algorithm.

4 Resource Cost Reachability Matrix *(RCR)*.

The ***Resource Cost Reachability*** (***RCR***) matrix can informally be defined as follows: $RCR(p,p') = r$; $r \in R^+$ indicates that it is possible to *move* a token from place p to p' and the minimum path in terms of resource utilisation has a cost of r. In other words, a subpart in buffer p can be processed and transferred to buffer p' with a total machine utilisation cost of r.

For example, a preliminary *RCR* matrix for the *FMS* in *fig. 19* is given in *table 4*:

	Astart	BufferA1	Aend	Bstart	Bend
Astart	0.0	+ ∝	+ ∝	+ ∝	+ ∝
BufferA1	2.0	0.0	+ ∝	+ ∝	+ ∝
Aend	5.0	3.0	0.0	+ ∝	+ ∝
Bstart	+ ∝	+ ∝	+ ∝	0.0	+ ∝
Bend	+ ∝	+ ∝	+ ∝	16.0	0.0

Table 4: pre-RCR for FMS of fig. 19.

RCR(Aend,BufferA1) represents the minimum machine utilisation for processing a subpart from task 1 to task 2 of job A. The minimum machine utilisation for processing job A is *RCR(Aend,Astart)* = 5.

RCR(Bend,Bstart) is 16.0 because although the operation cost is 8.0 it needs *M1* and *M3*. If *RCR(P,P')* = + ∝, it means that there is no relationship between two tasks. In fact, *RCR(P,P')* stands for the optimum path to produce a single part in the scheduling problem (in terms of machine utilisation) assuming that only a single subpart is being processed. It can be seen that, the *RCR* is a concept related to *b-NET's* and not to the *FMS* itself.

4.1 Formal definition of the *RCR* Matrix.

Definition 6: Transition resource number (Trn). *Trn(t)* returns the number of resources *used* by a transition *t*.

$Trn: T \Rightarrow N^{+} \cup \{0\};$

$Trn(t) = \Sigma_r I(t,r) \ \forall r; r \in R$.[10]

Definition 7: Resource time (Rt) of a transition. *Rt(t)* computes the total operation time modelled by transition *t*.

$Rt: T \Rightarrow R^{+} \cup \{0\}$;

$Rt(t) = Trn(t) \cdot \tau(t)$; with $\tau(t)$ the delay time associated with the transition.

[10] Recall that in the definition of a *b-net* the expression $Trn(t) = \Sigma_r I(t,r) \ \forall r; r \in R$ also holds, so $Tsm(t) = \Sigma O(t,r) \ \forall r \ ; r \in R$

Definition 8: *t* is said to be directly connected with *t'* where $\forall t,t' \in T$, if $\exists p \in P$ such that $I(p,t')>0 \wedge O(p,t) >0$ i.e., if there is a place that is an output for *t* and an input for *t'*.

Definition 9: *t* is said to be connected with *t'* $\forall t,t' \in T$, if $\exists$ *s*= *t*, t_0, t_1, ... ,t_n, *t'* ($n \geq 0$) where *s* is an ordered sequence of transitions such that:

i) *t* is directly connected with *t0*

ii) t_i is directly connected with t_{i+1}; $0 \leq i < n$

iii) t_n is directly connected with *t'*

s is also said to be a *connected sequence*.

Definition 10: Resource time of a connected sequence (Rs).

$Rs: T^n \Rightarrow R^+ \cup \{0\}$;

$Rs: (s) = \Sigma_{t \in s} Rt(t)$; *s* is a connected sequence.

Definition 11: *RCR* Matrix.

$RCR: Q \times Q \Rightarrow R^+ \cup \{0\}$

$RCR(p,p') = 0$ if $p=p'$

$\min\{+\infty, Rs(s)\}$; *s* is any connected sequence $\{t_0, t_1, ... t_n\}$.

where *p'* is an input place of t_0

and *p* is an output place of t_n.

Table 5 shows the *RCR* matrix for the *PN* in *fig. 19*, where:

$$RCR(AEnd, AInit) = \min\{+\infty, Rs(A12,A21), Rs(A11,A22)) =$$

$$\{+\infty, (4+3)\ (2+3)\} = 5;$$

$$RCR(BEnd, BInit) = \min\{+\infty, Rs(B13)\} =$$

$$\min\{+\infty, 2 * \tau(B11)\} = \{+\infty, 16\} = 16.$$

This means that to produce parts A and B we need a minimum utilisation of machines of *5* and *16* time units respectively.

	A0	A1	A2	B0	B1
A0	0.0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
A1	2.0	0.0	$+\infty$	$+\infty$	$+\infty$
A2	5.0	3.0	0.0	$+\infty$	$+\infty$
B0	$+\infty$	$+\infty$	$+\infty$	0.0	$+\infty$
B1	$+\infty$	$+\infty$	$+\infty$	16.0	0.0

Table 5: RCR matrix.

To summarise, for a *b-NET* obtained by parsing a *FmsML* description, if for J_{ij}, $J_{ij'}$ $(j'>j)$ included in Q, the element of the *RCR* matrix $RCR(J_{ij'}, J_{ij}) \neq +\infty$ is the minimum utilisation time of resources required for a token in J_{ij} to reach $J_{ij'}$.

4.2 An algorithm for the computation of the *RCR* matrix.

The algorithm shown in *fig. 20* computes the *RCR* matrix. We claim that, for a *b-NET* generated from a *FmsML* description, the *RCR* matrix can be constructed in polynomial time. Prior to entering the *while* loop, the algorithm only determines *RCR[p, p']* if p and p' are directly interconnected. The second part of the algorithm tries to find indirect connections between p and p''. If the pairs *[p, p']* and *[p', p'']* are both connected, then the *buffer-places* p and p'' are connected and the minimum resource time of the sequence is assigned to *RCR[p,p'']*. Every time a new value for *RCR[p,p'']* is found, we need to repeat the cycle (this is controlled by the variable *Change*). Since the creation order of the *buffer-places* during the *b-NET* generation ensures that J_{ij} is always defined before J_{ij+1}, the algorithm will only need one iteration to calculate the rest of the *RCR* matrix. It follows that, for a *b-NET* created by parsing an *FmsML* specification, the computational cost of obtaining the *RCR* matrix is $O(|Q|^2 \bullet |T| + |Q|^3)$.

Function Compute_RCR:

Receives: *N*: A *B-net* obtained from *FmsML*.

Returns: *RCR* matrix for *N*

Algorithm:

$RCR\,[p,p'] = +\infty \quad \forall p,p' \in Q \ \wedge p \neq p'$

$RCR\,[p,p] = 0 \quad \forall p \in Q$

$\forall p, p' \in Q$ and $\forall t \in T$

if $I(p',t) = 1 \ \wedge O(p,t) = 1$ **then**

$RCR[p,p'] = \min (RCR[p,p'] , Rt(t))$

Changes = TRUE

while *Changes*

Changes =FALSE

$\forall p,p',p'' \in Q$

if $(RCR[p,p''] > RCR\,[p,p'] + RCR[p',p''])$ **then**

$RCR[p,p''] = RCR\,[p,p'] + RCR[p',p'']$

Changes = TRUE

return *RCR*

Fig. 20: RCR computation algorithm.

The following section will show how to use the *RCR* matrix to obtain a heuristic function that represents a lower bound on the scheduling problem represented by the *b-NET* and thus the *cb-NET*.

5 Generating a heuristic function from the *RCR* matrix.

By definition 11 $RCR[J_i end, J_i start]$ represents the minimum machine utilisation for processing a product of job type Job_I, where $J_i start$ *and* $J_i end$ represent the initial and output job buffers respectively. Expression (1) represents the minimum total operation cost for processing a product of each job type where n is the number of jobs and p_i is the number of parts to produce each job.

$$(1) \qquad \sum (p_i \bullet RCR\,[\,J_i\text{end}, J_i\text{start}\,]); \textit{ for } i=1 \textit{ to } n.$$

By considering that the concurrence between resources is maximum (i.e. there is no conflict for the use of resource so each part does not need to wait to be processed) we can obtain (2) where *m* is the number of resources. Note that (2) is a lower bound of the minimum makespan or total completion time for the problem of ,for example, scheduling one product of each job.

$$(2)\ \Sigma (p_i \bullet RCR [J_i\text{end}, J_i\text{start}]) / m;\ \textit{for } i=1 \textit{ to } n$$

For example, for the *FMS* of *fig. 19* this theoretical lower bound is:

$$(RCR[J_A\text{end}, J_A\text{start}] + RCR[J_B\text{end}, J_B\text{start}]) / 3=$$

$$(5 + 16) /3 = 7$$

On the other hand, the optimum makespan for this problem is *8*.

A further step is to generalise expression *(2)* so that it computes a lower bound of the optimal makespan from any intermediate state of the *b-NET* to the goal marking M_F. The problem is thus defined as:

Given M *any state of a b-NET that represents the relaxed scheduling problem and* M_F *a final marking, obtain the optimum transition sequence, in terms of makespan, that starting from* M *reaches* M_F, *assuming no conflict for the use of resources*[11].

5.1 The heuristic function $h_{RCR}(M)$ definition and properties.

Definition 12: *RCR* based heuristic function $h_{RCR}(M)$.

Let *M* be a marking of a *b-NET*. Let M_F be a final marking for a *b-NET* which represents the goal marking for the scheduling problem.

We define $A:M \Rightarrow Q$ as the set of *place-tokens* (representing parts) of the marking defined as $\forall q \in Q$ if $M[q] = 0 \Rightarrow q \notin A(M)$; if $M[q] = n \Rightarrow \{q^1, q^2 \ldots q^n\} \subseteq A(M)$.

[11] This condition assumes that places of *M* representing resources contain as many tokens as the maximum number of operations that can be performed to produce the total number of parts. This assumption guarantees that parts do not compete for the use of resources.

The problem is to find the set $PAIRS(M_F,M)$ formed by elements of the form (q, p) $p \in A(M)$, $q \in A(M_F)$, that satisfies:

(1) $A(M_F) \equiv \{q \mid (q,p) \in PAIRS(M_F,M)\}$

(2) $\Sigma RCR(q,p)$, $\forall (q,p) \in PAIRS(M_F, M)$ is minimum[12]

i.e. all the *place-tokens* in M_F must be in $PAIRS(M_F, M)$ and the sum of the total operation cost for moving a part from p in M to q in M_F is minimal.

Finally: $h_{RCR}(M) = \Sigma RCR(q,p)/m \quad \forall (q,p) \in PAIRS(M_F, M);\ m=|R|$

In other words, $h_{RCR}(M)$ first computes the minimum machine utilisation needed (or total operation cost) to achieve the fully processed parts expressed as M_F (the goal state of the system) from a partial processed stage of these parts modelled by M. If we assume that the degree of concurrency between the machines is maximal, we obtain a theoretical lower bound on the minimum makespan by dividing the total operation cost by the number of machines (expressed as the number of *resource-places* R of the *b-NET*).

We claim that this function is admissible i.e $h_{RCR}(M) \leq h_{RCR}^*(M)$, because $h_{RCR}(M)$ represents the solution with minimum machine utilisation at the maximum concurrence rate. For any other possible solution, an alternate choice for a task must be considered instead, but then the utilisation time is at least the same but never less than the expressed by the *RCR* matrix. If we still consider the maximum concurrence between resources, the makespan will be the same or greater. Let us formalise this:

Proposition 2: $h_{RCR}(M) \leq h^*_{RCR}(M)$.

Proof: $h_{RCR}(M) = (\Sigma\, RCR(pi',pi))\ /m;\ (pi',pi) \in PAIRS(M_F, M)$ so the expression is minimal. By the definition of the *RCR* matrix, for each pair (pi',pi) there exists a firing sequence $s_i = t_0, t_1, \ldots, t_n$ $n > 0$ such that $Rs(s_i) = \Sigma Rt(t) \quad \forall t \in s_i$ is minimal. By proposition 1, this represents the minimum utilisation time of resources required for a token in p to reach p'. Thus we can express $h_{RCR}(M) = \Sigma\, Rs(s_i)\ /m$.

---

[12] the following equivalence must be assumed when obtaining : $q^i \equiv q$; $i \in N^+$ in order to obtain *RCR[q,p]*.

Consider $s_i' = t_0, t_1, \ldots t_k' \ldots, t_n$ an alternate firing sequence for $(pi', pi) \in PAIRS(M_F, M)$ with at least $t_k' \neq t_k$. Then, by definition 11 $Rt(t_k') \geq Rt(t_k)$ and so $Rs(s_i') \geq Rs(s_i) \Rightarrow Rs(s_i') \geq RCR(pi', pi) \Rightarrow h_{RCR}'(M) = h_{RCR}(M) - Rs(s_i)/m + Rs(s_i')/m \geq h_{RCR}(M)$.

Thus for any other possible sequence that reaches the final marking M_F from M, the makespan will be equal or greater to $h_{RCR}(M)$.

5.2 An algorithm for the calculation of $h_{RCR}(M)$.

The following algorithm (*fig. 21*) computes $h_{RCR}(M)$

Function $h_{RCR}(M)$:

Receives: *M*: the current Marking reached from an initial marking M_o
Computes: A lower bound *Cost* for the minimum makespan from *M* to M_F
Variables: *N*: A *b-NET* model.
RCR matrix of N
M_F: a copy of the goal Marking for *N*
NR: number of resources of the FMS modelled by *N*
Q: the subset of *buffer-places* of *N*.
F: the subset of places of Q representing *Job's* output buffers

Algorithm:

```
Cost =0;
∀p∈F
    while M_F[p] >0 do
        M_F[p] = M_F[p] - 1
        find p'∈Q | M [p']>0 such that RCR[ p';p ] is minimal
        M [p'] = M [p'] -1
        Cost = Cost + RCR[p', p]
Cost = Cost / NR
return Cost
```

Fig. 21: Algorithm for the calculation of h_{RCR}.

For each token in M_F at any final output buffer p, we search for any buffer p' in the current marking M, such that $RCR[p',p]$ is minimised. This place always exists

because a *b-NET* is practically live for any marking M reached from an initial marking M_0 (see property 1). We then remove the token indicating that a subpart in place p has been processed and increment the total resource utilisation cost by the *RCR* value.

The main loop continues until all parts have been considered. *Cost* is the total minimum machine utilisation needed to achieve M_F from a partially processed stage of the parts as modelled by M.

The cost of the algorithm is polynomial and depends on the cost of the operation ***find,*** which is O($|Q|$). However, Q must be organised as $Q = Q_1 \cup Q_2 \cup \ldots Q_n$ where Q_i contains the *buffer-places* belonging to J_i . This organisation reduces the cost of the operation ***find*** to O(max $|Q_i|$). The maximum number of times ***find*** can be executed is the total number of parts to be produced.

To conclude: the heuristic function is designed to quickly direct the search for *FMS* formulations where a) an acceptable degree of flexibility is observed; b) a balanced machine workload can be achieved (which is typically a desirable design objective); and c) there is a reasonably low operation cost variation between alternatives for a *task*. As stated in [Dario 96] in realistic *FMS*, the basic configuration and balancing phases of the design are aimed at ensuring that no individual part type (and we may add, machines) seriously degrades the system performance by continually bottlenecking particular machines. However, for problems where a resource is clearly a bottleneck and a balanced work-load is not possible, the estimate produced by *h(M)* might be too optimistic. The experience reported in chapter 6 confirms this hypothesis. Notice that in our scheduling approach, the search effort will be mainly controlled by means of an incomplete search algorithm. This is an interesting property from the point of view of the heuristic function, since even for *ideal FMS*, a machine breakdown can create a temporary imbalance of the system. In this situation, *h(M)* also becomes too optimistic, thus resulting in increased search.

5.3 Example of $h_{RCR}(M)$ application.

Figures 22, 23 and 24 show the search tree generated for the *PN* example of *fig.19* when using the *A** algorithm from chapter 3 employing $h(M)=0$; $h(M)=h^*(M)$ and $h(M)=h_{RCR}(M)$ respectively. The set *Unexplored* is ordered in the increasing

magnitude of *h(m)*. Notice that if several nodes yield the same value of *h(M)*, the one that is deeper in the graph has priority, breaking ties in a FIFO manner.

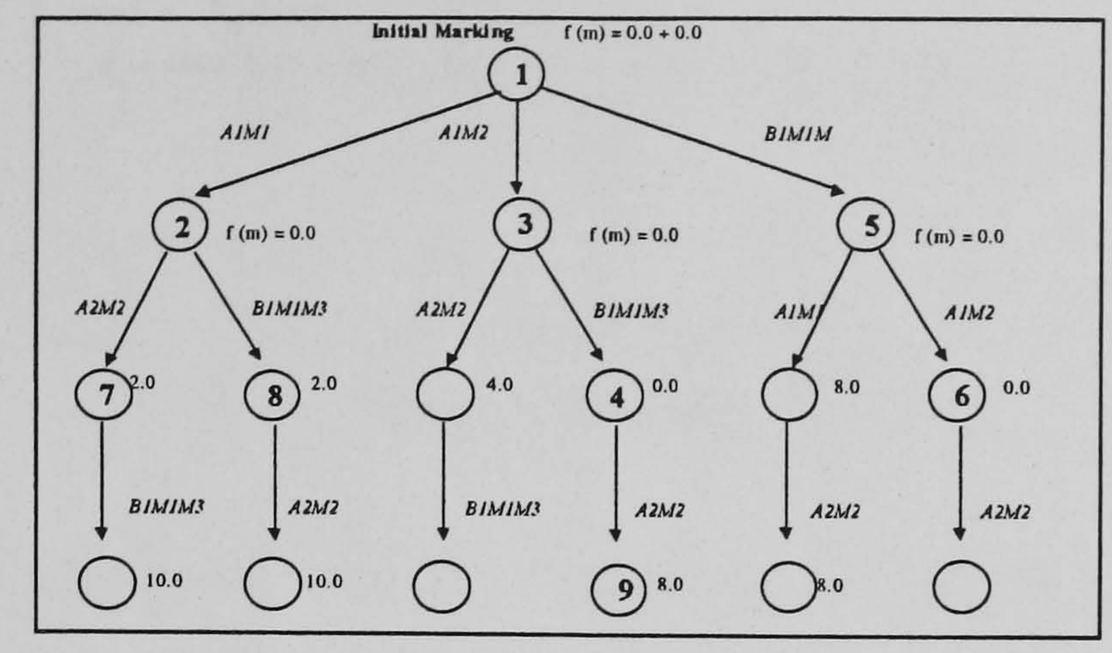


Fig. 22: search graph for h(M)=0.

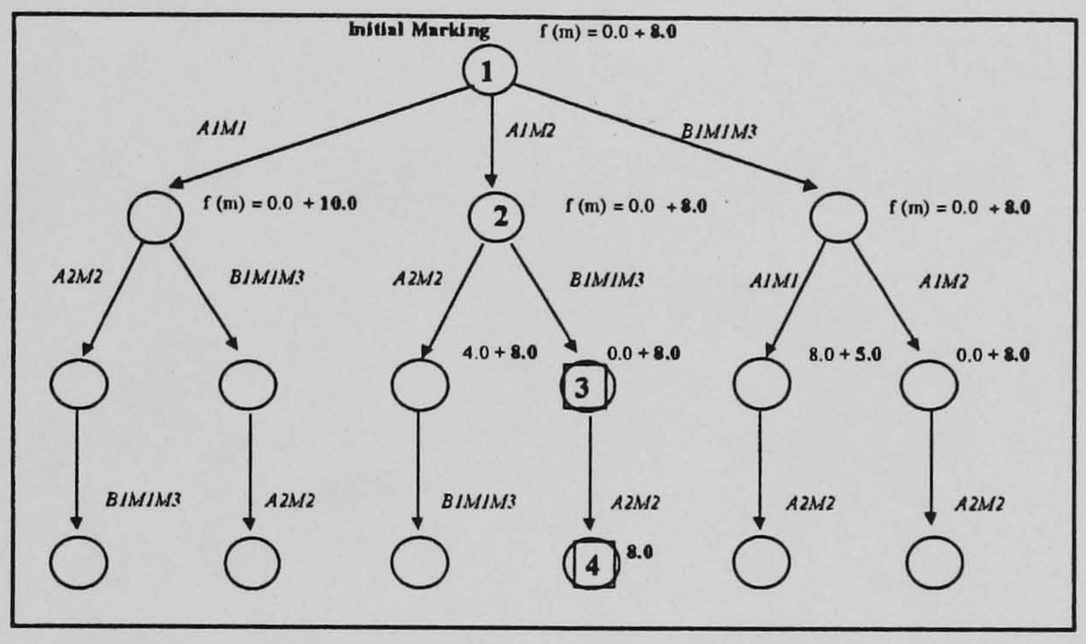


Fig. 23: search graph for h(M)=h(M).*

Notice that the heuristic function proposed quickly identifies the most promising markings and only explores one node more than the unrealistic setting *h(M)=h*(M)*. Notice that the execution for *h*(M)* follows a strictly depth-first search, and consequently finds the optimum solution in *O(d)* where *d* is the depth of the solution node. However since *h*(M)* already solves the problem, its calculation is of exponential complexity.

On the other hand, the breadth-search effect of not using predictive information can be observed for the pure best first search *h(M)=0*.

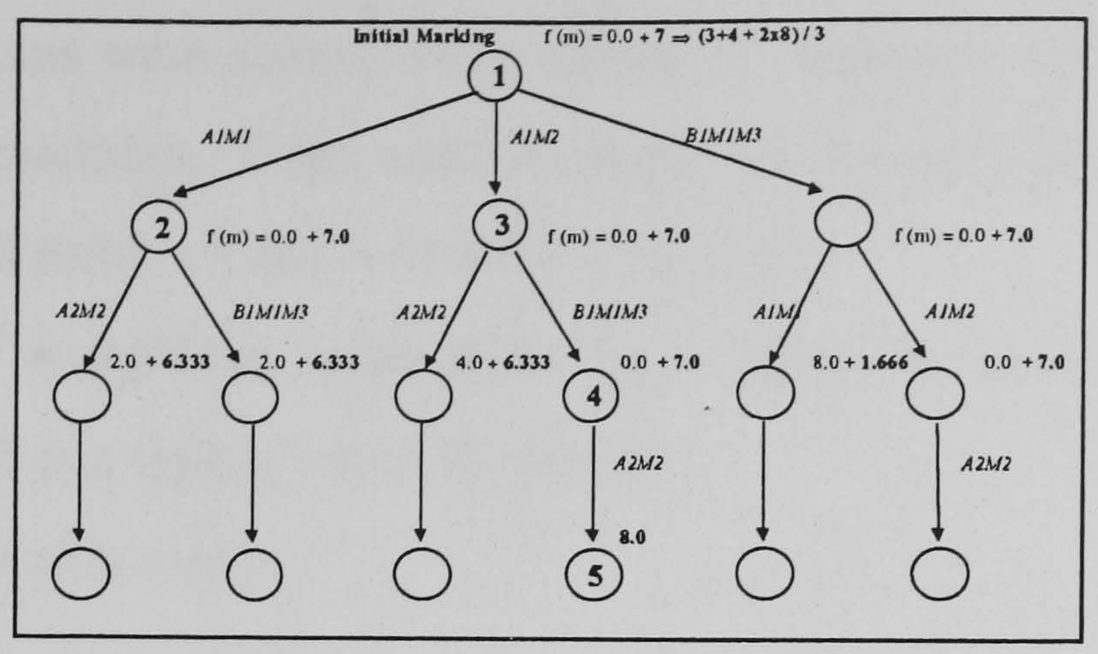


Fig. 24: search graph for $h(M)=h_{RCR}(M)$.

The effect of a heuristic function with a *depth-first* search component $h(M) = - A \bullet depth(m)$, is observed in *fig.25*. The value of A is set to $(4+2+3+8)/4 = 4.25$. Notice how the solution obtained is not optimum.

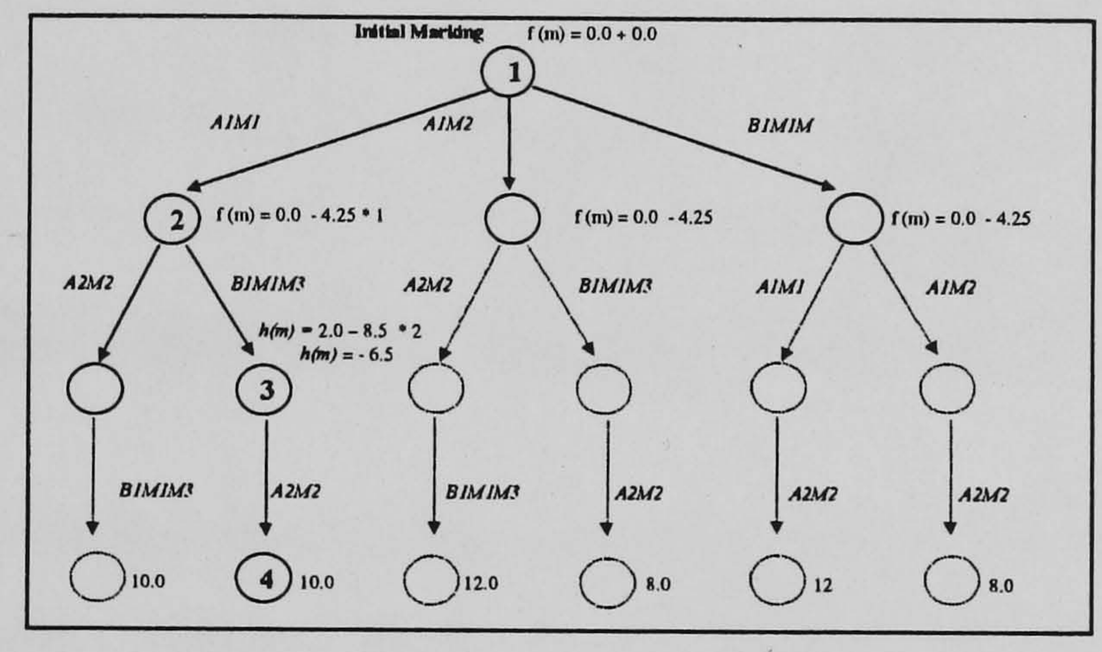


Fig. 25: search graph for $h(M) = - A \bullet depth(m)$.

6 Experimental results.

The following experimental analysis was performed in order to evaluate the effectiveness of the heuristic function developed. A collection of 2000 problems consisting of 3 different jobs and 3 machines were randomly generated. Randomness is achieved by the employment of robust random number generator routines for uniform and normal distributions [Numerical 92].

These problems were solved using a pure *A** approach without identification of previously reached markings. Three heuristic functions were evaluated:

a) $h_{BF}(m)=0$; which turns *A** into a *Best First* approach.

b) $h_A(m) = - A / N \bullet depth(m)$ being *A* the mean operation cost and *N* the number of resources. Which is a typical non-*PN* heuristic.

c) $h_{RCR}(m)$ a *PN* based heuristic.

The first analysis corresponds to the study of the search space explored by each heuristic. *Table 6* shows the descriptive statistics for the number of markings (states) explored by *A** employing each of the heuristic.

Heuristic Function	Minimum	Maximum	Mean	Std. Deviation
$h_{BF}(m)$	11	2293	479.56	399.92
$h_A(m)$	10	1997	307.39	284.14
$h_{RCR}(m)$	5	1131	105.22	115.03

Table 6. Descriptive statistics of the no. of markings explored for the set of 2000 problems.

The first interesting result is that, although $h_A(m)$ explores a larger number of markings than $h_{RCR}(m)$, which indicates a less informed search, it suffers from the problem of non-admissibility. i.e, while $h_{RCR}(m)$ always finds on optimum solution, $h_A(m)$ fails to do so in 6 problems out of 2000. Further experiments have shown that with less optimistic settings of $h_A(m)$ (decreased *N*) the search effort tends to match the search effort with h_{RCR} , but the number of cases were a near optimum solution is found also increases.

In order to compare in more detail the effectiveness of each heuristic function, we studied the statistics of the relative difference of the search space explored for each problem.

The relative difference, expressed as *Rd(A,B)* is calculated as the following expression., where the term *Explored( )* makes reference to the number of markings explored when employing each heuristic function.

$$Rd(A,B) = \frac{Explored(A) - Explored(B)}{Explored(A)} \%$$

Table 7 shows the descriptive statistics for *Rd(A,B)* for the set of problems solved. Figures 26, 27 and 28 show the frequency histogram of each distribution respectively.

Algorithm	Minimum	Maximum	Mean	Std. Deviation
$Rd(h_{BF}, h_A)$	0.0	94.88	31.42	21.05
$Rd(h_{BF}, h_{RCR})$	2.74	97.90	73.35	16.82
$Rd(h_A, h_{RCR})$	-35.59	96.6	62.48	16.62

Table 7. Statistics of the difference of search effort between heuristics expressed as Rd(A,B).

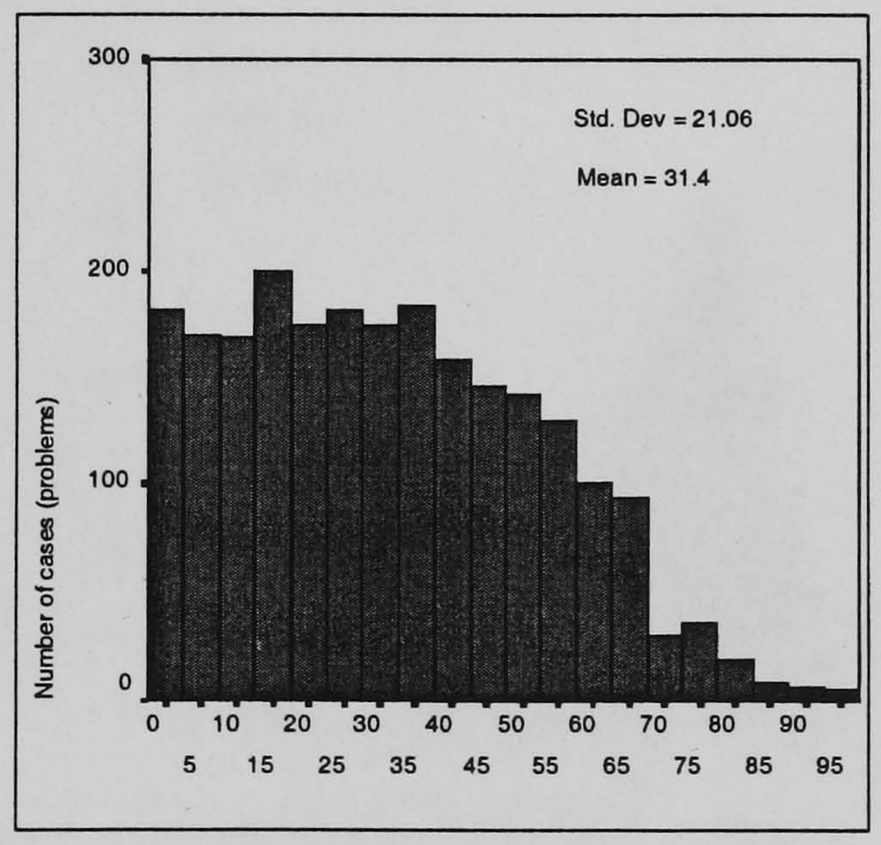


Fig. 26: Histogram for the % relative difference of markings explored of h_{BF} and h_A ($Rd(h_{BF}, h_A)$)

Notice that the distribution of the relative difference for the number of states explored by $h_{BT}(m)$ with $h_A(m)$ approximates to a uniform distribution, showing that for many problems $h_A(m)$ results are too optimistic, thus confirming the tuning difficulties of $h_A(m)$. Such a distribution is not produced for $Rd(h_{BF}, h_{RCR})$ as observed in *fig. 27*, where the search effort reduction approximates more to a normal curve where the number of problems for which h_{RCR} performs poorly is marginal.

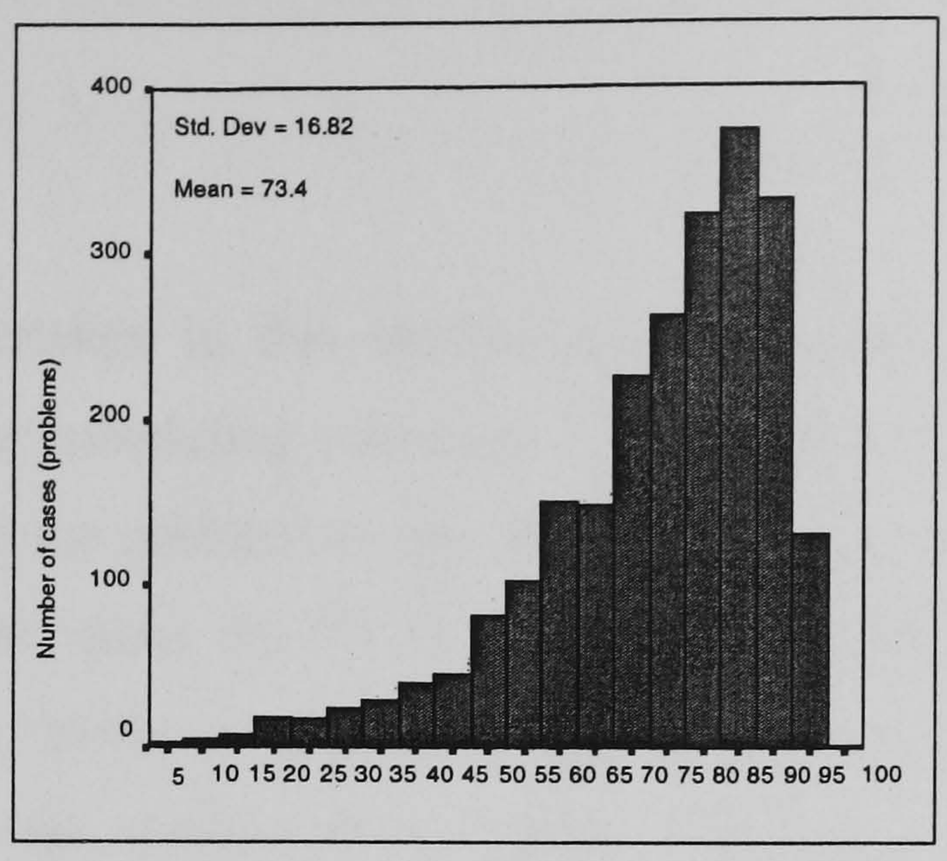


Fig. 27: Histogram for the relative difference of markings explored of h_{BF} and h_{RCR} expressed as $Rd(h_{BF}, h_{RCR})$

Finally *fig. 28* shows the histogram for $Rd(h_A, h_{RCR})$. Only in four cases is the search effort reduced with respect to $h_{RCR}(m)$ while, in general terms, the difference of search effort follows a normal curve with mean 62.5 and std. deviation of 16.62.

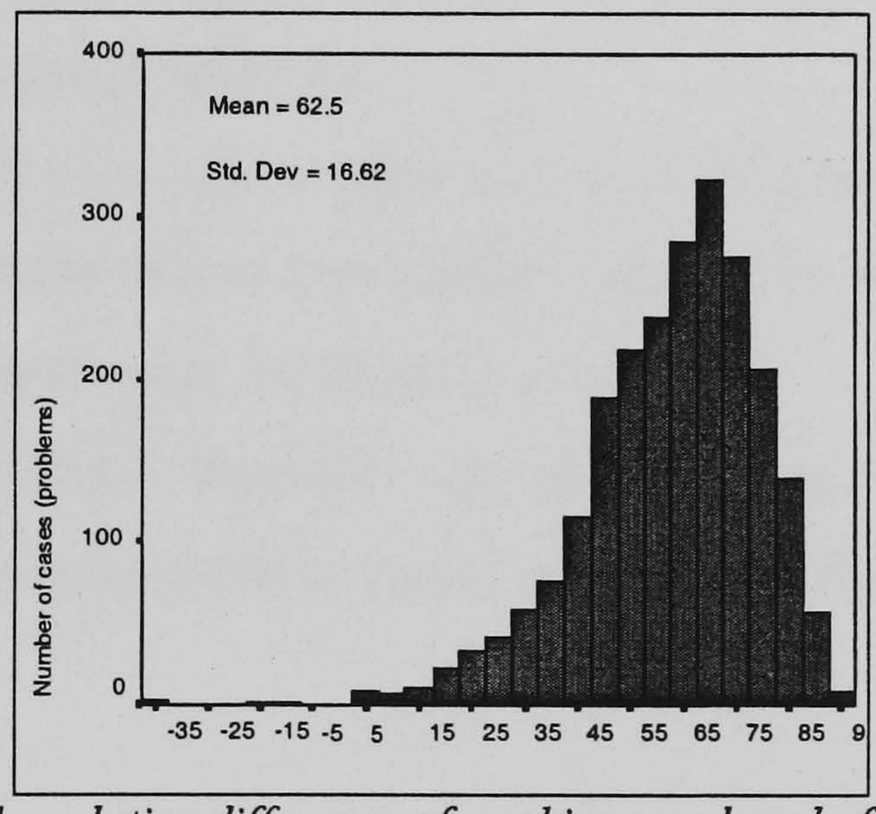


Fig. 28: Histogram for the relative difference of markings explored of h_A and h_{RCR} expressed as $Rd(h_A, h_{RCR})$

The results observed in these experiments leads to the following conclusions.

a) The fact that $h_A(m)$, which is not a *PN* based heuristic, seems to be less informed than $h_{RCR}(m)$ and is more difficult to tune.

b) As explained in section 5.2, the use of limited global information in $h_A(m)$ is likely to be leading the search to the exploration of the search space guided by strategies such as *do the fastest operations first*. This seems to be increasing the backtracking effort.

7 Summary.

The work undertaken in this chapter has shown the potential of *PN* to use reachability analysis for scheduling purposes. The *PN*, as a representation tool, easily allows the relaxation of the problem by the elimination of constraints that are naturally modelled by a *PN* model. Also, the *PN* theory and mathematical analysis allow an easy formulation of heuristic functions whose properties can be studied and that represent an approximate solution to the problem. From the point of view of calculating estimates, *b-nets* are useful since they are obtained from the *PN* model of the *FMS*. This is interesting since the algorithm for calculating the heuristic employs a modelling paradigm that performs close to the original model over which we simulate scheduling alternatives. Consequently good estimates of possible behaviours can be obtained.

The concept of the *RCR* matrix allows the core of the heuristic function to be calculated prior to search. This reduces the computational cost of calculating the heuristic function, thus supposing an improvement over previous *PN* based heuristics that needed from mathematical relaxation.

From the point of view of the effectiveness of the heuristic function proposed, the experimental results have shown considerable improvement over previous heuristics that have been widely employed in the literature consulted.

The benefits of the h_{RCR} heuristic function will be later demonstrated in chapter 6, where this heuristic is employed in conjunction with incomplete search procedures.

Chapter 5. Reachability graph truncation based in *PN* dynamics and heuristics

1 Introduction.

The *FMS* scheduling problem can be defined as a procedure that identifies the best way to reach a goal among a very large number of possibilities. In fact, obviating deadlocks, any possible sequence of operations that completes all the parts leads to the same final state in terms of part status. i.e, all parts have been processed, the difference is in the performance of the system. While certain schedules lead to optimum solutions, others do not. This chapter reviews the search space defined by *cb-PN* reachability analysis and how the knowledge acquired can reduce search effort by avoiding paths that do not lead to a better schedule.

The techniques studied here attempt to:

a) Identify the best alternative among partial schedules that lead to a similar system status.
b) Avoid the generation of schedule permutations where two or more non-conflicting operations become active at the *same* time instant.
c) Avoid the generation of partial schedules that do not yield an optimum solution.

This kind of *PN* reachability analysis has received little attention in the work consulted, despite its potential in terms of search effort reduction. This chapter is organised as follows: section 2 presents a comparative study of different strategies to identify previously reached states and how two states describing the same system status can be compared and differentiated. Results will show that incorporating a test to prune paths that assure optimality adds complexity to the search procedure. On the other hand, less complex heuristic tests may be used achieving higher search reduction. The price paid is loss of admissibility.

Section 3 studies why different paths reach a similar system status and proposes a *PN-based* branching scheme (*Controlled Generation of Successors*) that avoids branches that do not yield better schedules. In short this approach adapts the following

methodology: for each active operation (transition) the algorithm decides if the operation is to be delayed or applied. The decision to delay an operation is motivated by the desire to keep resources available for other operations not yet active. The algorithm maintains this decision until the evolution of the system forces an operation to be reconsidered. Empirical analysis will show that the method ensures optimality and considerably reduces the search effort, thus overcoming the drawbacks of the methods presented in section 2.

2 Identifying previously explored markings.

2.1 Similar and more promising markings.

When the *cb-NET A** algorithm was presented in *fig. 6* in chapter 3, a question arose about the meaning of the terms *similar* and *more promising than* (*fig. 29*):

If *M'* is *similar* to another marking *M''* in *UnExplored*
  if *M' is more promising than M''* **then**
    Put *M'* in *UnExplored*
    Remove *M''* from *UnExplored*
goto (1)

Fig. 29: Subpart of cb-NET A of fig. 6.*

The objective of this test can be explained as follows: *M'* is a new marking reached from M_0 following some sequence of transitions (*s'*). The marking *M'* indicates the state of the partially processed parts as well as the state of the resources. Let us suppose that another marking *M''* has been previously found expressing the same distribution of tokens (and hence the same state of parts and resources) reached from M_0 following its own sequence of transitions *s''*. That is, two markings, expressing the same state of parts and resources have been reached following two different sequences (two different partial schedules and consequently different system performance).

It is interesting to determine which path *s'* or *s''* represents a *more promising* way to go. Suppose that *M'* is found to be *better* than *M''*[13]. By rejecting the exploration of *M''* the algorithm will not duplicate search effort that is not *likely* to lead to a better solution.

An interesting question is: which sequence represents a better schedule in terms of our objectives?. Since both *M'* and *M''* represent the same marking in terms of the distribution of tokens, their associated heuristic functions *h(M'), h(M'')* will be the same[14]. It is tempting to think that the current makespan *g(M)* would easily differentiate between alternatives: the marking yielding the smallest *g(M)* indicates, a priori , a better schedule.

However, this approach is too simplistic for the following reason. A marking of a timed-*PN* has its tokens in two possible states: available and unavailable. *g(M)* express the current time of the system in terms of available markings as expressed in chapter 4. However, the status of the unavailable markings is not reflected in *g(M),* although it has a great effect in determining the next transition to be fired next (*Fig. 30*).

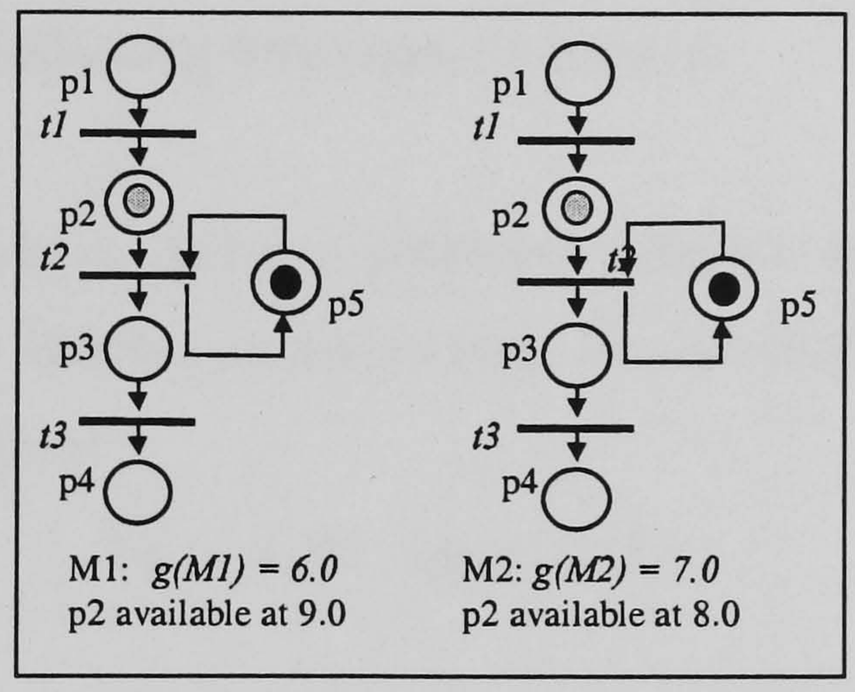


Fig. 30: two similar markings.

The distribution of tokens in both markings is equivalent, which means that the product is being transferred to the cutting machine. Comparing makespans, *M1* seems to be better than *M2.* However, at M1, *t2* will not became enabled until 3 time units, whereas at *M2, t2* will be enabled earlier (in 1 time unit). This is a simple example, but leads to the conclusion that an exact comparison between markings with the same token distribution is only possible if also taking into account the possible transitions to be

[13] Although we are actually comparing partial schedules, since each schedule is associated with a marking the expression *M' better than M''* is equivalent to *s' better than s''*.
[14] This is valid for a typical heuristic function as the ones presented in chapter 4.

fired. From the point of view of firing *t2*, *M2* represents a better solution, but this reasoning may not be applicable to other possible firing sequences.

As mention before, a second alternative is to use full node estimation: $f(m) = g(m) + h(m)$. The marking yielding the worst value of $f(m)$ may be discarded. This approach is employed in [Xiong 98]. Consider m and m' two markings with the same token distribution. Evidently, if $f(m)$ is based in the distribution of tokens in M, both nodes are equally valued, i.e, $h(m)=h(m')$. Hence, the result of the test depends on $g(m)$. If for the general case we cannot confirm that $h(m)=h(m')$ both markings are compared in basis of an estimation of the lower bound, so we cannot claim the admissibility of the test in this case.

2.2 A safe test that guarantees optimality.

To ensure that a marking M' will produce a better schedule than a previously reached marking M, each token in M' should be available sooner than the same token in M. To formalise this, the following definitions are given.

Definition 13: Define A_T as the set of pairs (p,t) where p represents a place and t the time when the token will become available in p. A_T is constructed from a marking M using the following definition:

$$\forall\, p \in A(M) \quad (p, t) \subset A_T$$

That is, for each token in a *place-token p,* an element (p, t) is included in A_T where t is the elapsed time of this token.

Definition 14: A marking M' is said to be *better* than a marking M with the same token distribution if it is possible to create a set of pairs $A_{T2} : A_T \times A_T'$ that satisfies the following conditions:

(1) If $((p,t), (p',t')) \in A_T^2$ then p represents the same place as p'

(2) $\forall\ (p,t) \in A_T$ there exists one and only one pair $((p,t), (p',t')) \in A_T^2$

(3) $\forall\ (p',t') \in A_T'$ there exists one and only one pair $((p,t), (p',t')) \in A_T^2$

(4) $\forall\ ((p,t), (p',t')) \in A_T^2 \quad t \leq t'$

Condition (1) guarantee that pairs of tokens can only be formed between elements that refer to the same place in the *cb-NET* model. Conditions (2) and (3) ensure that each element of A_T or A_T' must be included in A_T^2 but also that each element can only be *used* once to create a pair of A_T^2. Hence, $|A_T^2| = |A_T| = |A_T'|$.

Finally condition (4) ensures that a token of M will become available before or at the same time as its pair in M'. If M is *better* than M', the best schedule that can be obtained from M_0 to M_F through M' is equal to or worse than the best possible schedule through M. It is important to note that if M is not *better* than M' this does not necessarily imply that M' is *better* than M.

Proposition 3: If M is *better* than M' then $h^*(M) \leq h^*(M')$

Proof: Since M represents the same distribution of tokens as M' then any transition t enabled in M is also enabled in M. As M is *better* than M' each of the tokens that enable the transition b in M' has its corresponding pair in A_T^2 that becomes available earlier in M. Consequently transition t will fire in M no later than in M'. It is easy to see that the firing of t in M and M' produce two new markings M^2 and $M^{2'}$ which have the same distribution of tokens and satisfies the condition that M^2 is *better* than $M^{2'}$. Repeating the process with M^2 and $M^{2'}$ we will eventually reach two final markings M_F and M_F' where M_F is better than M_F'.

The problem with this approach is complexity. Each marking must be stored with its full state description. In addition the algorithm that determines whether a new marking M is better than another marking M' has a cost of $O(n^2)$ for the general worst case with n being the number of different tokens in the marking, and $O(n)$ for the general case. Since this comparison is made with all the explored and unexplored markings, the computational overhead is large. This is not surprising and has been noted previously [Nilsson 82].

2.3 The complete *cb-NET A** algorithm.

The complete *cb-NET A** algorithm can now be defined *(fig. 31)* from the one shown in *fig. 6* in chapter 3:

Algorithm *cb-NET A**:
Receives: M_0, M_F: The initial and final marking of a *cb*-NET *N* from *FmsML*.
Returns: Sequence of transitions representing the schedule.
Variables: *UnExplored:* List of *new* markings for exploration.
Explored: List of markings already explored.

UnExplored = M_0
Explored = ∅
(1) while *UnExplored* is not empty **do** the following
(2) Remove a marking *M* from *UnExplored* yielding the smallest *f(M).*
(3) If *M* matches with M_F **then**
Retrieve the path from M_0 to M_F
Exit with success.
(4) else Put *M* in the *Explored* List
(5) For every transition *t* *enabled* in marking *M* do the following:
Obtain the marking *M'* as result of firing *t* in *M*
(6) while exists *M''* ∈ *UnExplored* | *M'* = *M''* **do**
(6.1) **If** *M'* *is better* than *M''* then
Remove *M''* from *UnExplored*
(6.2) **else If** *M''* *is better* than *M'* then
goto (5) *// M is not considered*
(7) while exists *M''* ∈ *Explored* | *M'* = *M''* **do**
(7.1) **If** *M''* *is better* than *M'* **then**
goto (5) *// M is not considered*
(7.2) *// else continue'*
Put *M'* in *UnExplored*
goto (5)
goto(1)

Fig. 31: cb-PN based A algorithm.*

Step (6) compares the new marking *M'* with all the unexplored markings. For each marking *M''* with the same token distribution a comparison is made. If *M'* is determined to be *better* than *M'' (6.1) M''* is removed from *UnExplored*. If *M''* results in a *better* marking than *M' (6.2)* then *M* is not considered, and the algorithm considers the successor of the current marking under exploration.

If *M'* exits successfully from *(6)* it is compared to the set of markings which have already been explored *(7)*. If a marking *M''* is found to be *better* than *M'*, the latest is rejected *(7.1)*. If not *(7.2)*, *M'* may represent a better alternative than a previous explored path and is thus included in *UnExplored* for further exploration.

The following section presents an empirical study of four different tests that establish a *more promising* criterion between two markings that represent the same static state of the system (*similarity*).

2.4 Experimental comparison with different check tests.

Test 1: A marking *M* is *more promising* than a previously generated marking *M'* with the same token distribution if *M* is *better* than *M'*. This test is admissible, i.e, it never rejects a marking that leads to a better solution.

Test 2: A marking is *more promising* than a previously generated marking *M'* if $g(M) < g(M')$. This is a *heuristic* test, in the sense that it does not guarantee optimality, however, the computational complexity is lower than for *Test 1*.

Test 3: Test 2 only considers time information regarding the set of available tokens of the marking. Several heuristics can be proposed to add a time component to the tokens that are still not available. The following heuristic test has been shown to provide good results in our experiments. A marking is *more promising* than a previously generated marking *M'* if $j(M) < j(M')$. $j: M \rightarrow R^+$ is calculated as $j(M) = g(M) + U(M)$. *U:* $M \rightarrow R^+$ is the mean time for the as yet unavailable tokens of the marking to become available in *M*.

Test 4: is similar to *Test 1,* however the *better* criterion is constrained to the set of places *R* that model resources. The idea underlying this test is to determine which of two markings expressing the same distribution of parts in the *FMS* will release the resources earlier. A similar test is employed in [Inaba 98] with the aim of detecting repetitive processes.

The experiment consisted of the generation of a random set of *FMS* descriptions with 3 jobs, 3 machines and a maximum of 3 tasks per job. 85% of operations allowed alternate routing. The problem is small since it needs to be solved optimally. However, the problem set is representative enough i.e., 10x10x10 is just a larger version in terms of search space size but does not introduce new elements concerning scheduling decisions.

Each problem was first solved with the *cb-NET A** algorithm using h_{RCR} as the heuristic function without the test for similar markings. The same problem was then solved using each of the four *tests* defined above.

Table 8 summarises the descriptive statistics for the % of search space pruning achieved by each of the above-described tests when compared with a *pure cb-NET A** that not apply a test for *similar* markings.

Test	Minimum	maximum	mean	Std dev
Test1	1.1	67.0	27.7	16.0
Test2	91.7	98.4	96.1	1.3
Test3	88.0	98.4	95.7	1.4
Test4	56.6	79.0	44.0	19.2

Table 8: % of search space pruned for each test.

Table 9 shows the % of the number of problems not solved optimally by each test. It also shows the statistics for the relative distance of the solution obtained against the optimum solution known for the problem.

Test	% of problems not solved optimally.	Relative difference of the solutions obtained with respect to the optimum solution.			
		Min.	Max.	mean	Std dev
Test1	0.0	0.0	0.0	0.0	0.0
Test2	71.0	0.4	51.7	10.0	7.9
Test3	42.2	0.4	30.0	6.0	5.1
Test4	14.6	0.5	22.0	4.7	4.7

Table 9: Descriptive statistic of the optimality for each test.

Results clearly show that although *Test 1* ensures optimality, the search reduction is relatively small compared with *Test2*. However, it is noted that the percentage of problems not solved optimality with this test is close to 70%.

Test 3 achieves almost the same percentage of search reduction but considerably increases the quality of the solution reducing both the number of problems not optimally solved and the distance of the solution obtained from the optimum solution.

Finally, *Test 4* represents a compromise between the admissibility of *Test1* and the effectiveness of *Test3*.

In the work consulted, the majority of approaches that follow [Lee 94] consider the test for duplicate markings as a comparison of the current elapsed time of the marking *g(m)* or makespan i.e, *Test 2.*

We finish this section with the following argument. The strategy of identifying previously reached markings is inherit from the definition of the standard *A** itself. We only have made use of *PN* analysis to achieve effective search reduction, but *A** assumes complete exploration of the entire search graph. This represents a problem when incomplete search procedures are adopted, since the part of the entire reachability graph observed is minimal. Neither these strategies nor the ones in the work consulted investigate the application of a branching scheme, based on *PN* structures and dynamics, that reasons forward about the search space generation without having to compare with previous markings. The next section describes a *successor generation* strategy that avoids the generation of *PN* markings that represent futile or duplicate paths in terms of the optimisation objective.

3 Controlled generation of successors: *CGS.*

If we think about what situations may lead to *similar* markings we can readily identify two groups:

a) Permutations of the same schedule, where two or more transitions can be fired at the same instant, leading to identical states.
b) Schedules that do not lead to an optimum solution, there exists an alternate better schedule which is easily identifiable[15].

This is, obviously, not exhaustive – there will be other cases, but taking a) and b) into account should improve performance. Let us discuss a) and b) separately.

[15] Those schedules are usually known as non-active [Pinedo 95]. A feasible schedule is called active if no operation can be completed earlier by altering processing sequences on machines and not delaying any other operation.

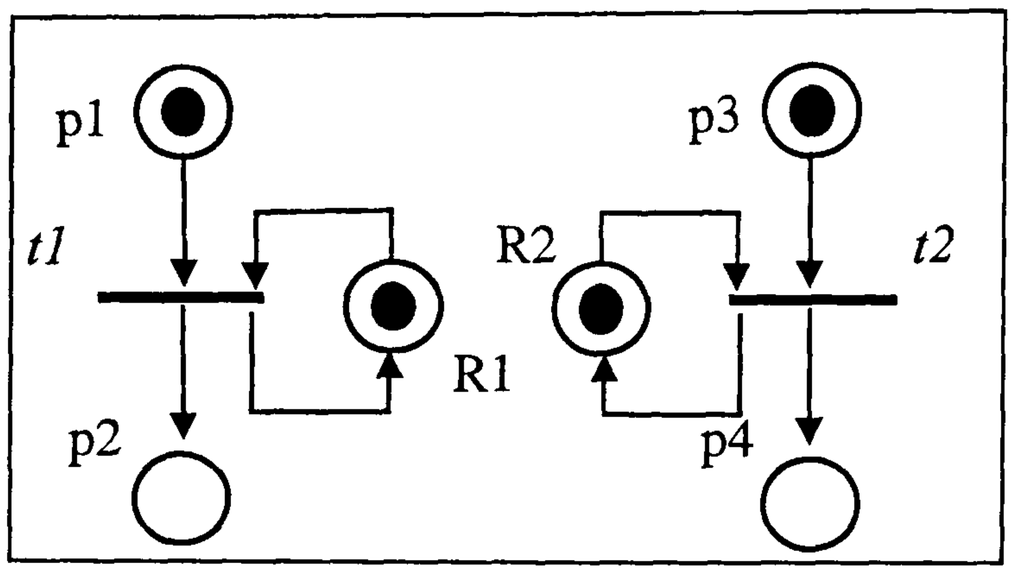


Fig. 32: Marking M_0 with two concurrent enabled transitions.

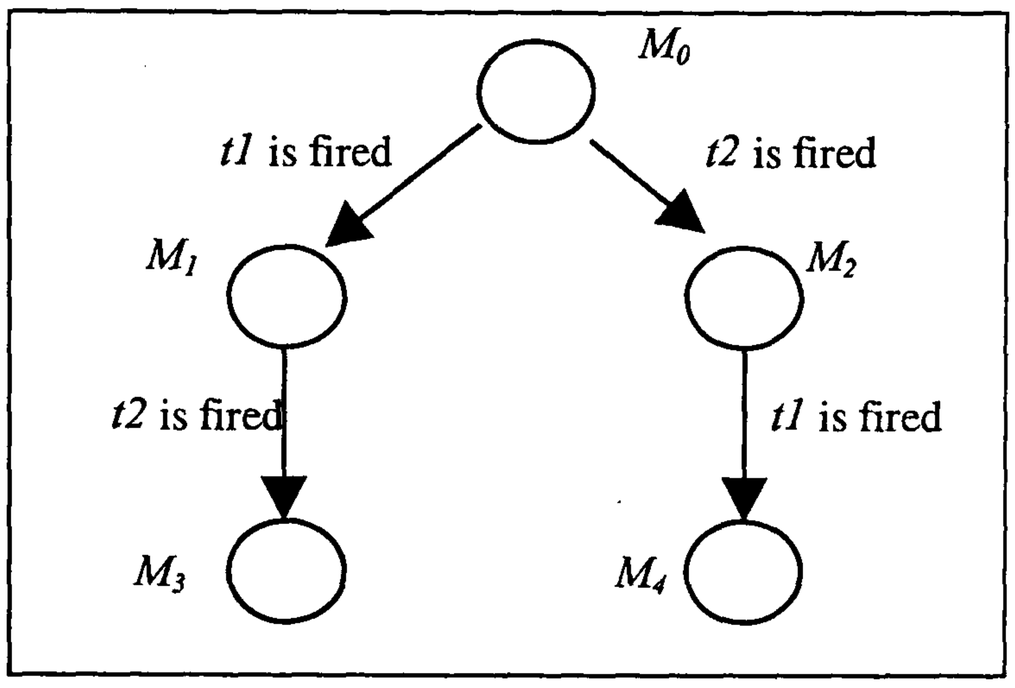


Fig. 33: Search graph for M_0 in fig 32

a) In the *cb-PN* of *fig. 32* two transitions can be fired simultaneously: *t1* and *t2*. This situation leads to the reachability tree expressed in *fig. 33*. Obviously, M_3 and M_4 are equivalent. *A** would eventually prune any of those markings using the techniques to identify same markings presented in the previous section. However, while not detected *(M_1 and M_2)*, they represent candidate paths to be explored and lead to unnecessary computational effort.

b) Now consider the *PN* of *fig. 34*. Again *t1* and *t2* are enabled, but now *t1* can be fired earlier than *t2*. An order permutation of the firing will lead to *similar* markings, as seen in *fig 35*, but this time one is clearly *better* than the other, i.e. the subpart in *p2* is available sooner in M_3 than in M_4. In other words, M_3 is an active schedule while M_4 is not. One may think that a successful strategy is to fire the transition that becomes enabled soonest, but this idea does not work[16]. However, if we eventually

[16] Such a strategy is known as a non-delay. A feasible schedule is called non-delay if no machine is kept idle when there is an operation available for processing [Pinedo 95]. It must be noted that the optimum schedule is an active one, but it is not a non-delay schedule for the general case. Hence, a scheduling search strategy based in the generation of only non-delay schedule is not optimal, although it reduces the search space [Doulgery 87]. A non-delay strategy is easy to implement with *PN* structures by always forcing the firing of those transitions that have all the tokens required available in the current marking [Azzopardi 94].

fire *t2* first (i.e, delay *t1*), it seems logical not to fire *t1* afterwards, since a better schedule could have been obtained by firing *t1* first.

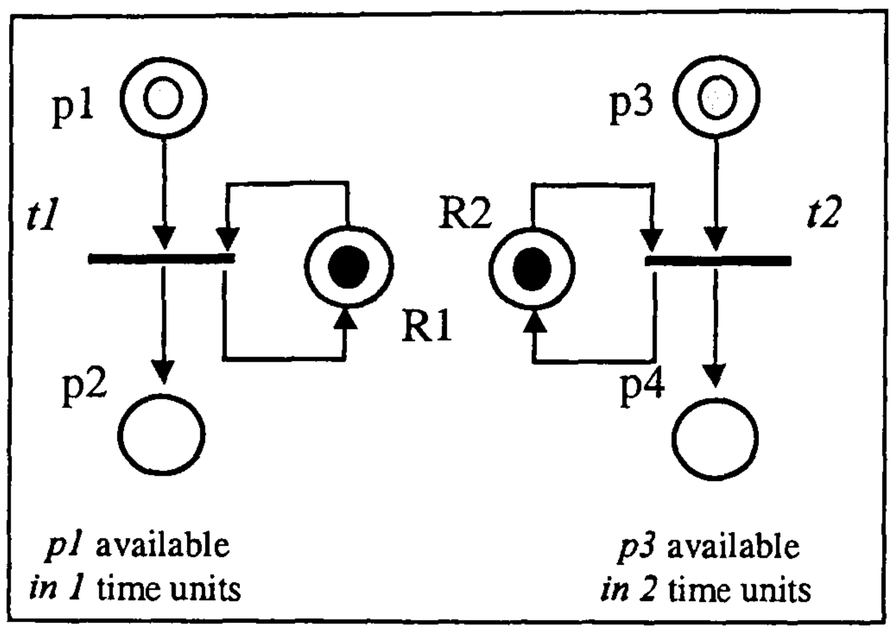


Fig. 34: Marking M_0 with two enabled transitions.

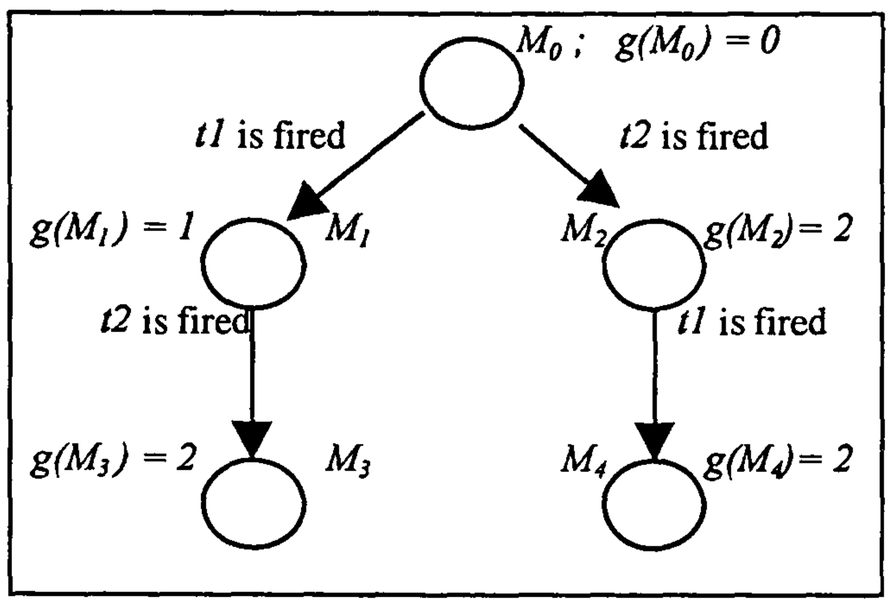


Fig. 35: Search graph for M_0 in fig. 34.

The strategy that we pursue with our *CGS* method is often observed in mathematical programming approaches under the name of *dominance rules*. [Hariri 97] include several dominance rules for Flow Shop scheduling based on analysis of the problem domain. [Demeulemeester 92] report a *next-node* generation strategy to avoid schedule permutation caused by unrelated operations and provide a rule to transform *non-active* partial schedules into active ones. However, the problem domain is a *JSS* and the fact that there is no alternative routing is considered may invalidate the results for *FMS* descriptions. In addition, to the best of our knowledge, in both approaches nodes are actually generated and then pruned. An enumerative algorithm for the generation of *all* active schedules for *FMS* formulation is given in [Shen 88], but the method is not event driven, leading to duplicate copies of schedules.

However, in works dealing with *PN* only the work of [Jeng 98] seems to have given attention to this issue. They devised a function that prunes paths that are permutations of non conflicting transitions that can be fired at the same instant. Apart from the fact that they only seem to be considering concurrent transition permutations,

their approach actually generates the markings which are pruned later. In other words, to the best of our knowledge no one else has considered a *PN* based pre-emptive pruning strategy.

3.1 Description of the method

Let us consider a marking *M*. Associated with this marking, is a data structure called *Agenda(M)* [17] as an ordered list of pairs *(t, r)* where *r* is the delay needed for all the tokens required by *t* to become available from the current time of the marking, provided that no other transition is fired earlier. *Agenda(M)* is ordered in the increasing magnitude of *r*. Such a list may be understood as the set of transitions that may fire in *M* to generate successor nodes. For the initial marking M_0, *Agenda*(M_0) is defined as the set of enabled transitions, however, it is important to mention that not all of the transitions enabled at an intermediate marking *M* are included in *Agenda(M)*.

Let us take the first pair *(t,r)* of the *Agenda*. We can now consider two possible actions:

a) Actualise the marking *M* increasing the time by *r* and apply *t*, obtaining a new marking *M'*.
b) Decide not to fire the transition, and generate a new marking *M''* equivalent to *M'* but with *(t,r)* removed from *Agenda(M')*.

A first interpretation leads us to a search tree of branching factor two as seen in *fig. 38*. Each marking represents a decision point that divides the feasible schedules into a) schedules were transition *t* was fired in *M*, and b) schedules where transition *t* was not fired in *M*.

[17] The term *Agenda* is imported from rule-based production systems. The agenda contains the *instances* of rules whose conditional part is satisfied in the current state.

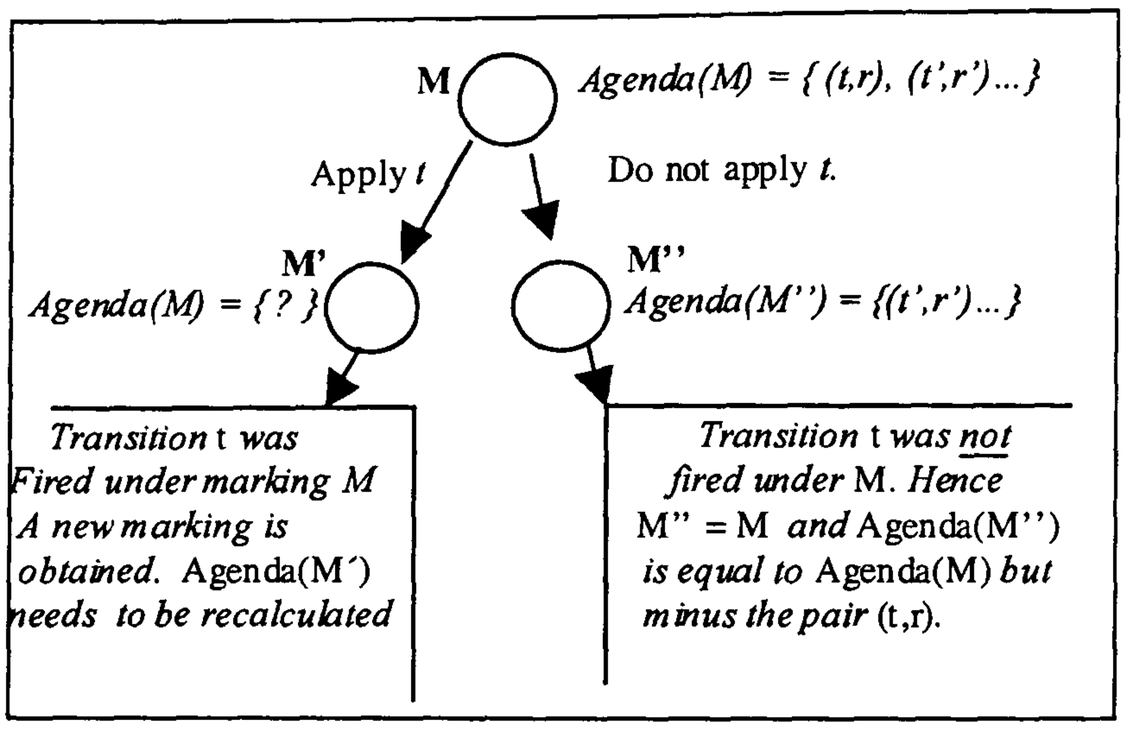


Fig. 38: Search tree generated considering Agenda.

The first question that arises is: *why not fire* t *?*. To answer this question, consider the *PN* of *fig. 39*. Resource *R1* is ready, and two parts from *B1* can be processed, however, resource *R2* is currently processing another subpart, but will be available in *1* time units. On the other hand, a part in buffer *B2* will be ready to be processed in *1* time units. The cost associated with *t1,t2* and *t3* are *1, 3, 1* respectively. If we decide not to fire *t2* now, and increase time by *1*; *t1* and *t2* can now both be fired, and after the release of *R1*, *t2* becomes enabled again and can be fired, resulting in a better schedule than if *t1* had been chosen the first time. The reason, then, for not firing *t1* first is to keep available resources that might be used by other operations and progress to a better schedule. This is usually referred to as strategic or tactical delay.

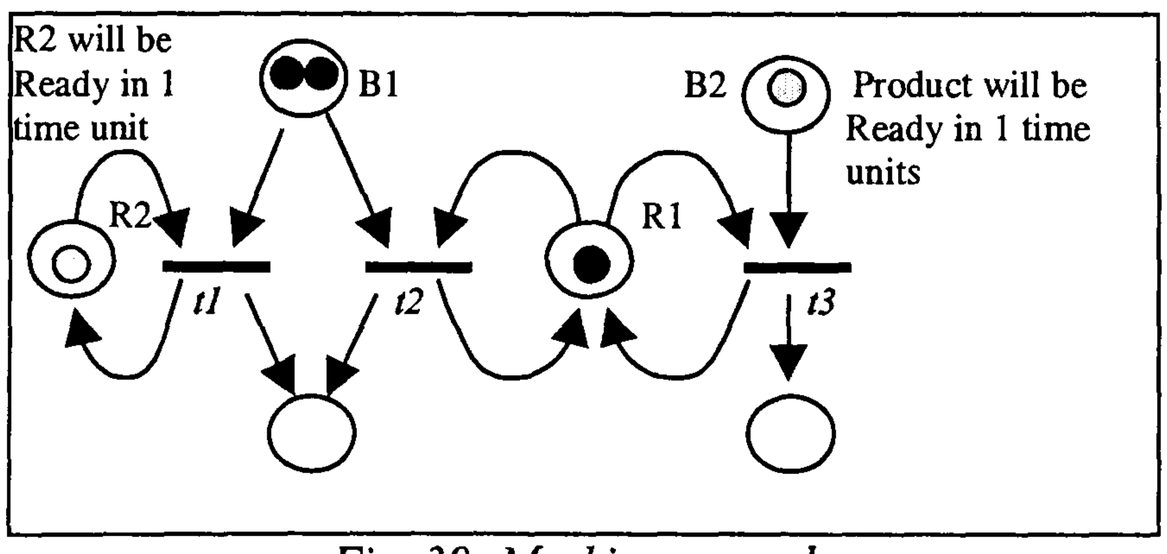


Fig. 39: Marking example

The agenda for this marking is *{ (t2,0) (t1,1) (t3,2) }*. Let us consider the branch where *(t2,0)* is not fired, which leads to a search state *M'* equal to *M* except that the pair *(t2,0)* has been removed from the agenda i.e. *Agenda(M')={(t1,1), (t3,2)}*. Notice that *t2* is still enabled in *M'* but it will not fire since it is not included in *Agenda*. The reason for this is clear, we have decided that *t2* should not fired and we will maintain this decision.

This, of course, leads to a second question: *If a transition has been removed from the agenda, when should it be re-considered.?* Let us try to answer this:

Two situations are defined where a transition *t* that was in *Agenda* is not in a subsequent marking:

a) The firing of a transition *t'* in marking *M* produces a new state *M'* where *t* is not enabled, since, for example, *t* and *t'* are mutually exclusive, or the firing of *t'* excludes the firing of *t* in terms of resource constraints. Consequently, *t* will not be further included in *Agenda(M')*.

b) Transition *t* has not been fired in *M,* and is removed from agenda, although it is enabled.

Situation *a)* is solved by the *PN* enabling rule i.e., transition *t* will be included in the agenda of any marking *M'* that is a successor of *M* where *t* satisfies the enabling rule. Situation *b)* escapes to the usual *PN* token game since *M* is actually enabled but has been removed from *Agenda* by an arbitrary decision. Transition *t* will not appear in the agenda of any marking *M'* that is a successor of *M* until:

b.a) Transition *t* is still enabled but follows situation *a)* i.e. eventually, the application of some other transition makes *t* not enabled. The re-inclusion of *t* is determined by the *PN* enabling rule.

b.b) Transition *t* stays enabled but a *sensitive* event concerning this transition happens that leads to the reconsideration of the re-inclusion of *t.*

Situation *b.b)* needs an explanation of the concept of *sensitive events* in the partial state identified by the inputs and output places of a transition. In other words we must define what *sensitive events* mean and how they can be modelled.

In table 2 in chapter 3, we established the parallelism between a *PN* model and the enabling and firing rule with a state change operation in a STRIPS-like [Fikes 71] problem representation. This definition is extended to include a new term, *sensitive.*

Operator: *t*

Precondition: *cb-PN enabling rule*

SENSITIVE: $S \subset P$

Post condition effect: *Firing rule.*

Where the set *SENSITIVE* specifies a set of places that are directly related to the transition (they are either input or output places) and changes to which affect the context in which the transition can be fired. For example, consider the *PN* of *fig. 40*.

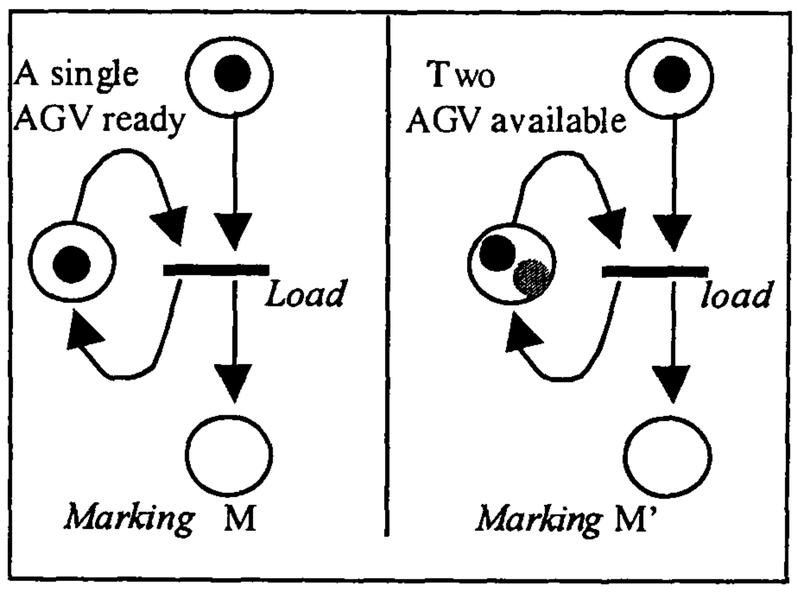


Fig. 40: a change in the partial state.

The figure represents a loading operation from an input buffer. To perform this operation, the system has two AGV's. In the left marking *M* of *fig 40*, transition *load* is enabled, although only one AGV is available at the time. The system can evolve to the right marking *M'* expressed in *fig. 40*, where *load* is still enabled; but now both AGV's are available. Marking *M'* represents a different context for *t* in the sense that two resources are available instead of the one available in *M*. If, at marking *M*, *load* was not fired, we should reconsider the decision at marking *M'* since a *change* in the place *AGV* has occurred. We considered that a *sensitive event* for a transition *t* is any change in the number of tokens observed in the places that represent resources (machines, buffers, etc.,).

There is a last question left to be considered. We need to determine when the decision to delay a transition is useless, i.e., the effect of delaying the application of an operation has no benefit. The following section, will present an implementation of the *CGS* branching method and will address the issues raised above.

3.2 *CGS* algorithm.

The branching factor of two first noted in *fig. 38* can be revised. *Fig 41* shows the search tree starting from a marking *M*. Notice that $M^2, M^4 \ldots M^{2n}$ is exactly the same as *M*. It is meaningless, from the point of view of systematic search, to store any of

these markings for further exploration. In fact $M^1, M^3 \ldots M^{2n+1}$, can be obtained by sequentially processing the list *Agenda(M)*.

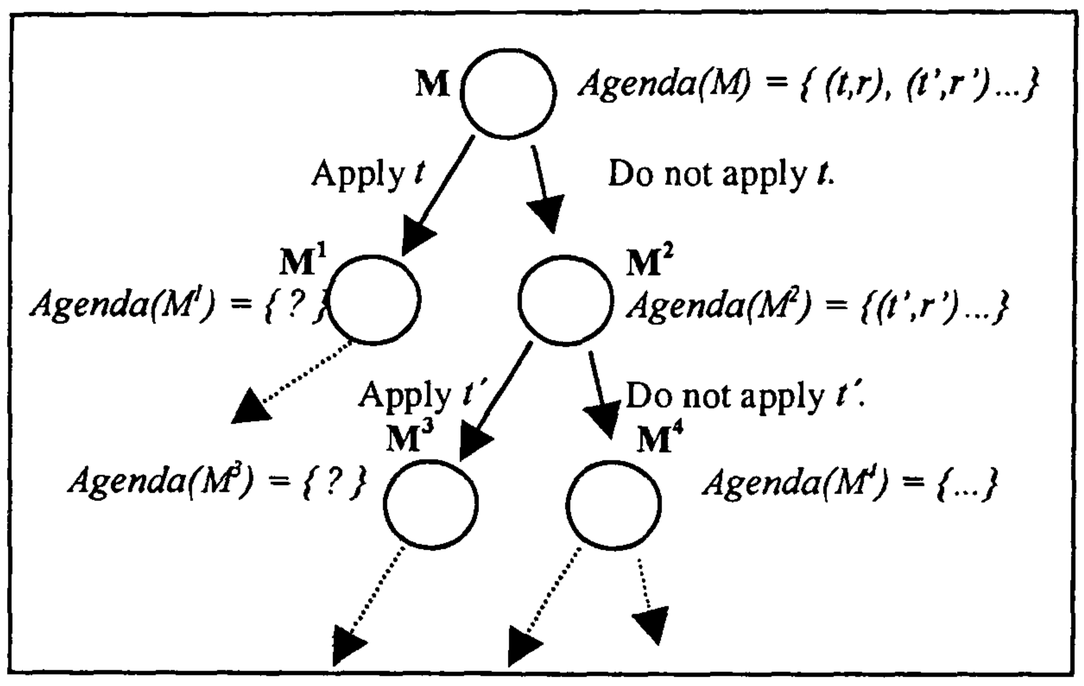


Fig- 41: Search tree generated.

The terms and concepts of *Agenda* and *Sensitive* are now formally defined:

For every transition t we define the set $Sensitive_t$ as $Sensitive_t = \{r \in R \mid I(r,t) \neq 0\}$ being R the set of resource places, and I the pre-incidence matrix of a *cb-PN*. Associated with any state of the search graph represented by a marking M we define two lists: *Agenda(*M) and *Delayed(M)* as follows. *Agenda(M),* as explained before, contains the set of transitions that are enabled at M and can still be fired to generate a new branch M'. The new list, *Delayed(M)* is formed by those transitions that, although enabled at *M,* have not been fired. Notice that the union of *Agenda(M)* and *Delayed(M)* equals the set of transitions enabled at *M. Agenda* contains pairs of the form *(t,r)* where t is an enabled transition and $r=c(t,M)$[18]. The elements in *Agenda* are ordered by the increasing magnitude of r. The list *Delayed* is also formed by pairs of elements *(t,r)* and is used to decide when the arbitrary delay of a transition is useless. To do this, we determine if no change occurs within the interval defined by $c(t,M) + \tau(t)$[19] for a transition t included in *Delayed.*

CGS (fig. 42) substitutes the standard branching procedure of *generate a new marking for every transition enabled.* For example, in our *cb-PN A** algorithm described in *chapter* 3[20].

---

[18] $c(t,M)$ is an alternate expression for $c(M,M')$ that is the cost of an arc in the search tree defined in page 70 . I.e, t is the transition that, if fired in M produces M'.

[19] $\tau(t)$ *is* defined as the delay associated with the system operation represented by t, either as the operation itself or as the transition marking the beginning of the operation.

[20] See chapter 3, figure 6, step 5.

Algorithm *CGS.*

Receives. *M* the current marking under exploration.

Computes. The set of markings successors of *M* and their associated *Agenda* and *Delayed* lists.

(1) while *Agenda(M)* is not *empty*
remove the first element *(t,r)* in *Agenda(M)*
generate the next marking *M'* as a result of firing *t* in *M*.
Agenda(M') = ∅; *Delayed(M')* = ∅;
for every pair *(t',r')* ∈ *Delayed(M)*
if *t'* is *enabled* at *M'* **and** *no change is observed in any place* $p \in$ **Sensitive**$_{t'}$
then include *(t',r'-c(M,t))* in *Delayed(M')*.
for every other transition *t'* *enabled* at *M'*.
if $t' \notin$ *Delayed(M')* **then** include *(t', c(M',t'))* in *Agenda(M')*.
if $\forall(t,r) \in$ *Delayed(M')* $r > 0$ **then**
if *M'* is *not a final* marking and *passes the test for similar markings* **then**
put *M'* in the list *UnExplored* for further exploration.
add the pair *(t,r')* to *Delayed(M)*. $r' = r + \tau(t)$ being
$\tau(t)$ the delay associated with the operation.
goto (1) .

Fig. 42: CGS algorithm.

Let *M* be the marking selected from *UnExplored* yielding the lowest *h(M)*. Let *Agenda(M)* be its associated agenda. The first pair *(t,r)* (i.e., the one that yields the smallest *r)* is removed from *Agenda(M)*. The marking *M'* from *(t,r)*, is obtained by firing *t*. *Agenda(M')* and *Delayed(M')* are built as follows: only the newly enabled transitions, or those delayed transitions in *Delayed(M)* for which a *change* has been observed in the places of $Sensitive_t$, are included in *Agenda(M')*. If no *change* is observed in the $Sensitive_t$ places of a delayed transition of *M* then the transition is still observed in *Delayed(M')* but its associated time information is actualised by subtracting the cost of applying *t* in *M*.

Next we determine whether the time variable *r* of any pair *(t,r)* of the elements of *Delayed* has reached 0. The rationale is that transition *t* was delayed in order to keep its resources available, but these resources have not been used by any other operation within the interval defined by $\tau(t)$ or new resources affecting *t* have become available (no change has been observed for $Sensitive_t$). Consequently, we consider that the

arbitrary delay of t is useless and the path associated with M is pruned since it represents a non-active. If this does not happen, then M' is included in *UnExplored* for further exploration if it is not the goal marking and it passes the checking for duplicate markings,

Finally, *Delayed(M)* is created by adding the pair (t,r'). By doing this, we consider that transition t is to be delayed and fix its *deadline* r' as the cost of performing the operation that t represents.

3.2 Example application.

To illustrate the algorithm, consider the *PN* model of *Fig. 43*. The system has two jobs with two operations each, and two machines.

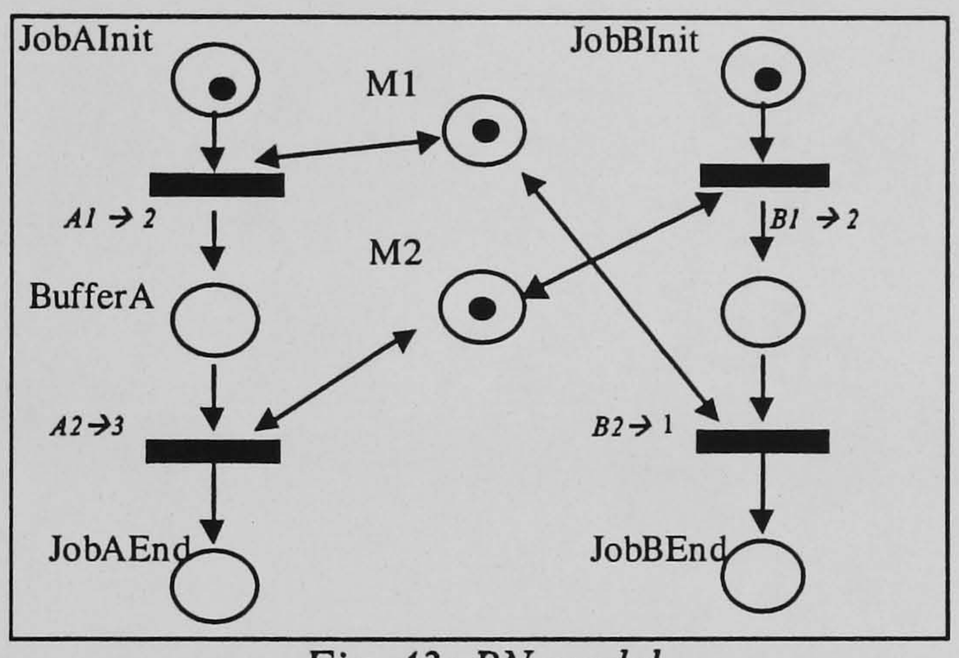


Fig. 43: PN model.

Fig. 44 shows the search tree. At every marking M, we consider as a possible successor the firing of any t enabled at M. *Fig. 45* shows the search tree generated with the control based in the structure *Agenda()* of the *CGS* algorithm. The contents of *Agenda( )* is given for each node in the figure. *CGS* generates 12 nodes, as opposed to 18 for the original approach. Note that the search tree is generated with a full enumerative search algorithm (such as *breadth-first*) and no pruning is performed.

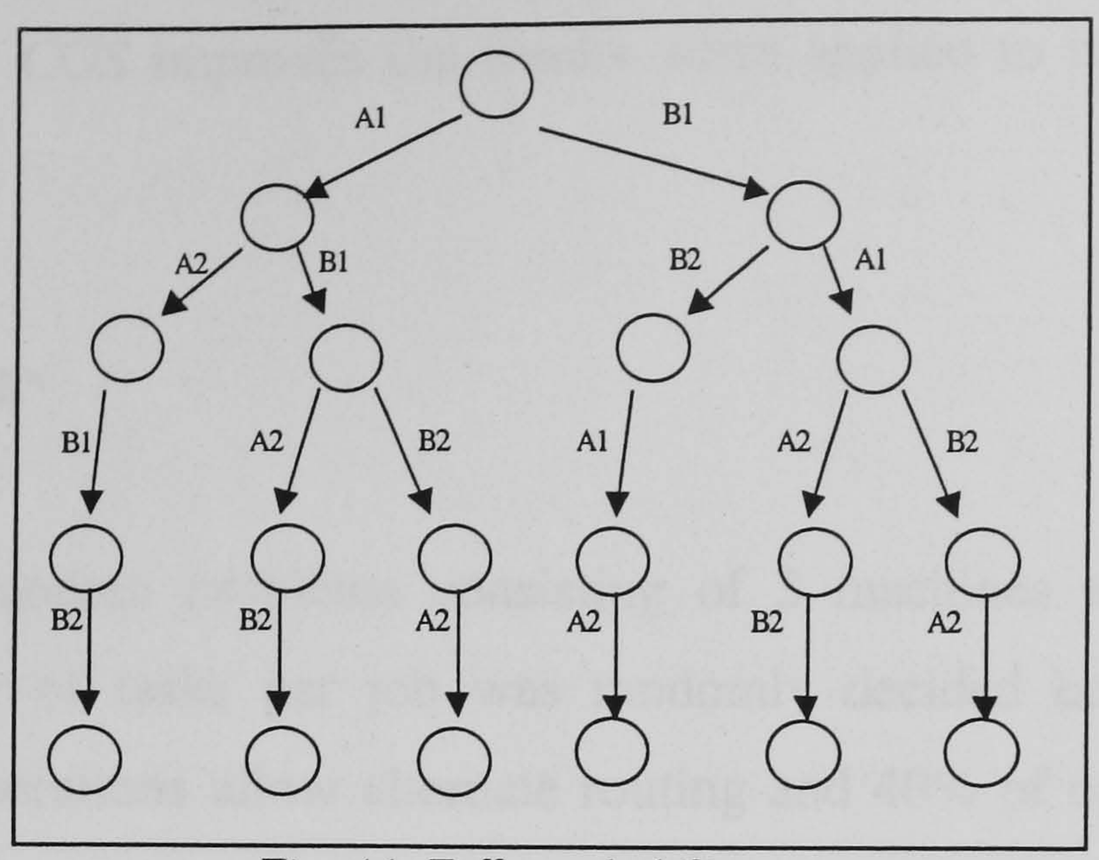


Fig. 44: Full reachability tree.

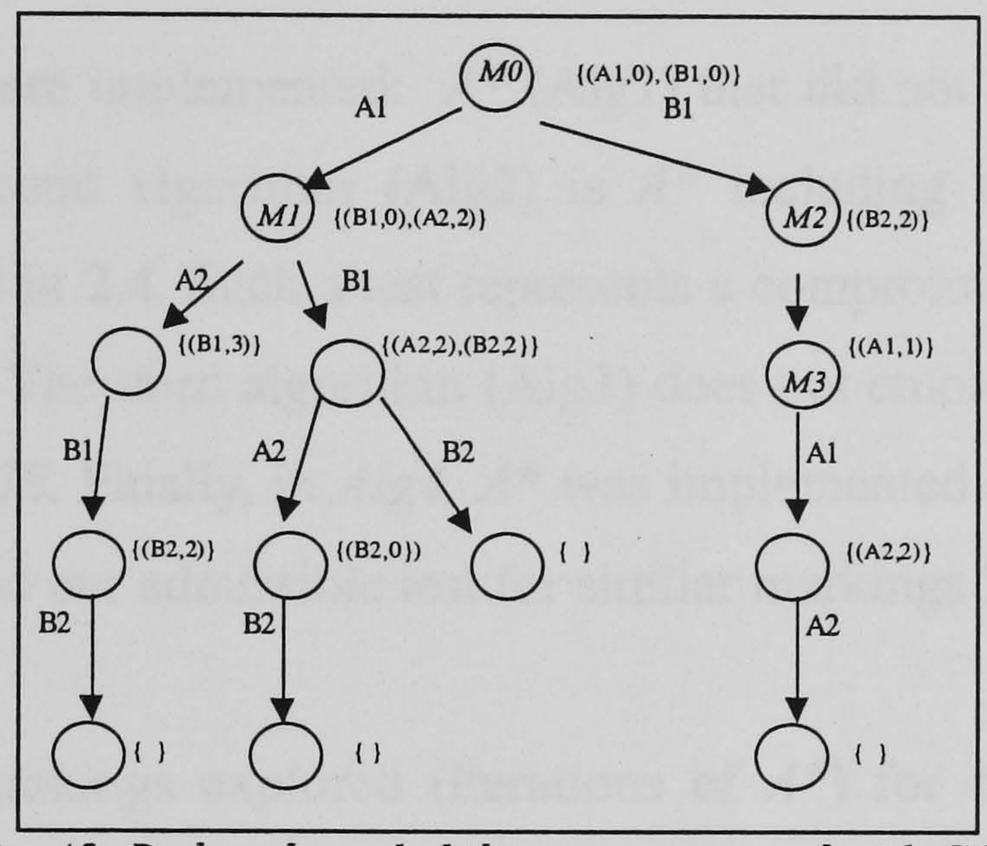


Fig. 45: Reduced reachability tree generated with CGS.

The initial M_0 contains two transitions from the *Agenda, A1* and *B1*. The second branch (M_2) results after firing *B1* and ignoring *A1*. Note that *A1* is still enabled but not included in *Agenda*(M_2). The only choice is to fire *B2*. Since *B2* uses *R1*, A1 becomes enabled again in M_3.

4 Experimental results.

In order to study the performance of the *CGS* approach, three experimental tests were carried out. The first one studies *CGS* as an alternative to substitute the pruning of paths by comparison of similar markings in *A** that we studied in section 2 of this chapter. The second studies the performance of *CGS* in branch and bound algorithms where it is not possible to apply such test for similar markings. Finally the last

experiment shows how *CGS* improves the results when applied to incomplete search procedures.

4.1 Integration with *A**

A set of 200 random problems consisting of 3 machines and 3 jobs were generated. The number of tasks per job was randomly decided between 2 and 3, additionally, 75% of operations allow alternate routing and 40% of operations require more than one machine.

Four algorithms were implemented: *A** (Alg1) that did not include any test for similar markings, the second algorithm (Alg2) is *A** including the test for similar markings *Test 3* presented in 2.4. Such a test represents a compromise between pruning efficiency and optimality. The third algorithm (Alg3) does not employ a test for similar marking but integrates *CGS*. Finally, in *Alg4, A** was implemented employing both the branching scheme *CGS* and our admissible test for similar markings *Test 1*.

The number of markings explored (iterations of *A**) for each problem when solved with *Alg1* was compared with the number of markings explored with the rest of algorithms. The comparison is expressed as the relative difference:

$$\frac{\textit{iterations}\ \text{Alg1} - \textit{iterations}\ \text{Alg}\ x}{\textit{iterations}\ \text{Alg1}}\ \% \qquad (1)$$

Table 10 provides descriptive statistics for the relative difference between markings explored by *Alg2, Alg3* and *Alg4* versus *Alg1*. Also, the percentage of problems solved optimally is shown for each algorithm.

	% difference of markings explored from *Alg1*				% problems solved optimally
	Min	Max	Mean	Std. Dev.	
Alg2	64.4	99.8	96.2	5.1	66%
Alg3	13.1	92.9	76.0	10.5	100%
Alg4	22.7	98.1	82.0	10.9	100%

Table 10. Comparative results.

The first interesting observation is that the relative difference of markings explored (close to 77%) indicates a greater search reduction than if we implement a pure breadth-search, (where our experiments showed a reduction of mean 44%). An explanation for this may be that *CGS* is not generating markings whose heuristic estimates do not vary significantly from each other[21]. A reduction of these equally heuristically evaluated nodes results in a relaxation of the *breadth-first* aspects of *A** with an admissible heuristic function. *Fig. 47* shows the frequency histogram for 1500 problems of the same characteristics if solved with *A** (*Alg1*) versus *A** incorporating *CGS (Alg3)*.

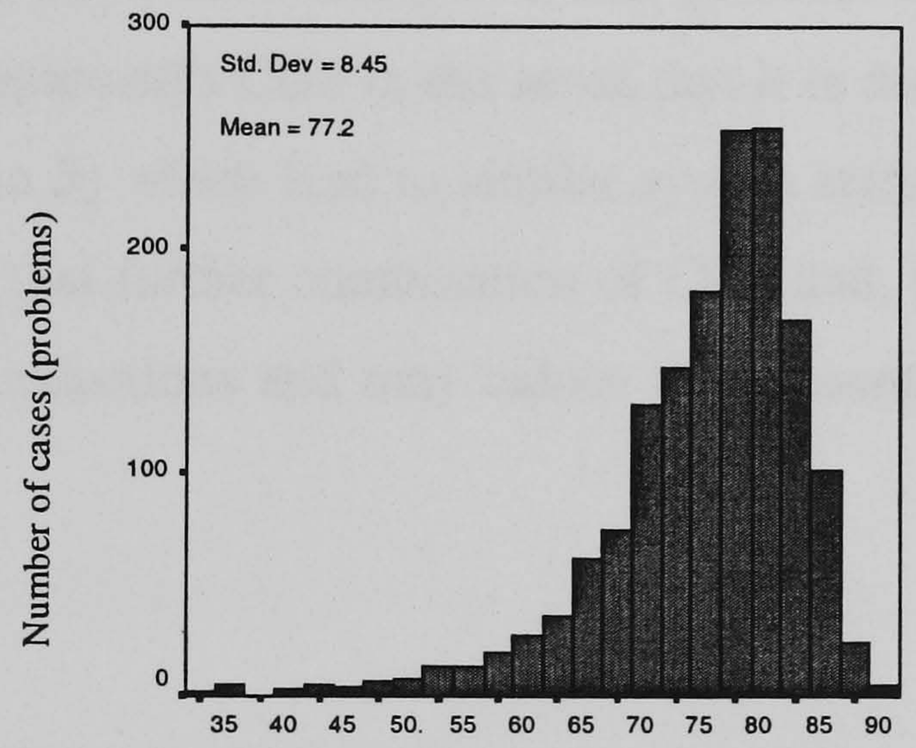


Fig. 47: % relative difference of markings explored

A second interesting observation: theoretically, a *safe* pruning procedure such as *Test 1* will partially[22] achieve the same effect as *CGS*, i.e., it will prune paths that lead to the same state but with a longer (or equal) makespan than a previous path. However, if we compare the results obtained in section 2.4 for *Test 1* the search reduction is on

[21] Which happens for markings which are permutations of concurrent transitions, or non-active schedules and their associated active-schedule.

[22] We say partially since it is not always the case that a positive comparison results in discarding one of the markings (see algorithm in section 2.3).

27.7. This performance is noticeably inferior to *Alg3*. However, as we studied in section 2.4, one may increase the effectiveness of *Test 1* by employing heuristic tests such as *Test 3 (Alg2)*. Unfortunately, this introduces the disadvantage of inadmissibility. Any of these techniques produce an overhead, since markings and paths are actually generated, stored and list-search and comparison algorithms are needed.

Apart from achieving a considerable search reduction embedding *CGS* as a branching scheme, the approach appears to be, in practice, admissible. Note that now there is no need to maintain a *UnExplored* list and the computational overhead mentioned above is not produced. It is worth noting that the reduction of the search effort is achieved without appreciable additional computational cost. The *CGS* algorithm has the same asymptotic linear cost as the normal procedure used to identify enabled transitions, the memory expenses due to the storage of the *Agenda* list are small compared to the search space reduction, which results in a reduction of the storage required for the nodes.

Finally, a combination of an admissible test *(Test 1)* with the branching scheme (*Alg4*) gets close to the performance of *Alg2* but also provides the optimal solution. This indicates that *Test1* complements *CGS* in the sense that it is detecting those *other cases* (see beginning of section 3) which lead to similar system status which are not covered by *CGS*. This indicates that further combination of *CGS* and, for example *Test 3*, may produce greater search reductions and may reduce the amount of problems not solved optimally.

4.2. Integration with *Branch & Bound*.

CGS gains relevance for the scheduling approaches based on *Branch & Bound* described in [Lloyd 95] [Chen 94] [Abdallah 98]. Unlike *A** a *B&B* algorithm bases its strategy on the progressive optimisation of a candidate solution. The memory requirements are kept bounded at the cost of not maintaining a list of previously visited markings, which means it is impossible to fully perform comparison tests between markings in an *A** manner. Often ([Lloyd 95], [Chen 94] [Abdallah 98]) a second test is applied that checks whether the new marking obtained is in the current solution path. If

the time for the new marking is greater than the one previously reached then the branch is ignored. This is not very effective, since few markings are available for comparison.

The same set of 200 problems described in section 4.2 was first solved with a *B&B* approach that uses the admissible heuristic function employed in *A** as the lower bound. Each problem was also solved with the same *B&B* algorithm but including *CGS*. *Table 11* shows the descriptive statistics of the relative difference of the number of markings explored between both algorithms calculated as:

$$\frac{\textit{iterations}\ \text{B\&B} - \textit{iterations}\ \text{B\&B-IGC}}{\textit{iterations}\ \text{B\&B}}\ \% \qquad (2)$$

	% difference of markings explored			
	Min	Max	Mean	Std. Dev.
% rel. diff	44.7	93.9	74.9	10.3

Table 11. % difference of markings explored.

Results show a considerable reduction of the search space generated. Notice that final markings representing solution paths were not considered. Again 100% of problems where solved optimally.

4.3 Integration with an incomplete search procedure.

This experiment aims to study the effect of *CGS* with incomplete search algorithms, where candidate markings are arbitrarily pruned based on heuristic information. One such algorithm widely employed is *Beam-Search* (see [Shi 91] for example). The objective is to search a number of potentially optimal decision paths in parallel, eliminating the need for backtracking and to obtain a solution quickly.

Beam-Search will be presented in the following chapter, for now we simply observe that *Beam-Search* represents a dramatic decision in terms of both avoiding backtracking and limiting the number of paths for further exploration. It is important that markings representing different paths to achieve the same, schedule permutations

and non-active schedules are either not generated or identified as soon as possible so that they are not included in the search. The integration of the branching scheme within *Beam-Search* successfully achieved this objective as the experimental results will show.

A set of 200 random problems consisting of 10 machines, 10 jobs, and 10 tasks per job was generated. The rest of the settings were the same as for the set of problems of section 4.2. Each problem was first solved with *Beam Search* set to perform 2000 search iterations. The criterion for selecting markings for further exploration is the heuristic function $h_{RCR(m)}$. *CGS* was then incorporated into the algorithm and the problem was solved again (the number of iterations still set to 2000). The solutions obtained with each algorithm were compared in terms of the relative difference between their makespans calculated as:

$$\frac{\text{makespan without } \textit{Branching scheme} - \text{makespan } \underline{\textbf{with}} \textit{ Branching scheme}}{\text{makespan without } \textit{Branching scheme}} \% \quad (3)$$

Fig. 48 shows the frequency histogram for the value of (3) for the set of 200 problems. A better solution was obtained for 70% of problems when employing the branching scheme.

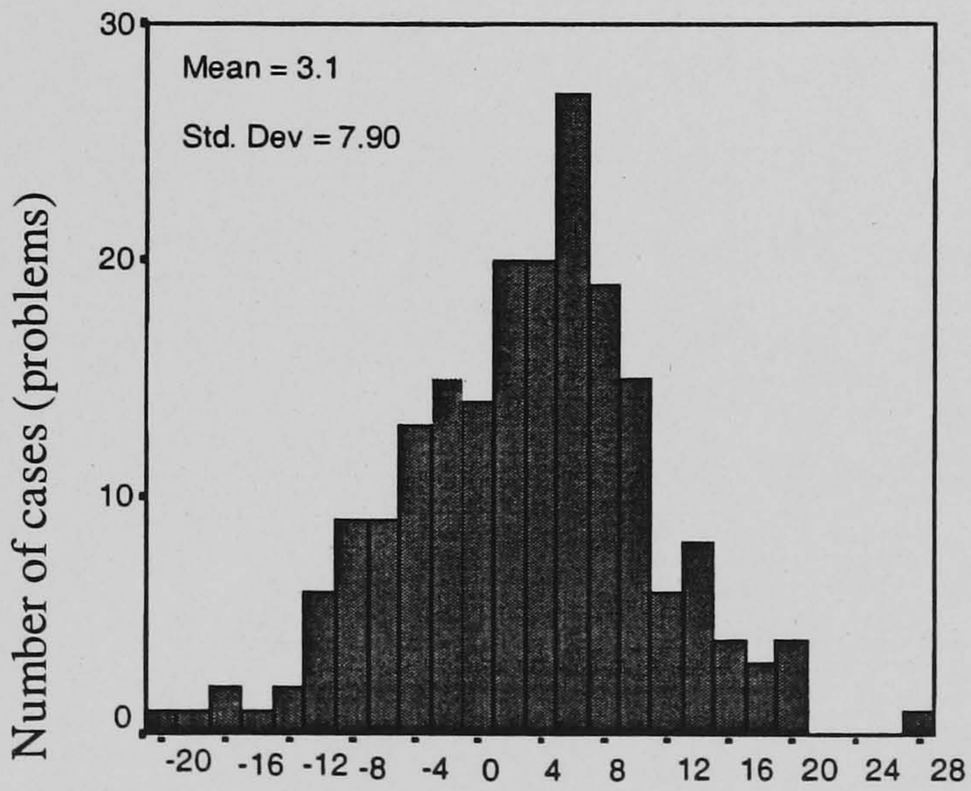


Fig 48. Histogram for the relative difference of makespans.

5. Summary.

This chapter has studied the search space associated with a *PN* model of *FMS* descriptions and has proposed different techniques to reduce the search effort, a matter that has received little interest in the work consulted. The first part of the chapter has studied different pruning methods based on comparison with previously visited paths. Such techniques allow alternatives from pure admissible tests to sub-optimal but very effective ones. However these tests are only effective if a complete algorithm is employed and require storage of and comparison with previously explored markings. Since we intend to provide a incomplete search procedure, we explored why different paths reach a similar system status and proposed a *PN-based* branching scheme called *Controlled Generation of Successors* that avoids the generation of branches that do not yield better schedules. Finally, experiments reported in [Reyes 00b] have shown that the branching scheme considerably reduces the search effort for optimal *B&B* algorithms and achieves similar search reduction to the standard pruning methods of *A**. Also, results indicate that when applying the approach to an incomplete search procedure, it produces useful results in terms of enhancement of the quality of the solution obtained.

Chapter 6. PN-based hybrid heuristic search.

1 Introduction.

This chapter is about the development and study of a hybrid search algorithm to be applied to large *FMS* scheduling problems.

In chapter 4 we concentrated efforts on developing a heuristic cost estimation, namely h_{RCR}, which was obtained after the analysis of the *cb-PN* model which is a function of both the *PN* structure and dynamic information expressed by the *cb-PN* marking. Results showed that the search effort was considerable reduced if compared with a pure best first search strategy and previously proposed heuristics [Yim 96] [Sun 94] [Lee 94].

In chapter 5 we dealt with techniques to identify non-promising paths and showed that it was possible to dramatically reduce the search effort at little cost. Even with such reduction, the *cb-PN-A** is only applicable to small problem instances. To see this, consider the following problem from [Lee 94]. It consists of five jobs, three machines, and four tasks per job. The total number of operations to schedule is 20. The *Cb-PN A** algorithm solved optimally after exploring 20,666 markings, from a total of 150,730 markings generated. In other words, 130,000 intermediate states were generated, stored in *UnExplored* but never expanded. Execution took about an hour on a PC Pentium II at 300 Mhz.

Apart from the considerably amount of memory needed, the problem is the cost of asserting a new marking, in order, in the list of candidate markings. This operation has a cost of *O(n)*. We are aware that algorithm optimisations are possible, but this does not solve the main problem described in chapter 3, i.e., that the NP-hard nature of the reachability graph generator did not allow the application of an admissible *A** algorithm without, sooner or later, overwhelming computational cost.

We believe that it is possible to create a branch and bound approach that will not suffer from this effect and that employs a *PN* based heuristic rule to determine the next transition to fire. *CGS* can be exploited to reduce the search as we showed in the last chapter.

B&B approaches have been used traditionally to solve production scheduling problems represented by a mathematical model. For example, [Carlier 89] solved,

optimally, a ten job, ten machine Job Shop problem proposed 30 years earlier [Muth 63]. The problem is that an *FMS* not only increases *JSS* complexity because of the existence of alternate paths and operations, but also involves lot-sizes which increase the total number of operations to be scheduled. One possibility is to use incomplete or truncated *B&B* algorithms, for example [Chu 92]. This algorithm is stopped when a given number of consecutive branching steps do not improve the makespan. Approaches of this kind tend to confirm our intuition that *B&B* fundamentally will experience the same problem as *A**, i.e the existence of a large number of equally promising paths. Nevertheless, in chapter 3 we justified the use of an *A** approach rather than a *B&B* method since *B&B* limits the application of *PN* based information to guide the search. Heuristic decisions are restricted to the next transition to fire at the current node as the search is governed by a depth-first strategy.

In chapter 3 we also stated that in order to obtain an affordable *A*-like* search algorithm that guarantees a sub-optimal but acceptable solution, two general strategies were possible:

The first adds a *depth-first* component to the heuristic function to prevent excessive backtracking. The experimental tests carried out in chapter 4 seem to confirm that using a non admissible heuristic function as a way to reduce the search effort caused non-admissibility and resulted in difficult parameter tuning to control the search effort. Nevertheless, in the conclusion to chapter 3, we noted that if the exploitation of *PN* information is used to guide search, the search effort must be controlled by an *incomplete* algorithm where an admissible and *pure PN*-based heuristic function is applied.

This chapter is organised as follows. Section 2 reviews the literature on affordable *PN* based heuristic search algorithms for *PN* models of *FMS,* and presents their deficiencies. Section 3 analyses the potential of two arbitrary pruning approaches by studying two algorithms (*HST* and *DWS).* A revision of these methods leads, in section 4, to a final algorithm *DLSS** that overcomes the difficulties of previous approaches. Experimental results in section 5 will suggest that the algorithm is polynomial, and produces interesting results both in terms of optimality and in comparison to other *PN* based heuristic search algorithms.

2 Review of previous approaches.

An obvious methodology to avoid the exponential growth of the number of markings contained in the *UnExplored* list, is to limit the number of nodes that this list might contain. This is an approach that has been independently used by three different authors [Yim 94] [Sun 96] and [Inaba 98]. [Sun 96] calls this algorithm limited expansion *A**. At the end of each iteration of the *A** algorithm of [Lee 94], if the number of nodes in *UnExplored* exceeds a given number *b*, the list is truncated and only the *best b* nodes are kept. The criterion for rejection is based on the estimation function *f(m)*.

The main attraction of this method is that the maximum number of markings that are explored to obtain the first solution is bounded by *O(b*n),* where *b* is the size of the *UnExplored list* and *n* is the depth of the goal marking.

Unfortunately, this method presents problems with an admissible heuristic function. In general terms, the search may fall into a backtracking free *breadth-first* search. In other words, the *UnExplored* list will be completed with markings of depth *l* and from there, new markings of depth *l+1* will be generated. Notice that, in the worst case, a single successor of the current marking may be included when the list *UnExplored* is full.

To illustrate such behaviour, consider the simple search tree created with a branching factor of two. Suppose that the general rule is that $f(m')>f(m)\ \forall m,m' \mid depth(m')>depth(m)$. Although this supposition may seem to be, a priori, too strong, note that at a certain given of the search, *UnExplored* will contain the *b* markings yielding the lowest values of *f(m)* and we have already shown experimentally that the number of equally heuristically valued markings grows exponentially, which causes *breadth-first* behaviour. Hence we can reasonably assume that (except for very large values of *b),* the variance of *f(m)* within the limited *UnExplored* list will be rather small. Also, as stated in [Nilsson 82] an admissible heuristic function usually satisfies $h(M) - h(M') \leq c(M,M)$ and consequently $f(M') \geq f(M)$ for the general case.

Fig. 49 shows the evolution of the contents of the *UnExplored* list for the artificial search tree mentioned above as markings are explored by a limited expansion *cb-NET A** algorithm. The size of *UnExplored* is set to *8* for illustrative purposes. A

number represents a marking of this depth, '-' represents an empty space in the *UnExplored* list.

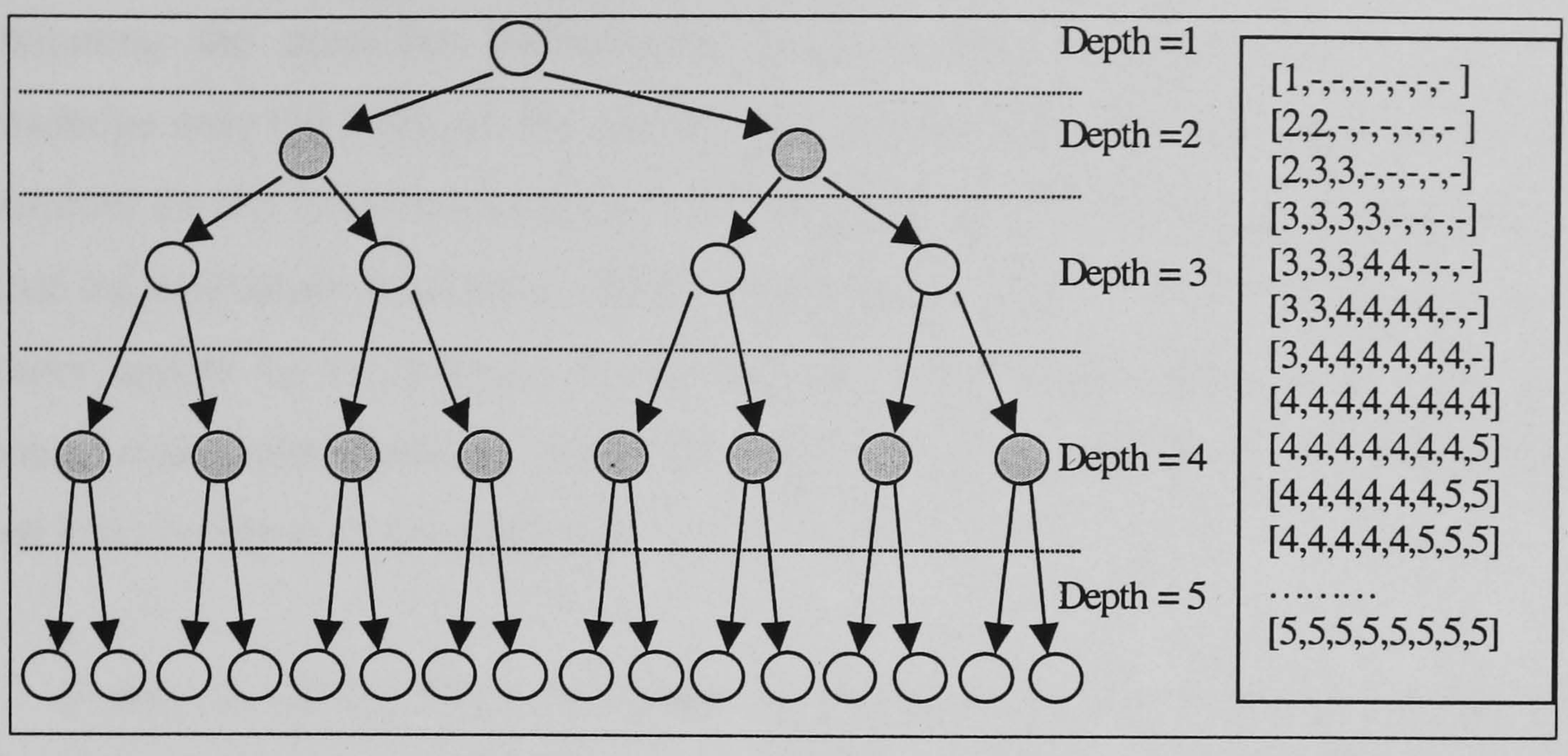


Fig. 49: Evolution of the contents of the UnExplored list (right) as limited expansion A explores the tree (left).*

Only if the heuristic function is an extremely good estimate (which is difficult for a general *FMS*) and the *UnExplored* list is large will this situation be avoided. In general, a limited *UnExplored* is likely to contain markings of near depth, which limits the recovery distance in terms of backtracking. Again, as in the pure stage search mentioned before, this may have unwanted effects in terms of solution quality due to local optima [23].

The implication for the search of such limited expansion is poorly addressed in the literature. For example, [Yim 96] considers a limitation of the list *UnExplored* of 50, but they also employ a heuristic function with a major component of depth-first search. Not surprisingly, they conclude that there is no difference between a pure *A** and *A** with this limitation. This is obvious, since analysis of [Lee 94] demonstrates that the depth-first behaviour almost eliminates backtracking and consequently the *breadth-first* effect. This is a clear example of what happens if both the algorithm and the heuristic function pursue the same objective of focusing the search quickly on an optimum solution. In both [Sun 94] and [Inaba 98], the *Limited Expansion A** is again combined with heuristic functions that essentially follow [Lee 94]. Unfortunately, the experimental results for both approaches are restricted to a single case study. In some ways this last method is to *A** what *beam* search is to pure *breadth-first search*. The

[23] We show this experimentally in section 5.8.

Beam Search algorithm (see [Ow 88]) has been extensively applied to production scheduling problems (particularly on-line scenarios), as we reviewed in chapter 2. The objective is to search a limited number of potentially optimal decision paths in parallel, eliminating the need for backtracking and to obtain a solution quickly. To our knowledge only the work of Shi and Sekiguchi [Shi 1991] implements a *Beam Search* procedure for the on-line scheduling of *FMS* based on *PN* modelling. A major problem is that the evaluation function to select the best nodes is not described in the paper. The authors appear to be utilising *Beam Search* as a decision module for solving the resource assignment problem, whilst the sequence of jobs seems to be determined by a three level heuristic dispatching rule.

Finally, in [Xiong 98], a two-phase algorithm combining *Branch & Bound* and *A** is presented as means of dealing with complexity problems in search spaces defined by a *PN* model of *FMS*. Two hybrid algorithms are proposed in this work:

- *B&B-A** : A depth-first search is performed until a maximum depth is found. Then, the last marking found is used as a root node for an *A** based search. The *A** search employs an admissible heuristic function based on the longest remaining job.
- *A*-B&B*: The *A** algorithm finds the best marking at a specific depth, and then *B&B* is used to find a leaf.

The authors report better results with the *B&B-A** approach. The rationale is that this overcomes the intractability of the problem by progressing quickly to the end of the search tree using *B&B* and then using *A** to find the best leaf starting with that path. The idea is interesting but we have many criticisms. First, the *PN* model of the *JSS* guarantees that any sequence of transitions is a feasible schedule (excluding deadlocks). The paper specifies clearly that the algorithm stops when the first solution is found, hence, there is no backtracking capability and the *B&B* approach is actually a depth first irrevocable strategy since no heuristic information seems to be employed at this stage. This yields poor results in terms of the quality solution found.

Nevertheless, since pure *A** search does not admit large search trees, the depth where *depth-first* search changes to *A** must be close to the depth for the solution

marking. Consequently, for medium-large problems, the *depth-first* irrevocable search is predominant over *A**, and thus is likely to produce weak results.

From the reviewed works, we conclude that improvements can be made. We intend to avoid algorithms with severe backtracking limitations (*Beam Search)* and algorithms that critically limit selection capacity based on *PN* heuristics *(Depth-first search and Branch&Bound)*. The following section will present two algorithms that use pruning techniques based on the search space defined by a *PN*.

3 *PN* based incomplete search. *HST* and *DWS* algorithms.

An *incomplete* algorithm results when constraints are imposed on any of the two dimensions of any systematic *PN* reachability-tree-based search algorithm. This is obvious if we consider [Pearl 84]:

- Scope of selection: the capability of an algorithm to consider all the possible fireable transitions at the current marking. A heuristic algorithm based on a dispatching rule only considers one conflicting transition for firing and no other possible choices are considered.
- Backtracking capability: the capability of the algorithm to return to previously unexplored paths in the *PN* reachability tree, *cb-PN A** has full backtracking capability because of the management of the *UnExplored* list. An algorithm with no backtracking capability is said to adopt an irreversible strategy.

From the full range of capabilities of *A** in selection or backtracking, a compromise situation can be found that allows dramatic reduction of the search effort whilst ensuring a useful degree of optimality.

3.1 Heuristic selection of transitions: *HST.*

For any marking *M*, there exist a number of possible transitions that may be applied. Additionally, due to flexibility, the number of conflicting transitions that can be found is typically large. As they are conflicting transitions, the firing of one will

block the firing of the other. This mutual exclusion is produced either because they compete for a resource, or because they represent alternate routings to achieve a sub-task. Whatever the case, the fact is that the firing of conflicting transitions produces two different markings that will generally lead to different schedules in terms of makespan. The *A** algorithm stores these markings in order to backtrack whenever *f(M)* indicates a more promising path. An immediate way to reduce the scope of selection is to implement one or several dispatching rules or heuristic rules that can be used to discriminate amongst all enabled transitions for a specific marking. The most promising transitions (one or more) in terms of these heuristic dispatching rules are selected, whilst the rest are discarded, and *A** continues.

As we stated in chapter 2, the on-line conflict resolution within production scheduling problems has been widely solved with the use of dispatching or scheduling rules associated with buffers: *FIFO, LIFO, job-priority* or to resources: *SI, LI, SR, LR*[24] (see [Gupta 89] for a reference on FMS scheduling rules). However, as stated in [Basnet 94] [Harmonosky 91], a fixed policy encounters problems with the degree of flexibility of *FMS*. Additionally, its computational cost is constant and this prevents a decision module from making use of extra computational time to improve the quality of the decision.

The fact is that *PN* easily allow the implementation of simple dispatching rules based on the dynamic information expressed by the marking and the *PN* structure. The irreversible and unreliable decisions made by considering a single transition can be corrected by considering several choices and allowing backtracking. In other words, heuristic dispatching rules can be integrated with a general graph search procedure allowing us to reject the application of operations that seem to be very unpromising. This effectively limits the possible alternatives generated by *A**.

In this section we propose an approach, *HST* (Heuristic Selection of Transitions) to bound the *scope* of evaluation of the *cb-PN A** algorithm based on the use of traditional dispatching rules adapted to *PN* dynamics.

An alternate approach that integrates basic dispatching rules and systematic search algorithms is presented in [Abdallah 98]. They propose a *branch & bound* approach where the next transition to be fired is determined by a two level heuristic that implements *SRT* (shorter remaining time) and *SPT* (shorter processing time). Also in

---

[24] Shortest operation time, largest operation time, shortest remaining processing time, largest remaining processing time respectively.

[Elmekkawy 98] experimental results are reported for the employment of heuristic rules in transition firing selection embedded in a *Branch and Bound* approach. The effect on makespan as well as on the computing CPU time was studied. Their conclusion is that the dispatching rule that provides the best solution, in terms of makespan, depends on the characteristics of the *FMS*.

The approach is not seen in *A** implementations, and the reason is obvious: a best first based algorithm uses the heuristic estimation *f(M)* to determine the next marking to be explored.

Our intention is not to use dispatching rules for heuristic search guidance as in [Abdallah 98]. We intend to identify those transitions that will rarely be selected for firing by a collection of well-known and effective dispatching rules. The rationale is that they indicate a bad choice (very large operation time for example) and so they can be rejected without excessive loss of optimality and yet still achieve a reduction of the search effort.

We studied the following heuristic rule k: $T \times M \rightarrow R^{+}$ which is defined as

$$k(t,M) = sj(t,M) + \tau(t)$$

$sj(t,M)$ models the *SRT* rule and is the minimum time needed for all the tokens required by transition t to become available in M. $\tau(t)$ is the delay associated with the transition t and models a kind of *SPT* rule.

The methodology is integrated into the *cb-PN A** algorithm as follows. At each iteration of the algorithm the set E containing the transitions that are enabled under the current marking M is created. E is ordered in the increasing magnitude of $k(t,M)$ for any transition t of E. Then, the set E is truncated keeping the N first transitions.

We conducted the following experiment: 30 random problems with 3 machines, 4 jobs and between 2 and 3 tasks per job. The number of alternatives for each operation was randomly selected between 1 and 3. Each problem was solved for values of $N = 1,2,\ldots, 12$.

Notice that when $N = 1$, the algorithm corresponds to an irrevocable strategy based on the *SRT/SPT* dispatching policies. The maximum number of transitions that can be applied at each marking is bounded by the number of parts in the system multiplied by

the maximum number of alternate routes for an operation. Thus all possible transitions are considered, and the optimum solution is obtained, when N is 12.

Fig. 50 (left) shows the mean relative difference of the solution obtained when *N=[1...11]* with the optimum solution obtained for *N=12*. On the other hand, *fig 50* (right) shows the evolution of the search effort, in terms of the number of markings explored by each setting of N.

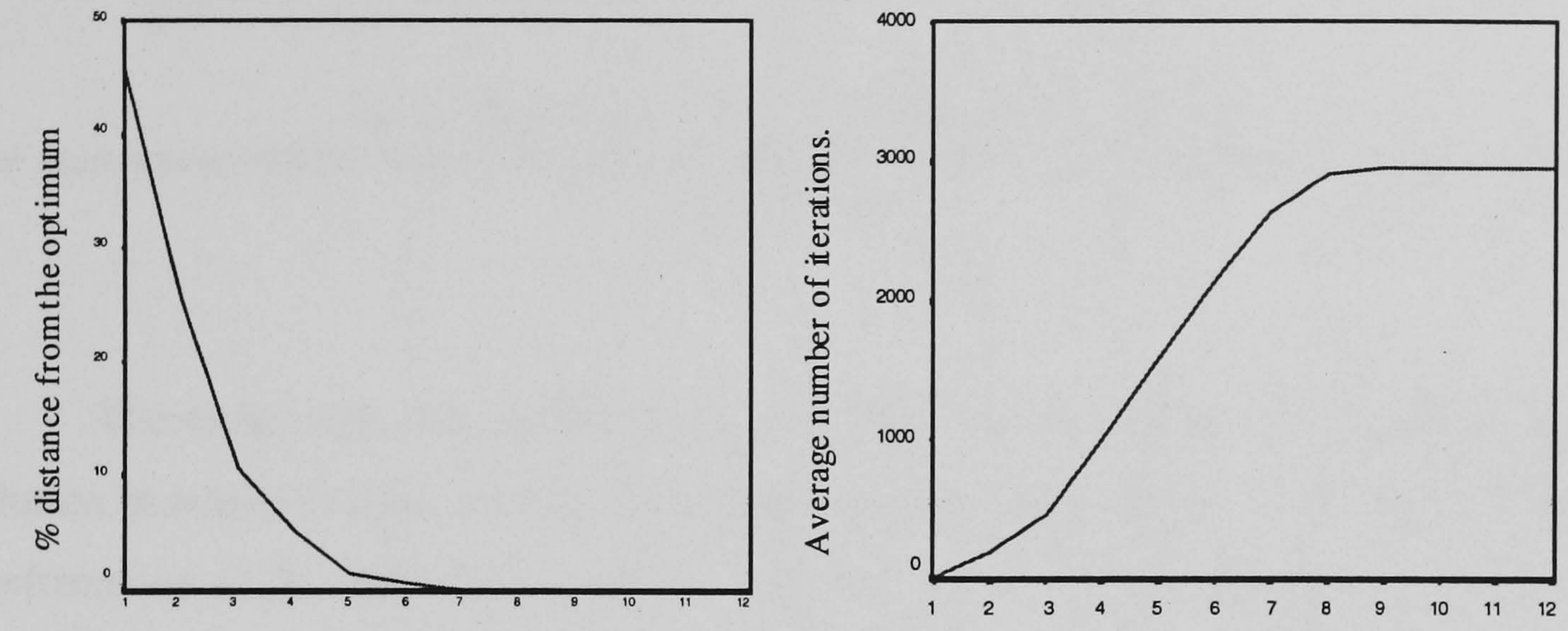


Fig. 50: Average % relative difference from the optimum (left) and average number of nodes explored (right) with respect to N.

Note that convergence on the best solution (optimality) obtained for *N = 12* is quicker than the convergence of the search effort expressed as the number of markings. E.g., for *N=5,* the relative difference of the solution obtained with the optimum is around 1.25% but on average, it explores 53% of the total markings explored when *N=12.*

The second study includes the test for similar marking (*Test3*) proposed in chapter 5 in order to evaluate the performance of *HST* for larger problems with more search space pruning[25]. The experiment consisted of a set of 100 random *FMS* descriptions comprising 5 jobs, and between 2 and 7 tasks per job. The maximum number of alternatives for each operation was 3. Each problem was solved employing *HST* for N (maximum number of successors per node) between *[1...15]*.

The plots show the evolution of the mean for the relative difference of the best solution obtained for the theoretical minimum makespan calculated as $h^*(m_0)$ *(fig. 51 left),* and the number of nodes explored *(fig. 51 right),* as in the previous experiment.

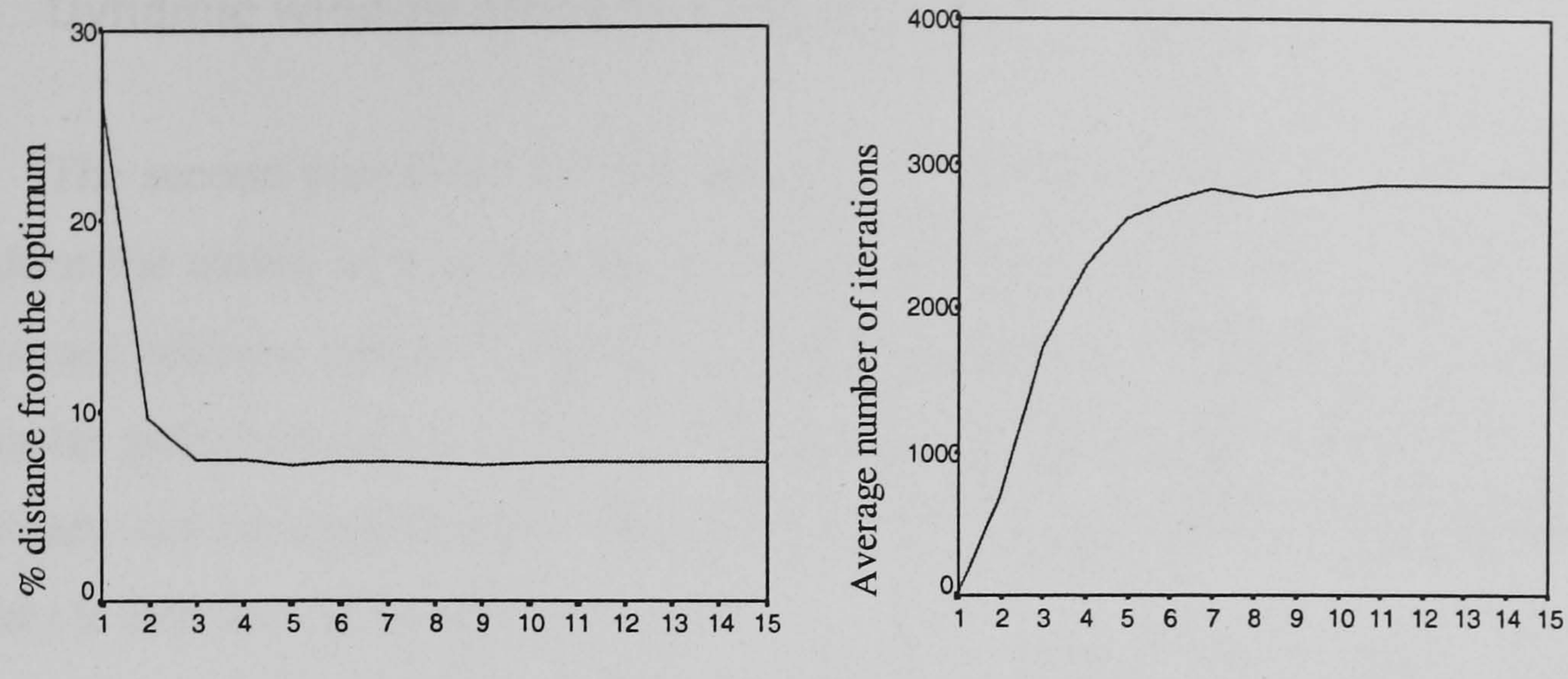


Fig. 51: (left) Relative difference of the solution obtained for each setting with the theoretical minimum makespan and (right) average number of nodes explored with respect to *N*.

The most interesting effect observed is that the convergence of the quality of the solution is reached faster than the convergence of the search effort. This provides some confirmation of the effectiveness of the approach in eliminating unpromising markings.

Finally we studied a second heuristic-dispatching function $k_2(t)$. The motivation behind this was to obtain a dispatching policy that adjusted better to the characteristics of the *CGS* branching method. $k_2(t)$ is informally defined as $k_2(t,M) = sj(t,M) + r'$. r' models how soon we will be able to apply the next operation and is calculated as the average delay time of unavailable tokens in the marking M', resulting in the application of t in M. Results indicated slightly better results for k_2 with respect to the previous k only if the value of N is lower.

The interest of these results is that it seems possible to avoid the generation of paths, based on *PN* based heuristic dispatching rules. These paths are not likely to lead to the optimum solution. Obviously, the approach does not solve the complexity problem for large *FMS*, but it is an interesting arbitrary pruning method to be considered and integrated within other approaches; for example, the one presented in the next section.

---

[25] *Test3* is a non admissible test that may prune paths leading to the optimum solution.

3.2 Dynamic window stage search algorithm: *DWS**.

The second possibility we discussed for achieving effective search reduction is to reduce the ability of a systematic search procedure to backtrack. In other words, if one cannot afford a pure A^* strategy, it may be desirable to take a risk and prune the tree so far generated by the search in order to allow deepening. Many variants of these techniques can be implemented. They all attempt to avoid backtracking over sub-trees unlikely to improve optimality.

A pruning technique that has gained some popularity is called *stage search* [Pearl 84]. Translating this to a *PN* domain, instead of maintaining the entire reachability tree generated by A^*, only the most promising sub-tree is retained. At the end of each storage reclamation stage, a subset of markings not yet explored is kept. The best sub-schedules of these markings are remembered and the rest of the markings are discarded. The search continues as normal until the storage allotment is again exhausted, which again forces node selection. Of course, optimality is not guaranteed.

In our *PN* based scheduling context, we can think about the effects that early decisions (operations) might have on operations very distant in sequence. If we assume that the application of an operation will only affect the schedule locally, we can risk forgetting the earliest choices still unexplored.

In order to study the benefits of such a methodology, we have adapted a more sophisticated version of stage search to the *cb-NET* reachability graph search employing A^*. We have called this algorithm *cb-NET Dynamic Window A* Search (cb-NET DWS*),* since the application of A^* is constrained to a search window that dynamically advances towards markings that are deeper in the reachability tree. The following is a basic description of the algorithm:

The search starts as an A^* approach, but markings at a given depth *(TopDepth)* are created but not explored. When a certain number of markings *MaxNodes* at *TopDepth* have been created, the markings whose depth is less or equal to *ForgetDepth* are discarded and the rest kept. *Fig. 52* shows the *cb-NET DWS** algorithm.

Algorithm *cb-NET DWS

Receives: M_0, M_F: The initial and final marking of a *cb*-NET *N* from *FmsML*.

Returns: Sequence of transitions representing the schedule.

Cb-NET A* is embedded in the algorithm

Variables: *MaxNodes, TopDepth, and ForgetDepth* settings of the algorithm.

UnExplored = M_0

(1) **while** number markings at *TopDepth* ≤ *MaxNodes* **do**

Apply cb-NET A* algorithm *(if a solution is found then exit successfully)*

Retrieve paths to markings whose *depth* > *ForgetDepth*.

Remove markings whose *depth* ≤ *ForgetDepth* from *UnExplored*

Actualise the attribute *depth* in all remaining markings as:

Depth(M) = *depth(M)* – *ForgetDepth* $\forall M \in Explored$

goto (1)

Fig. 52: cb-NET DWS algorithm*

Notice that the application of the *cbNET A** algorithm within this schema is constrained to the exploration of markings whose depth is less than *TopDepth.*

Each of the three parameters considered affects the performance of the algorithm in the following way:

(a)*ForgetDepth*

Choosing *ForgetDepth* equal to *TopDepth-1* we only keep the first *MaxNodes* found at the *TopDepth.* This maximises the algorithm's forgetting capability and represents a drastic decision. On the other hand, Choosing *ForgetDepth =1* (which is the lowest level), we progressively forget only the last level thus damping the irrevocable decisions made.

(b) *TopDepth.*

Determines the depth-size of the tree within which we apply *A**. Experiments showed that affordable values are between the range of 5-20, although this is obviously problem dependent. Notice that a full backtracking *A** algorithm is applied in this search space and will be expensive for large values of *TopDepth.*

(c) *MaxNodes.*

Determines when to enter the *forgetting* procedure. An infinite value of *MaxNodes* implies the complete exploration of the sub-tree delimited by *TopDepth.*

The following experiment aims to determine the evolution of the performance variables of the *DWS** approach. A set of 20 random problems was generated consisting of three machines, five jobs, and between three and seven tasks per job. Each operation can be achieved by a maximum of 3 alternatives, each operation requires a single resource. The lot size was set to 3 parts for each job, meaning an average number of 75 operations to schedule, which is impractical for a pure *A** strategy.

The *cb-NET A** algorithm employed h_{RCR} as the heuristic function and included *CGS* and *Test3* for checking similar markings. *Test3* was based on an estimation of how soon parts and resources will be ready to process in the current marking. As demonstrated in chapter 5, this test may reject paths leading to the optimum solution, but optimality has been already sacrificed with the application of *DWS**.

The problem set was solved for different values of *TopLevel (windows size)* and *MaxNodes*, while *ForgetLevel* is always set as equal to *TopLevel-1.*

Fig. 53a shows the mean relative distance of the solution obtained to the theoretical optimum makespan $h^*(h_{RCR})$ for the 20 problems for each setting of *cb-NET DWS**. *Fig.53b,* shows the number of markings explored for each setting. Finally *fig. 53c* shows the execution time on a *Pentium* PC at 166 Mhz. Each algorithm setting is represented as the pair *(TopLevel, MaxNodes)* in each graph.

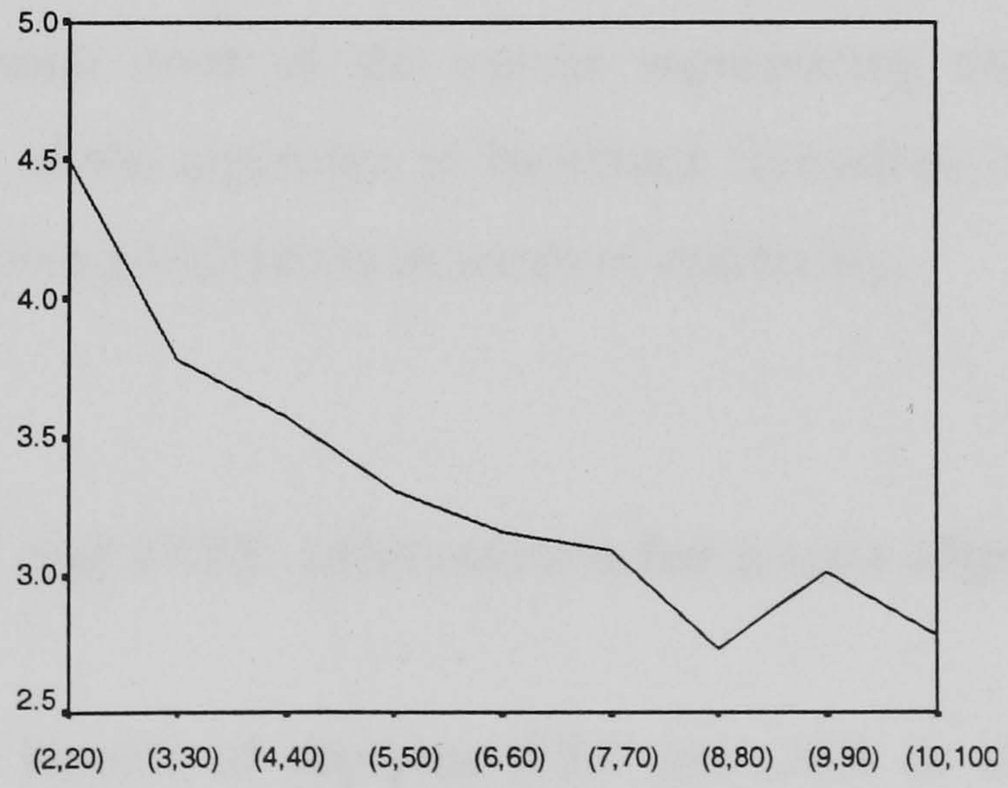


Fig. 53a: Relative difference from theoretical minimum makespan

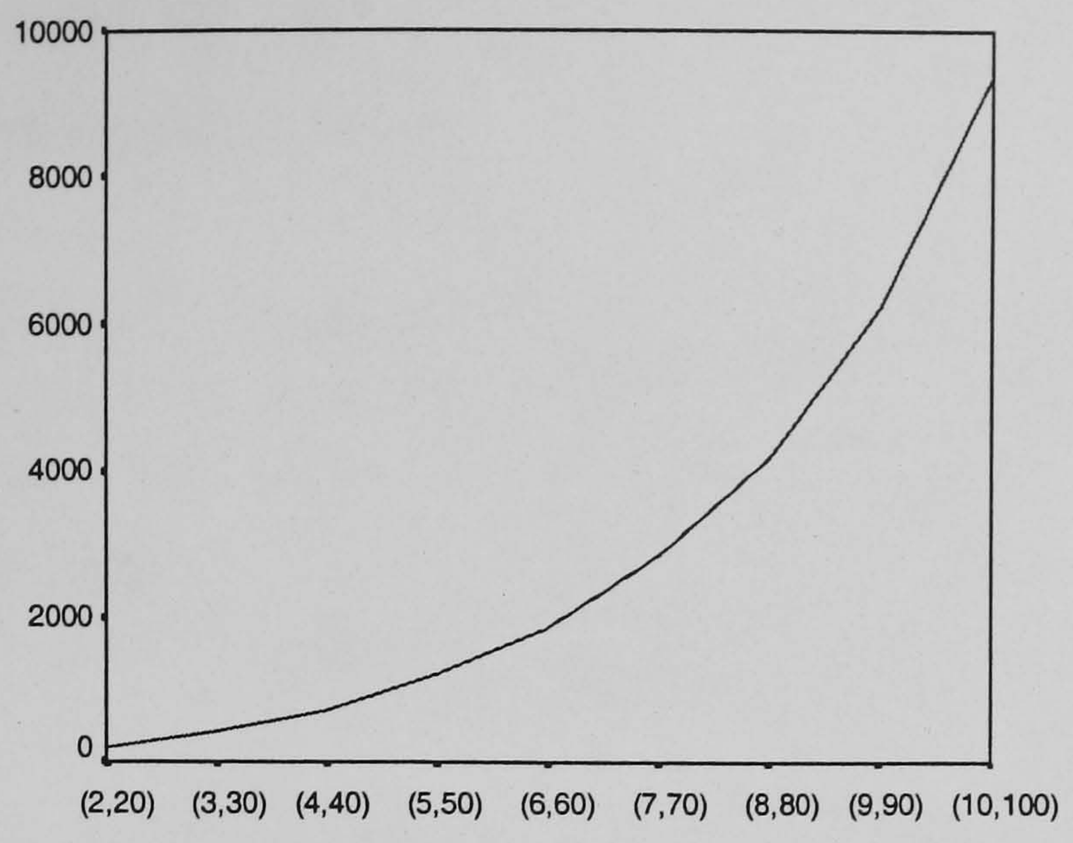


Fig. 53b: Total number of nodes explored

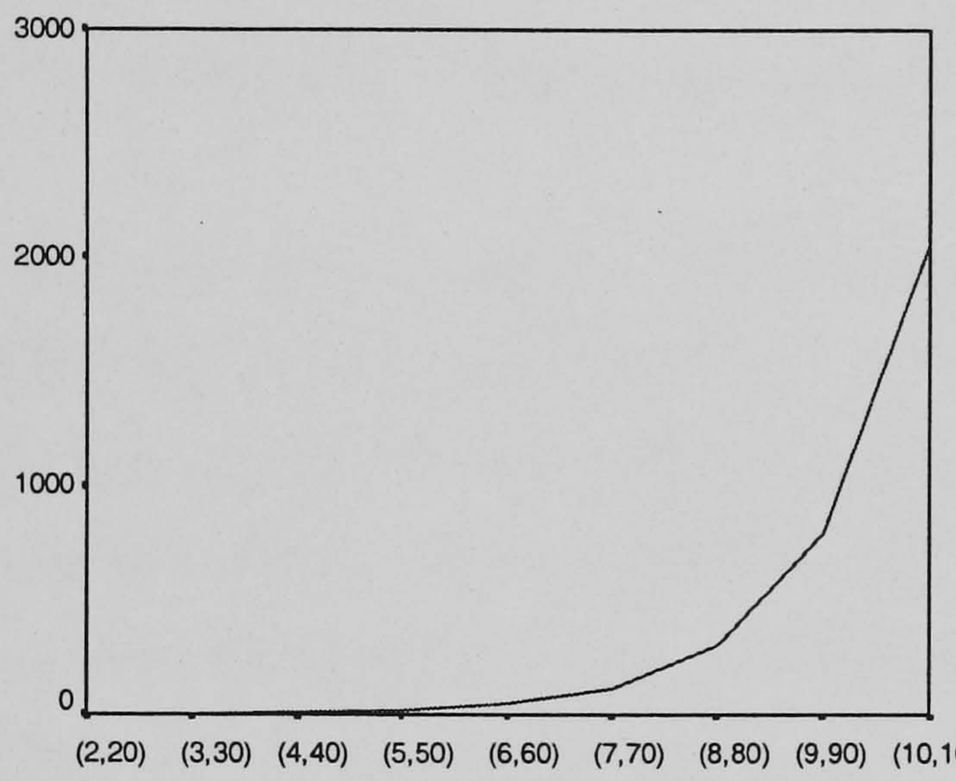


Fig. 53c: Execution time in seconds.

The results indicate that *DWS* the approach is promising. It seems to be possible to *escape* the exponential zone of the curves representing computational cost by reducing the capability of the algorithm to backtrack (introduce irrevocable decisions) while being able to achieve good results in terms of optimality.

3.3 Analysis of *HST* and *DWS*: motivations for a new algorithm.

The immediate benefit of applying *HST* and *DWS* to *A** with a *depth-first* heuristic function [Lee 94] is that reduces the *depth first* component of the heuristic function and improves the poor results obtained due to this fact. For example, in a case study proposed in [Lee 94], the best makespan reported there employing an *A** version

with $h(m)= -w \bullet depth(m)$ was 426. w was set to *2*. *Table 12* summarises our results obtained with different algorithm settings.

Algorithm	*w* setting.	Makespan	Markings Explored
*Cb-NET A**	1.5	396	474
*DWS**	1.25	389	878
DWS + HST*	1	375	937

Table 12: execution summaries.

Our *cb-PN A** algorithm incorporating *IGC*, allowed us to reduce the setting of w to 1.5, but failed to finish the search, (e.g. for $w=1.25$) If we employ *DWS* with a window size of 5, we can apply this setting and finish the search successfully. Finally, incorporating *HST,* further relaxation is possible due to the heuristic identification of promising transitions.

Unfortunately, *h(M)* still needs to add a depth first search component to the search and problems arise when an admissible heuristic function is used to guide the search[26]. The reason is that neither *HST* nor *DWS** escape from the combinatoric explosion. This is because a *best-first* search is applied within the search window and although *HST* can reduce the branching factor, even with a binary search tree the search space grows exponentially.

In addition, the way irrevocable decisions are performed in *DWS** plays a major role in the quality of the solutions obtained. If we set *ForgetDepth* equal to *TopLevel-1* (we refer to this setting as pure stage search) we guarantee that a constant number of nodes are kept at each *A** stage. The main problem with this is that the irrevocable decision is much more drastic, increasing the chances of a local optima effect. To smooth this effect, the parameter *ForgetDepth* in *DWS** may be set to values which are relatively distant from *TopDepth*. *DWS** will then still go deep into the reachability tree whilst it progressively cancels the possibility of backtracking to markings beyond a maximum depth. The capability of *DWS** to recover from bad moves is then superior to a pure stage search. The problem is that now the number of markings that are kept every time the *search window* advances is not constant and will typically grow exponentially.

---

[26] Although in the experiment of the last section, we employed h_{RCR} as an admissible heuristic function, the problem set was created with the intention to increase the chances of h_{RCR} to guide the search effectively.

We confirmed this behaviour experimentally. The first experiment studies how the search effort is affected by different settings of the parameter *ForgetLevel,* when the size of the search window is kept constant in *DWS**. A set of 20 random problems consisting of 3 machines, 10 jobs and 4 tasks per job was generated, making a total of 40 operations to schedule.

The window size was set to 5, *MaxNodes* was set to 1 and each problem was solved four times for values of *ForgetDepth* from 1 to 4. *Fig* 54*b* shows the average number of markings explored versus the value of *ForgetDepth* while *Fig. 54a* shows the average relative distance of the solution obtained from the lower bound expressed by $h_{RCR}^*(M_0)$ for each problem.

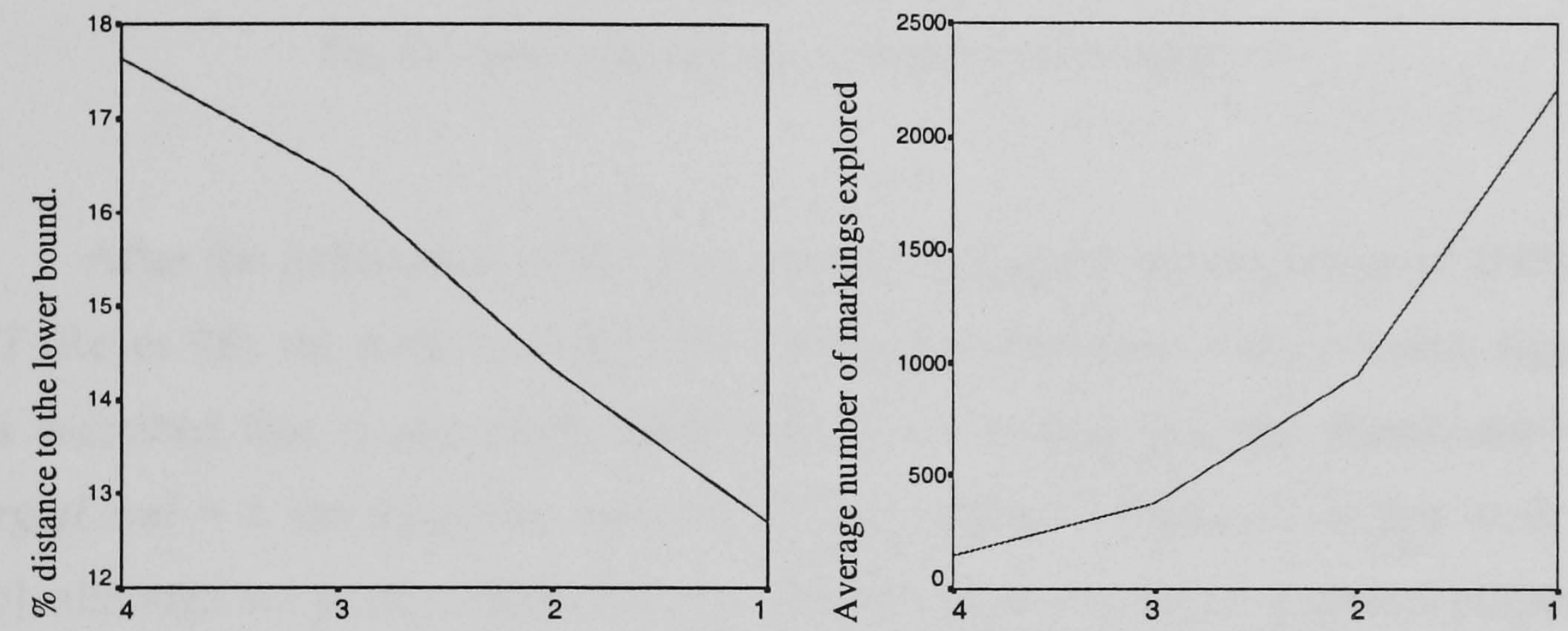


Fig. 54: (a) Average relative distance to the lower bound (b) Average number of markings explored ForgetLevel *versus values of* ForgetLevel.

It is obvious that since settings of *ForgetDepth* closer to *TopDepth* increase the search effort better solutions can be expected. However, it is our intuition that this improvement is also due to the greatest backtracking capability.

The second experiment is more meaningful. A simple problem consists of two machines a robot and two part types (jobs). Each part needs two operations to be completed. However, the robot is only employed in one of the operations which creates imbalance in the use of resources or, in other words, h_{RCR} becomes too optimistic.

Even with a low setting of *DWS* (TopDepth=3, ForgetLevel=1, MaxNodes =1)* the search effort grows exponentially with the size of the problem. *Fig. 55* shows the number of markings explored when the number of parts to be produced of each type is increased from 1 to 7. These results clearly indicate that even for small problems it is

not possible to increase backtracking to the limit in *DWS** although it is an interesting thing to do to avoid excessive irrevocable wrong decisions.

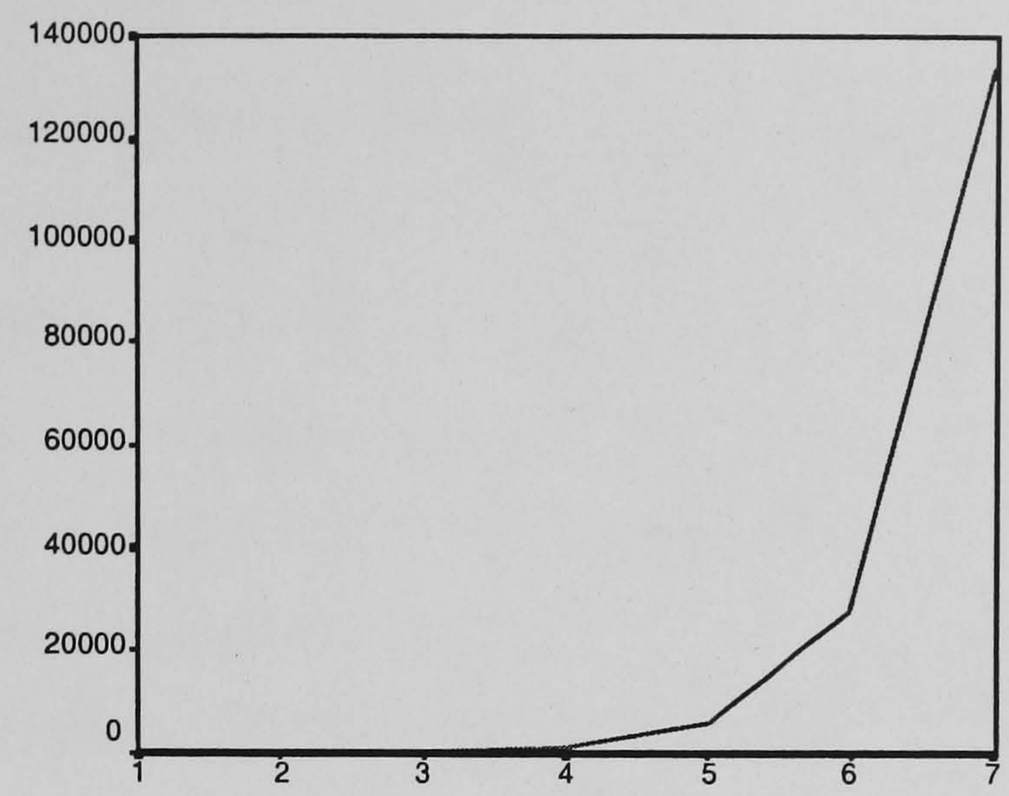


Fig. 55: Number of markings explored versus problem size.

After the publication of the first results employing an integration of *DWS** and *HST* [Reyes 98], the work in [Jeng 98b] came to our attention. A stage search algorithm was described that is essentially *DWS* with the following settings, *MaxNodes=1* and *ForgetLevel = 1,* the algorithm only allowed the tuning of *TopLevel.* In this work [Jeng 98b], although not purely admissible, the heuristic function can for practical purposes be considered to be admissible. This is why they were forced to consider a very weak setting for their version of stage search (*TopLevel=4 if we translate* to *DWS*)* when solving medium/large size problem instances (100-200 operations to schedule).

Our own experience leads to the same tuning needs. When the case study proposed in [Lee 94] was solved with *DWS* (TopDepth=5, ForgetLevel=1, MaxNodes =5)* and h_{RCR} as the heuristic function, a solution was not obtained in a reasonable amount of time, despite the fact that h_{RCR} is a good estimate for this problem and is typically satisfactory for the first 70% of the search[27].

The following section will present an algorithm that preserves the benefits of *HST* and *DWS* but which deals effectively with the complexity problem thus allowing the application of well-informed *PN* based heuristic functions.

---

[27] The best solution that we have obtained for this problem is *329* while $h^*(M_0) = 310$.

4. Dynamic look-ahead stage search: *DLSS*.*

4.1 The *breadth-search* effect.

When an admissible heuristic function is employed in a *Best First* algorithm, the evaluation function $f(M) = g(M) + h(M)$ represents an optimistic estimation of the actual makespan ($h(M) \leq h^*(M)$). As the search advances, previously ignored markings now become candidates in terms of $f(M)$. This forces the algorithm to backtrack to previous markings. It is possible that by reconsidering alternate previous paths, the search might continue successfully. This is obviously true, but, in practice, what happens is that the distance of $h(M)$ to the actual makespan decreases the closer we are to a goal marking.

In other words, a previously rejected marking is now explored but, as soon as its successors are generated, the optimistic estimation is corrected by the actual schedule and a new backtracking procedure towards better heuristically valued markings is performed.

To show this idea, let us consider the problem given in the introduction to this chapter. The *FMS* with five jobs, three machines, and a total of 20 operations. A software monitor was included in the algorithm that records the depth of the marking that is currently being explored.

The following graph in *fig. 56* shows the evolution of the search in terms of depth of the marking explored as the search advances in terms of number iterations.

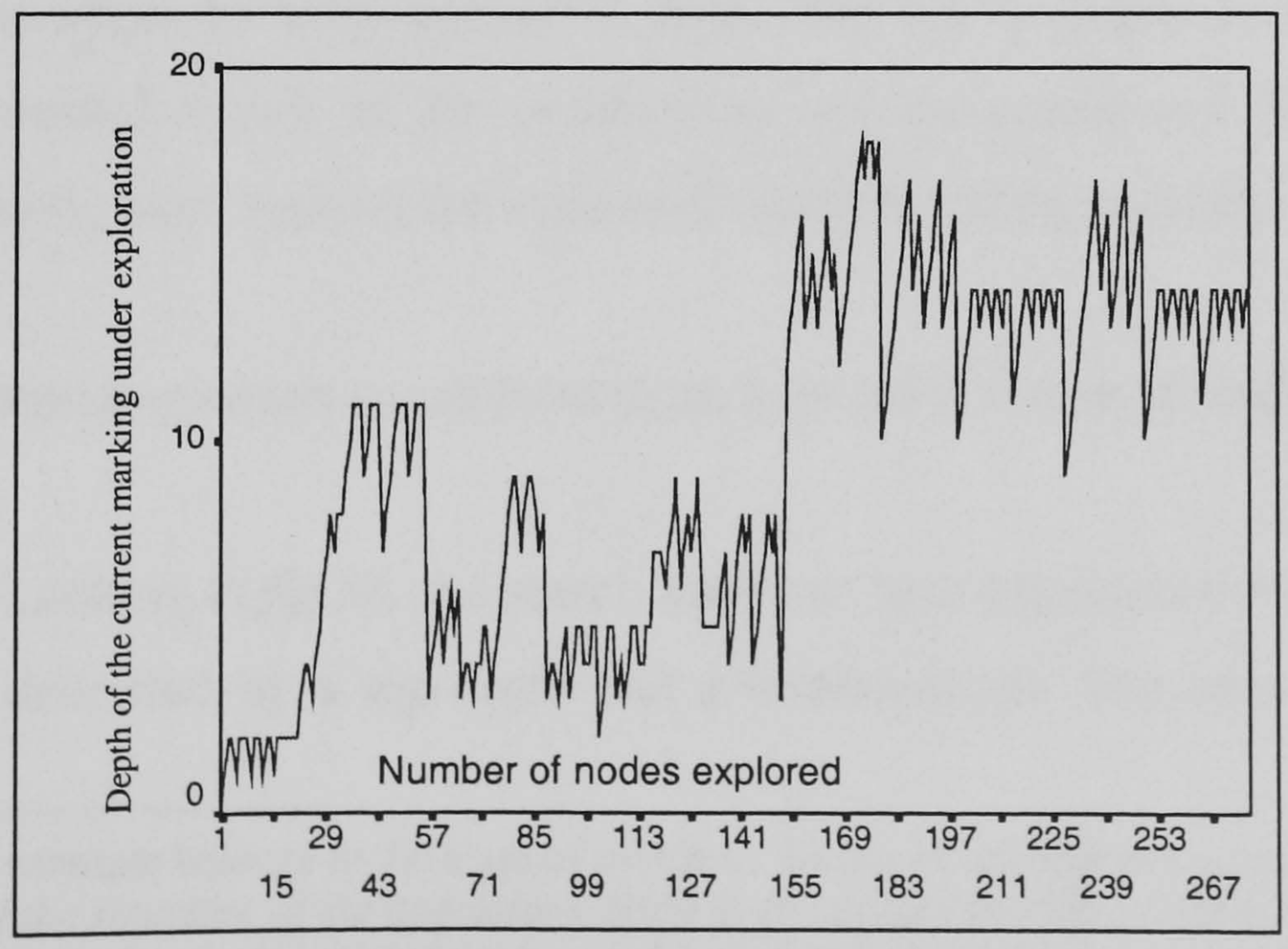


Fig. 56: Evolution of the search in terms of depth of the marking.

It is clearly seen in *fig. 56* that the search is characterised by a phase where the heuristic function guides the search towards deeper markings (which are closer to the solution) and a phase of backtracking over heuristically better markings.

By considering three products of each job type, which makes a total of 60 operations to be scheduled, a second experiment was performed. Obviously, the problem was unaffordable for the *cb-NET A** algorithm, so the execution was halted with no solution being found after 80.000 markings where explored. *Fig. 57a* represents the evolution of the search in terms of the depth of the marking for the first 4% of the search. *Fig. 57b* shows the complete search space after 78.000 nodes.

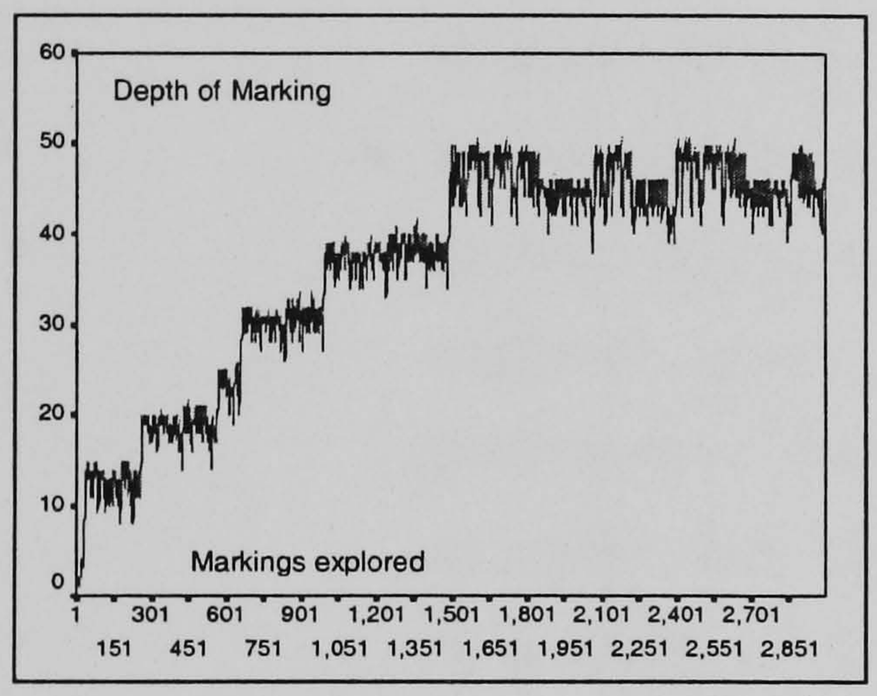


Fig. 57a: first 4% of the search

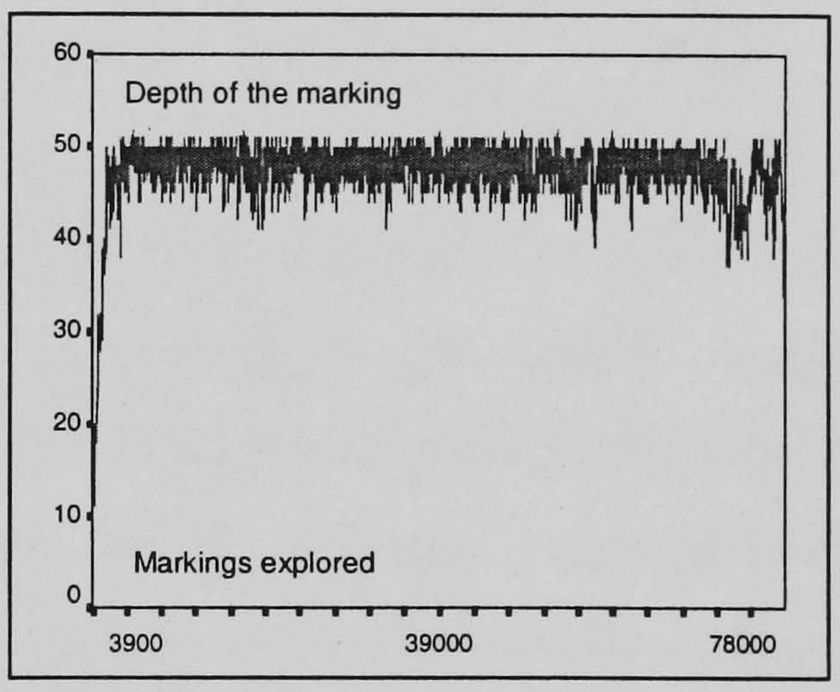


Fig. 57b: the search after 80000 markings explored

The *PN* based heuristic function guides the search relatively well during the first 75% of the operations to be performed[28]. After this, the heuristic search cannot deal with the exponential nature of the reachability tree generated and the breadth-first behaviour is clearly seen, making the algorithm collapse before reaching a solution.

Two interesting observations from these three plots can be identified:

- Looking at *fig 56,* the search seems to be constrained within a bandwidth delimited by a top-depth and a bottom-depth. The heuristically guided

[28] Notice that this example belongs to the class of problems discussed in chapter 4, section 5: *Jobs are similar in terms of the structure of the operations, there is an acceptable degree of flexibility, and there are no critical bottlenecks, the heuristic function performs satisfactorily.* In a situation where the heuristic function is too optimistic, the *breadth-search* stage is reached earlier.

depth-first search stages are clearly identified, followed by a recovery stage.

- This effect becomes more evident in *fig 57a* and *fig 57b,* where drastic recovery backtracking stages are missing but the search can be seen as a decomposition of two effects: A band-width limited best-first /backtracking strategy and a breadth-first search strategy that dominates.

This appears to be common to all our case study problems. The conclusion is that the search seems to be constrained in a top-bottom bandwidth that follows a breadth-first. A dual hypothesis about the *FMS* scheduling problem can be formulated as follows:

- The flexibility of the system results in multiple alternate schedules which are heuristically valued similarly in terms of makespan.
- The consequences of choosing the wrong operation have little impact on the overall cost, and the effect is limited (and can be corrected in a relatively small local neighbourhood). In other words, the performance of the *FMS* does not depend on a single decision made at an earlier stage, but on a complex and difficult to identify strategy. It is necessary to maintain a reasonable local backtracking capability in order to test the effect of different scheduling decisions.

These hypothetical characteristics of the problem are compatible with the lack of clearly advantageous paths identified by the heuristic function and the breadth-first search degradation of the optimistic heuristic as the search advances.

4.2 DLSS description.*

The aim of *DLSS** is to implement the band-width limited *depth-first /best-first* backtracking strategy observed in *fig 56* and *fig 57a/b,* but avoiding the exponential generation of heuristically equal markings as the search progresses. The objective is to overcome the *breadth-first* effect. In other words *DLSS** monitors whether the expected results are reasonable during the search. If this is the case, search is allowed to continue in a *depth-first* manner, if not, the *best-first* effort will be increased to a limit.

The *cb-NET A** search is constrained by a number of markings that are contained in a geometric structure called the *Search Frame (SF)* which is conceptually identified with the *UnExplored* list in the *cb-NET A** algorithm. The number of markings that this frame can contain is limited; thus avoiding exponential grown. *SF* changes dynamically both in a) contents and b) search limits:

a) If *SF* has reached its limited capacity, the inclusion of a newly generated marking is controlled by determining how promising the new marking is with respect to the ones already included in the frame.

b) The *cb-NET A** is also geometrically limited by *SF* in two aspects: the backtracking is limited to the markings contained in *SF;* and no markings beyond the limits of the search frame can be generated. Since the search must progress to a final marking representing a solution, this frame must not remain stationary. In this sense, as *SF* advances it allows *A** further exploration at the expense of introducing irrevocable decisions. This dynamic behaviour of *SF* is controlled by a set of rules.

At each iteration of the algorithm, a marking *M of* the *SF* yielding the lower value of *f(M)* is selected for further exploration. New markings are obtained and the inclusion criterion for these markings is examined. The following sections explain the algorithm in detail.

4.2 Geometry of the search frame.

Each marking *M* in the reachability tree is associated with the number of transitions that have been fired. This value is equivalent to the depth of *M* in the search tree: *depth(M)*. In our modelling paradigm, since operations are represented by transitions, *depth(M)* also matches the number of operations that have been scheduled. The geometry of *SF* is defined by two edges: (*vertical* and *horizontal*). We define the *vertical* edges of *SF* in terms of two integers: *bottom-depth* and *top-depth* which defines the minimum and maximum depth of the markings that *SF* can contain. Hence, a marking *M* is only included in *SF* if $depth(M) \in [top - bottom]$

These two values limit the application of the *A** algorithm in two ways:

a) The backtracking capability is constrained to markings whose depth is equal to or bigger than *bottom-depth.*

b) Nodes of depth equal to *top-depth* can be generated but not explored.

The *horizontal* edges of *SF* make reference to the maximum number of markings of *[bottom-depth ... top-depth]*, that the frame can contain. We have organised *SF* into levels, each level is labelled with a number that represents a depth in the *PN* reachability graph, and therefore *SF* contains a total of *top - bottom* levels. A level *l,* can only contain a maximum of *max_nodes(l)* markings of *depth(M) = l.* The number of markings currently allocated in this level is expressed as *size(l).*

The *vertical* edges of *SF* limit the backtracking capability of the algorithm; the *horizontal edges* limit the scope of selection, and the *breadth* of the reachability tree. As a result, *SF* is shaped as a vertically symmetrical geometric figure.

In a first implementation of this methodology, we have consider *max_nodes(l)* to be equal to a fixed constant *max_wide* for every level. This defines a rectangular shape that can allocate a maximum number of markings given by the expression *[top - bottom] • max_wide.*

An immediate advantage of the organisation of the candidates markings to be explored in levels, is that it facilitates the search for similar previously explored states. This is justified because the number of operations to complete any job is fixed. [Chen 95] however, describes a second type of *FMS* (asymmetric), where a job can have alternate routes or *plans* with a different number of operations. These paths can be transformed into symmetric ones with the introduction of intermediate transitions.

4.4 Introducing irrevocable decisions: *Dynamics of SF.*

From the previous section, we can conclude that the *UnExplored* list of *A** has been substituted by a structure called *SF* which constrains the application of a pure *A** search both in terms of backtracking and depth.

In this sense, *SF* provides a bounded safety frame within which to apply a heuristic best first search that considers the markings at *bottom* level as irrevocable

decisions. It still remains the case, of course, that the search must progress towards a final marking representing a solution to the problem. The *horizontal* edges of *SF* are, in this initial approach, kept unchanged during the search process. But the *vertical* edges of *SF* must change in order to allow *cb-NET A** to: a) assume new irrevocable decisions and b) allow a deeper generation of the reachability tree. As we will see, the *advance* of *SF* will be determined by two rules. A first rule forces the search to progress under the existence of heuristically equal paths. A second rule allows the search to progress *depth-first* if the heuristic function is able to identify promising paths.

The *advance* of *SF* follows the rule shown in *fig. 58*:

Rule 1: *if* the bottom *level of* SF *is* empty *then increase* bottom *and* top *edges*[29]. The explanation for this is that since the *bottom* level represents the limit of the markings where the algorithm can still backtrack; if *bottom* level is emptied during the search[30], level *bottom+1* becomes the new backtracking limit. Notice that no new markings for *bottom* level can be generated, as this would mean that a marking from level *bottom -1* has been chosen for exploration, which contradicts the specification of the *Search Frame.*

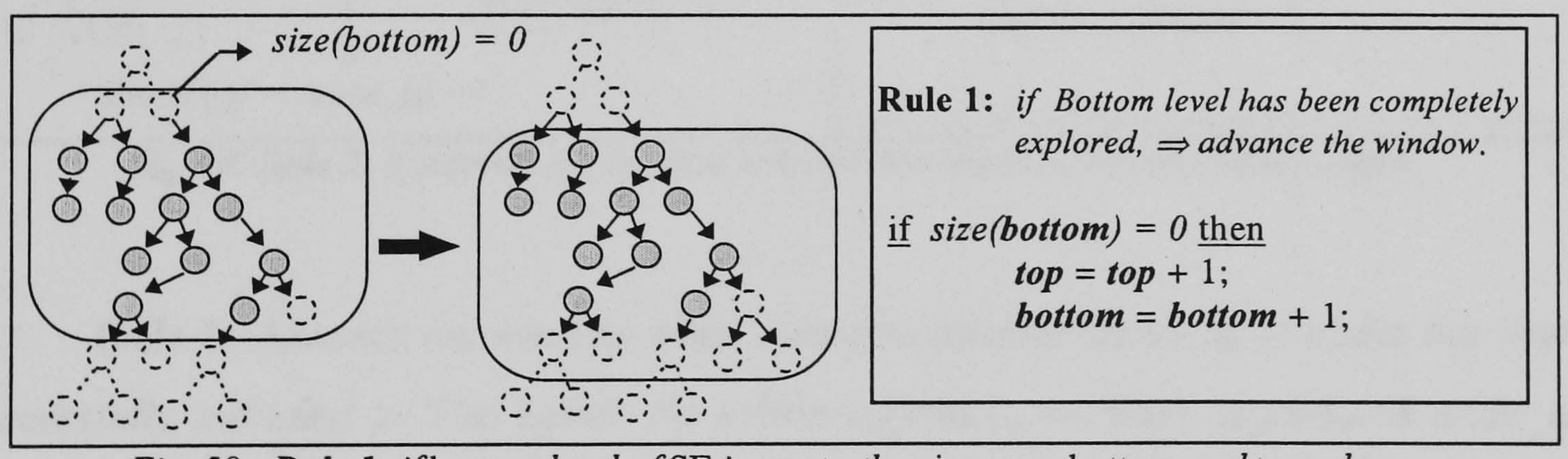


Fig. 58: ***Rule 1:*** *if* bottom *level of* SF *is* empty *then* increase bottom *and* top *edges.*

This rule is enough to ensure the advance of *SF:* the number of markings kept for exploration is limited and no new markings beyond *top* level are generated. However, controlling the behaviour of *SF* with this single rule, inhibits the capacity of a well-informed heuristic function to guide the search quickly towards a solution. In other words, it forces all markings contained in *bottom-level* to be explored. A rule decision

---

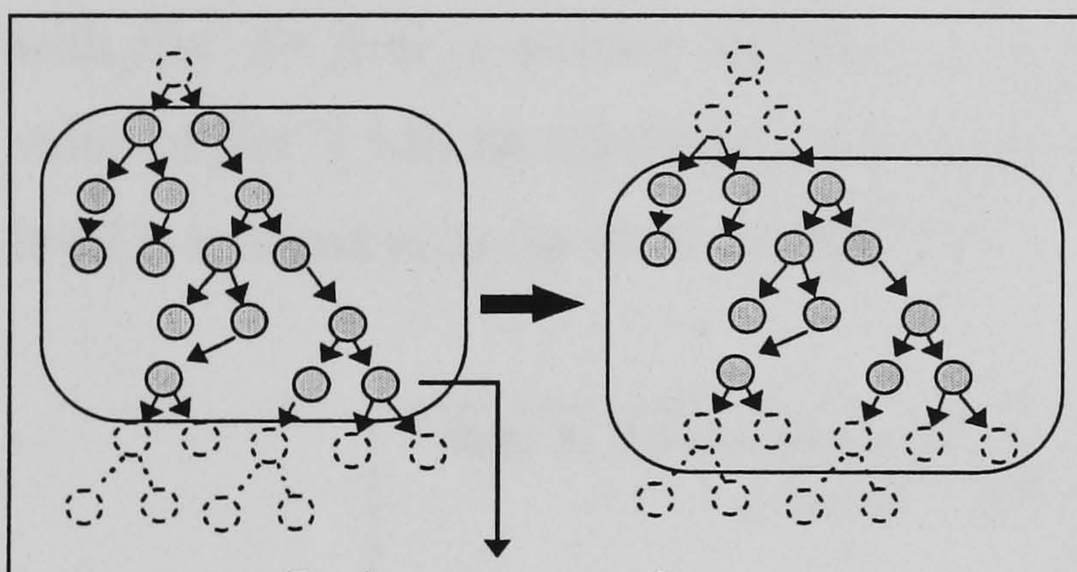
[29] From now on, an increase in the *top* and *bottom* edges means *to advance SF.*

[30] Remenber that when a marking is selected for exploration, it is removed from the *UnExplored* list, now expressed by *SF* .

which is the extreme of *Rule 1* could be as follows: when the first marking of level *Top* is found, advance *SF*.

In a pure *A** approach, whenever *f(M)* is optimistic, the search tends towards a final-goal marking. If during this *heuristic* depth-first process, the path becomes less promising than was thought, we backtrack. The *DLSS** approach has a limited backtracking capability, so it is important to consider how promising the paths are beyond the recoverability edge. In other words, we must define a criteria to determine when *DLSS** can take the risk of introducing irrevocable decisions.

As a compromise between the two extremes represented by *Rule1: explore all the markings at bottom level before advance SF* and by the rule *Advance SF as soon as you can.* We experimented with an alternative shown in *fig. 59: Rule 2,* which was intended to explore when one could take the risk of introducing irrevocable decisions (advance *SF*).



Rule 2: *if relevant information is found,* ⇒
consider nodes at bottom+1 as irreversible decisions and advance the window.

if *size(top)* ≥ ***move-at*** then
Reject *markings* in *bottom-Level.*
top = *top* + 1;
bottom = *bottom* + 1;

Fig. 59: ***Rule 2:*** *if* relevant information is found *then* increase bottom *and* top *edges.*

Rule 2: *Advance the window when a certain number* move_at *of nodes has been successfully included in* Top *Level.* As a first approach, we have considered *move_at* equal to *max_wide.*

If *h(M)* guides the search reasonably well, markings at *top* level will easily be found, if not, a greater search effort will occur within *SF.* Notice that *Rule 1* guarantees that *SF* advances in the case that *Rule 2* is never satisfied due to continuous backtracking.

4.5 Inclusion criterion.

The last two sections have defined the scope of evaluation of the *DLSS** algorithm and how the search advances towards a solution.

The second dynamic aspect of the *Search Frame* is the heuristic selection of markings for exploration for *SF*. Notice that, as said before, *SF* is limiting the number of paths. This obviously introduces the need for a strategy to optimise the quality of the candidate markings to be explored. This strategy is implemented in *rule 3* shown in *fig 60.*

The immediate idea is to use the estimate function *f(m)* as the inclusion criterion. It is the simplest and cheapest thing to do and it maintains the coherence of the heuristic approach. For each level *l* of *SF*, markings are ordered in the increasing magnitude of *f(m)*. Any newly obtained marking *m* can be included into a level if the level is not yet complete. If the level is complete, the marking will be included only if a marking *m'* with *f(m')> f(m)* is already included. In this case, the marking *m''* with the greatest value of *f(m'')* will be rejected for exploration and discarded. The marking *m''* for each level is referred to in *fig. 60* as *WORST(l)*.

Rule 3: *if a new marking* m *is created for level l.....*
if *size(l) < max-nodes(l)* then
include *marking* m *in SF*
else if *f(m) < WORST (l)*
include *marking* m *in SF*
reject *WORST (l)*
else **reject** *marking.*

Fig. 60: Rule 3.

4.6 Final algorithm, integration of *A**, *HST*, *CGS* and *SF*: *DLSS**.

Fig 61 shows the pseudo-code for the algorithm. *SF* is initialised to the initial marking. The generation of the *PN* reachability tree follows *cb-NET A**. The next marking is selected from *SF* using the heuristic function *f(m)*. The *PN branching scheme CGS* is integrated within the algorithm to determine the set of transitions to apply. *HST* truncates the set based on the *PN* dispatching rule applied.

Every new marking obtained is compared with the final marking and with previously generated markings in the normal way using any of the tests for similar markings described in chapter 5. If the new marking is a candidate for exploration, *Rule 3* is applied to see if inclusion in *SF* is possible. Finally, *Rule* 2 is checked to see if *SF* should advance.

Algorithm *DLSS:**

Receives: M_0, M_F: The initial and final marking of a *cb*-NET *N* from *FmsML*.

Returns: Sequence of transitions representing the schedule.
Cb-NET A* is embedded in the algorithm

Variables: *move_at, max_wide, bottom and top level* settings of *DLSS**
SF: Search frame containing *new* markings for exploration.
Explored: List of markings already explored.

SF = { M_0 }
Explored = ∅
(1) while *SF* is not empty **do**
Apply **Rule 1** *; If the bottom level is exhausted then advance SF*
Remove a marking *M* from *SF* with the smallest *f(M) and depth(M) < top.*
Put *M* in the Explored List
(2) for every new marking *M'* that can be generated using *CGS* and HST do
if *M´ is ithe goal marking* **exit** with success.
else-if *M'* pass the test for *similar markings* of *cb-NET A** **then**
Call **Rule 3** for marking *M'*
Apply **Rule 2** *; If move_at markings have been found then advance SF*
goto (2)
goto (1)

Fig. 61: DLSS Algorithm.*

The integration of all the methodologies developed results in a *cb-NET* based heuristic search algorithm that overcomes all the difficulties encountered with previous approaches analysed in section 2 and 3. Each methodology concentrates on different aspects of the search space and aims to reduce the search effort whilst maximising admissibility.

The following summarises the function of each of these components.

- ***cb-NET A* & PN* Admissible Heuristic function:** guides the search towards the most promising paths.
- ***PN* based branching scheme *CGS*:** Avoids the generation of paths that do not yield better solutions.
- **Check for previously markings**: Prune markings that have been reached before by other better paths.
- ***HST*:** Heuristically determine those operations whose application is not likely to lead to a near optimum solution.
- ***DLSS*:*** Control the search effort by heuristically avoiding exponential generation of markings and force the search to progress to a solution. Heuristic rules control the rejection of heuristically less favoured paths that have passed *CGS, HST* and have not been explored before.

It is worth noting that *DLSS** represents the most dramatic decision in terms of rejection of paths by the use of rule *Rule 3*. It is important that markings representing different paths to achieve the same, schedule permutations and paths which are not optimum are determined before are not included in *SF*. The integration of *CGS* and the test to identify previously reached markings successfully achieves this objective.

4.7 Deadlock avoidance and *DLSS*.*

An exploration algorithm that has limited its backtracking capability does not guarantee finding a solution. For the problem domain of *FMS*, deadlocks might occur, due to buffer policies and time constraints imposed on sub-parts[31].

In [Abdallha 98b] deadlock control has been addressed by three approaches: Deadlock Recovery, Deadlock avoidance, and Deadlock prevention.

The first technique allows the occurrence of a deadlock and then implements a recovery procedure. The second method tries to minimise the occurrence of deadlocks by a control policy. The third implies the use of a deadlock free algorithm.

---

[31] Which are modeled as constrained places in a *cb-NET* markings.

*DLSS** algorithm can be considered as a deadlock avoidance policy, since deadlock situations (dead markings) can be detected within the *Search Frame* and several alternatives are available for exploration[32]. But complete backtracking is not possible with the actual approach, and hence the algorithm is not deadlock-free, as for example *A** or *B&B*.

4.8 Effects of the heuristic function in irrevocable decisions.

We justified the limited expansion/limited backtracking nature of *DLSS** by the nature of the *FMS* scheduling problem, based on the hypothesis that the consequences of choosing the wrong operation is limited and can be corrected in a reasonably small local neighbourhood. In other words, we assumed that the performance of the *FMS* does not depend on a single decision made at an earlier stage, but on an overall strategy. With this assumption we have presented a scheduling algorithm, *DLSS** that performs local search based on a strategy that in the main follows a heuristic *best-first* search.

The strategy of *DLSS** lies in the estimation given by *f(m)* as the combination of what has been achieved (*g(m)*) and what is left to be achieved (*h(m)*).

Since the backtracking capability of *DLSS** is limited, it is important that *h(m)* depends on the state of the system and not just on limited information (as for example the number of operations remaining). If not, a local optima effect is likely to be produced.

We performed a detailed study of how the cost of the candidate marking under exploration evolved during the search process. We observed that, when scheduling the last 30% operations, the quality of the solution decreased noticeable more as it did for the first 70% of operations. This fact was initially taken as obvious and explained as a result of a smaller degree of freedom (choices) when few remaining operations were left to schedule. Although this is partially true, a more detailed observation revealed the following effect: *DLSS** followed a *SPT* strategy scheduling the fastest operations first, leaving the longest for the end. Obviously this will not affect a complete *A** as it will correct the situation, but not so *DLSS** since it has backtracking limitations and hence this *SPT* behaviour may influence the overall strategy previously mentioned.

---

[32] Recall that in chapter 3, section 6, several systems with buffer constraints were solved satisfactorily by *DLSS**.

Fortunately, the heuristic function h_{RCR} proposed in chapter 4 offsets this situation by taking into account the operations that have not yet been achieved. Consequently paths with the fastest transitions will produce a greater value of *h(M)*, as a result of the longer operations still not being scheduled. However, despite the capability of h_{RCR} to identify situations of this kind, the *SPT* effect still happened as experimental results showed.

A simple modification of the heuristic function, by incorporating in *DLSS** heuristic strategies, such as production rates and balance of different product types, showed an improvement of the results and a minimisation of this effect.

An alternative heuristic function was proposed as: $h_{RCR}'(m) = h_{RCR}(M) - Balance(M)$. *Balance* (see *fig. 62)* considers the places of the *PN* that represent operations performed. A marking where the parts are spread over all the system, and not concentrated in certain places gets greater values for *Balance*. On the other hand tokens concentrated at certain nodes (as may happen if only the fastest operations are considered first), get a lower value of *Balance*. Obviously, this distribution of *WIP* parts is heavily influenced the buffer constraints.

Function Balance
Receives *M* marking of a *cb-NET*.
returns *d* a heuristic value.

$d = 0$
$\forall\ p \in Q$
    if $M[p] > 0$ **then** $d = d + 1$

Fig. 62: Algorithm for Balance.

With the aim of testing the superiority of our heuristic function avoiding local optima we performed the following experiment. Also we were interested to study the effect of a heuristic function that adds a depth-*first* component to *DLSS**.

A set of 250 *FMS* descriptions consisted of a random number of jobs, machines and tasks. 70% of operations have alternate routing, multiple resources may be needed to perform a task and variance of operation time among alternate choices is set to 33.3% of the mean cost. Number of parts to be produced per job is between 1 and 10.

*DLSS** was employed to solve the problem with the following settings: the vertical and horizontal size of *SF* were both set to 5, as was the number of markings, to

find at top level before advancing *SF*. *ICG* & *Test1* for similar marking is included, *HST* is set to 10, employing $k'(m,t)$.

Each problem was solved three times with exactly the same settings but using different heuristic functions:

a) The heuristic function proposed in [Yim 96] and [Lee 94] (see chapter 4, section 2): $h_{Yim}(m) = -w' \bullet \bar{O}p \bullet depth(m)$, where w' was set to the inverse of the total number of resources, to model maximum machine parallelism.

b) $h_{RCR}(m)$

c) $h_{Bal}(m) = h_{RCR}(m) - Balance(m)$.

Table 13 summarises the comparison of the makespan obtained among different heuristics. The value presented for two heuristics *A* and *B* is the % of relative difference of the makespan of the solutions obtained and it is calculated by:

$$Rd(A,B) = \frac{Makespan(A) - Makespan(B)}{Makespan(A)} \%$$

	Minimum	Maximum	Mean	Std. Deviation
$Rd(h_{Yim}, h_{RCR})$	-13.5	28.9	7.5	7.7
$Rd(h_{RCR}, h_{Bal})$	-14.9	19.3	1.1	4.8

Table 13: Comparison of the makespan obtained

Table 14 summarises the average of nodes explored using each heuristic.

	Minimum	Maximum	Mean	Std. Deviation
Iterations using h_{Yim}	203.0	1523.0	680.4	267.0
Iterations using h_{RCR}	219.0	1448.0	678.0	250.8
Iterations using h_{Bal}	214.0	1421.0	666.31	242.6

Table 14: Descriptive statistics of the search effort.

The effect of including *Balance* obtains results that are an average of 1.1% better than pure h_{RCR}.. On the other hand, the makespan difference with h_{Yim} from h_{RCR} is noticeable, 7.5% greater on average, despite the fact that in general terms, h_{Yim} slightly increases the search effort.

The study reveals the importance of developing a heuristic function that takes into account *what has been done* in order to estimate what *has to be done* (h_{RCR} achieves better results than h_{Yim}).

We finish this section by noting that, theoretically, if a more drastic backtracking limitation policy is applied, the greater the tendency towards local optima. For example, the effect of not using *Balance* with h_{RCR} in a pure stage search *(DWS*; TopDepth = 5, ForgetDepth=TopDepth-1)* gives a relative difference of 2.5% with the makespan obtained if *Balance* is considered. This observation may be general if a severe limitation of backtracking is considered [Sun 94] [Yim 96] [Inaba 98] such as in *Beam Search* [Shi 91], as we initially suggested in section 2.

5 Experimental Results.

This section presents experimental tests conducted to investigate the different aspects of the *DLSS** algorithm presented in this chapter. The results obtained complement preliminary results reported in [Reyes 00] and [Reyes 00c].

The following notation is used in the rest of the section:

- *DLSS*(height, width, N)* represents the parameter setting for *DLSS** where *height* indicates the distance between *bottom-level* and *top-level*; *width* stands for the maximum number of nodes per level and *N* is the maximum number of transitions to be kept and fired by *HST* at each marking.
- $h_{RCR}^*(M_0)$ represents the theoretical lower bound for the problem, i.e. the optimum makespan of the relaxed problem where no conflicts for resources are considered. This lower bound is considered as the optimum solution for comparison purposes.
- To compare the solution provided with other solutions or values, we use the *relative difference (Rd)* expression. For example, the relative difference of two makespans is expressed as *Rd (A,B)* and is calculated by the following expression:

$$Rd(A,B) = \frac{Makespan(A) - Makespan(B)}{Makespan(B)} \quad \%$$

All the algorithms were implemented in C++ and run under Windows95 on a 300 Mhz PentiumII PC with 64 Mb RAM.

5.1 Empirical study of computational costs of *DLSS**.

Since the number of nodes in the search window is limited, we can expect the computational cost of the algorithm to be polynomial on the number of operations to schedule.

10 *FMS* descriptions consisting of 3 machines, 5 jobs and four tasks per job were generated. The degree of flexibility (alternate routing) and the operation cost were randomly generated for each problem. 30 problem sets were obtained by increasing the number of parts per job from 1 to 20, resulting in problems with between 20 to 600 operations.

Each of the 30 problem sets contained 10 problems which were solved by *DLSS*(10,5,15)*. Three parameters where obtained for each set. a) The average relative difference *(Rd)* of the makespan for each problem with respect to the lower bound $h_{RCR}^*(M_0)$ b) the average number of iterations of the algorithm (number of markings explored) and c) the average execution time.

Figs. 63a, 63b, 63c represent the evolution of these values as the problem size increases, given the number of operations to schedule for each of the 30 problem sets.

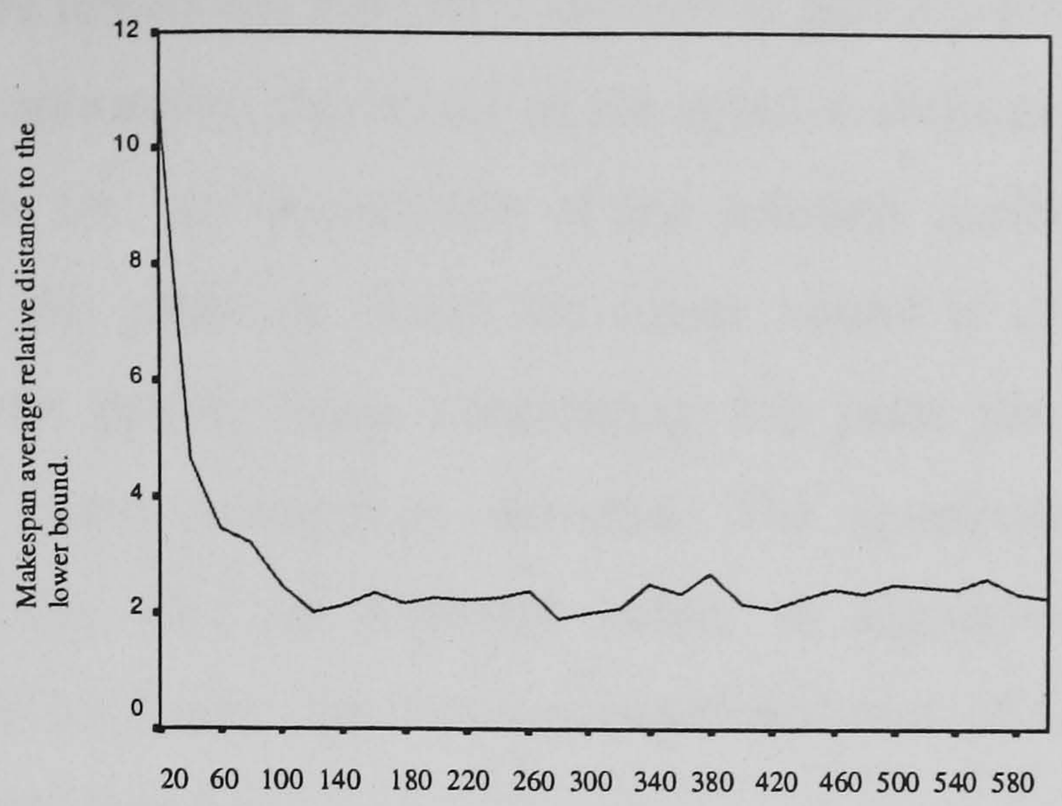


Fig. 63a: Evolution of the solution obtained.

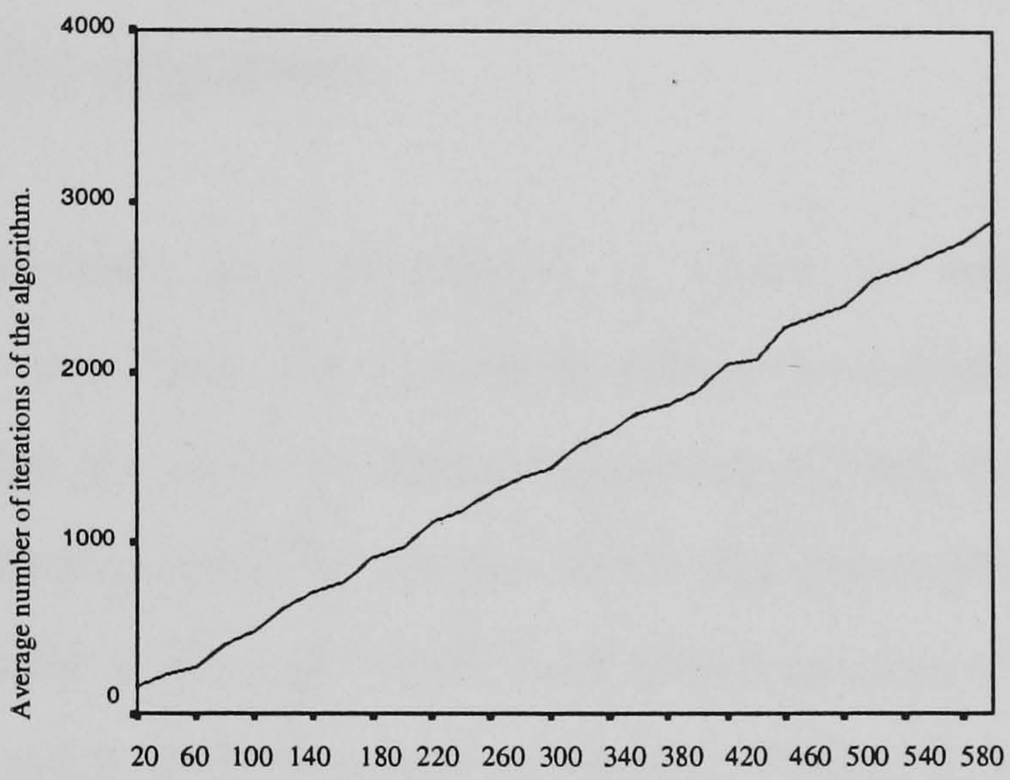


Fig. 63b: Evolution of the search effort.

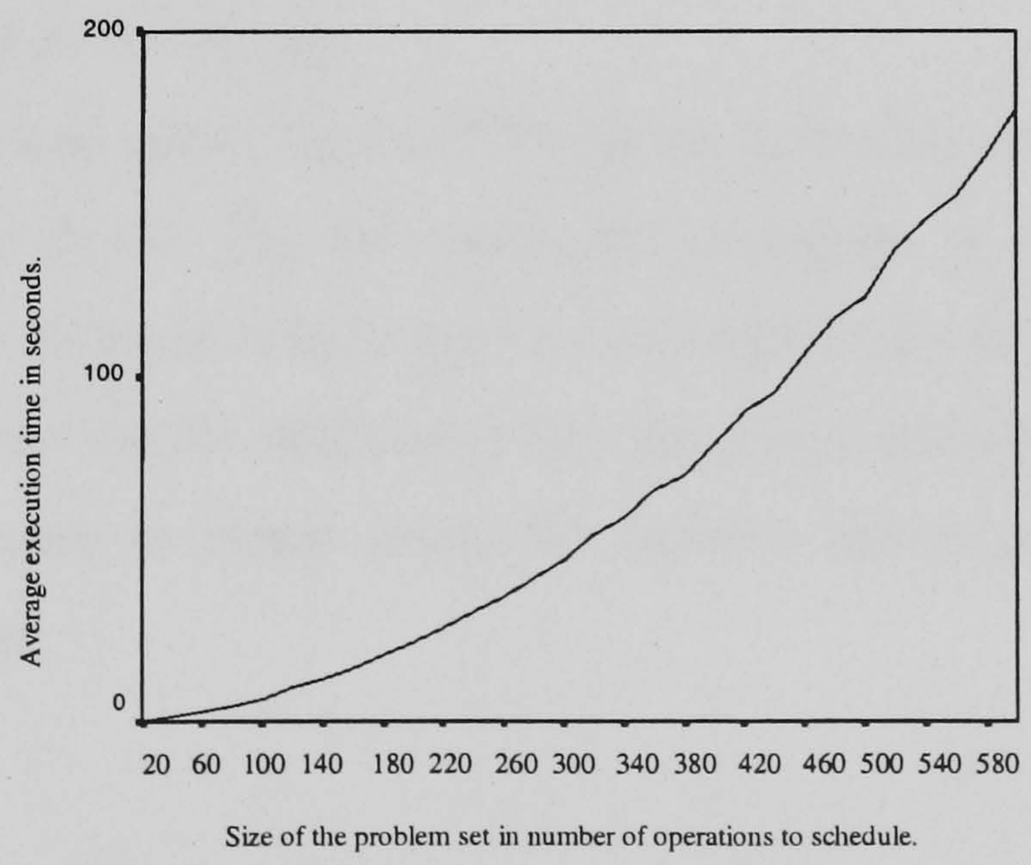


Fig. 63c: Evolution of the execution time (sec).

From the above results the algorithm appears to perform polynomially in the size of the problem. The optimality, expressed as the relative distance to the lower bound, seems to be constant, i.e., no degradation of the solution quality is observed as we increase the size of the problem. Since the lower bound is obtained assuming no machine idle time, the results when considering 1-2 parts per job show a greater distance due to a lower production demand. The quadratic behaviour of the computational time was also an expected value, as algorithms employed in the simulation module of the system have a computational cost of $O(n^2)$, where n is the number of tokens in the marking.

5.2 Optimality of the algorithm.

A first experiment was conducted in order to determine the degree of effectiveness of the algorithm for problems where it is likely that we may obtain optimum solutions that are closer to the lower bound defined by $h^*(M_0)$. A set of 1000 random problems where generated with the following characteristics. The system has 3 machines and 5 different part types (jobs) each with a number of tasks between 3 and 6. 50% of jobs have 2 alternate plans. 85% of operations can be performed by more than one machine. Each operation is assigned a random *ground* cost from an uniform distribution [1..100]. The actual cost of each alternative is randomly obtained from a normal distribution of mean *ground* and variance of 15%. A total of ten parts for each job are to be produced in the system.

Each problem was solved by *DLSS** with the following settings (*Top_Level=10, Max_Nodes=5, Move_At=5). Fig. 64* shows the histogram of the relative difference (*Rd*) of the makespan obtained with respect to the heuristic lower bound $h^*(M_0)$. Notice that *DLSS** may be reaching the optimum as the hypothetical lower bound may be lower than the actual optimum in many cases. We believe this explains the skew of the distribution to the right.

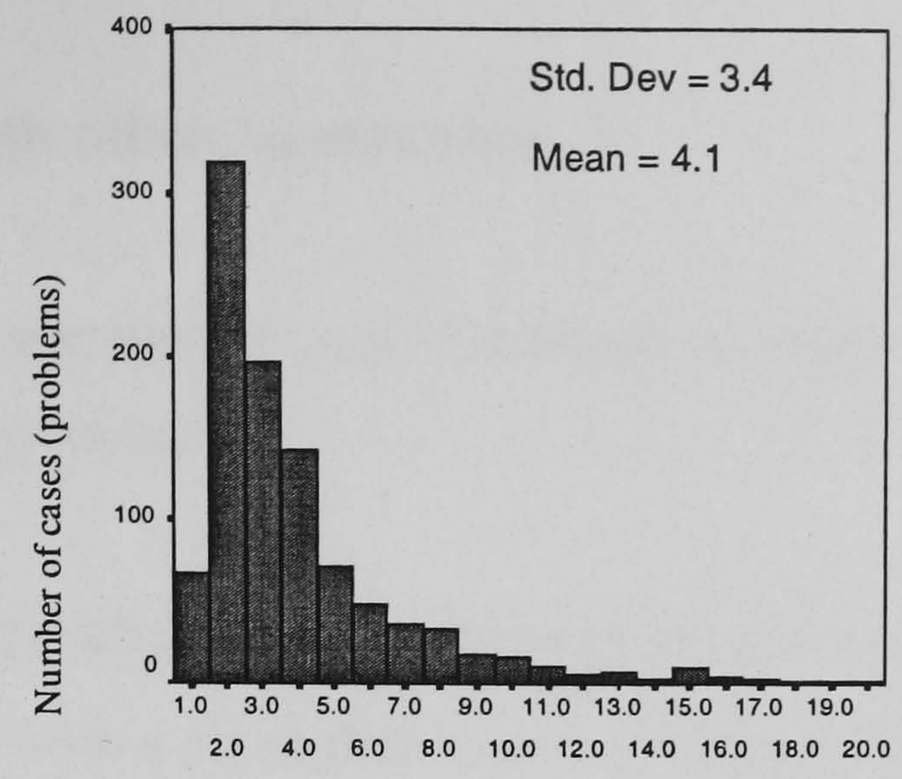


Fig. 64: Relative difference with the lower bound

Each of the problems was also solved with a heuristic dispatching algorithm that selects the next transition to fire based on the *LWRT* (least working remaining time) rule, ties are broken by applying the *SPT* (shortest processing time) rule. This approach is conceptually equivalent to [Abdallah 98]. The makespan obtained is an average of 18.8% times greater than the heuristic lower bound.

Results indicate that the heuristic function quickly directs the search for those *FMS* with extreme flexibility, where balanced machine workload can be achieved (which is a desirable design objective), and there is low variation of operation cost between different alternatives for a *task*.

For problems where a resource is clearly a bottleneck and a balanced work-load is not possible, the estimation of *h(M)* might be too optimistic. However, in our approach, the search effort is mainly controlled by *DLSS**. Again, this is an interesting property, since even for those *ideal FMS*, a machine breakdown can create a temporarily imbalance of the system. In this situation, *h(M)* also becomes too optimistic, and forces *DLSS** to increase the search within the *Search Frame* whilst ensuring that the search will advance towards a solution.

5.3 Comparison with other approaches.

The following experiment was conducted in order to compare *DLSS** with different *PN* search algorithms:

a) *DLSS**(10,15,|*T*| / *2*). Where */T/* is the number of transitions in the system.

b) Non-admissible *A** with a *depth-first* heuristic function [Yim 96]: $h(m) = w \bullet A \bullet E - w \bullet A \bullet depth(M)$. Where *A* is the mean operating cost of all operations and *E* is the nominal number of transition firings from the initial marking to a goal marking. Because there is no concept of concurrency in the total operating cost, *w* explains the extent of reduction of this expression. We set *w* to *2/m* with *m* the number of resources in the system.

c) Incomplete *Branch & Bound* procedure. The search stops when the number of nodes expanded at least doubles the average nodes explored by *DLSS**. We employ h_{RCR} as the lower bound for pruning purposes. For selecting the next branch a combination of the *FWRT* and *SPT* rules is adopted. Such an algorithm can be considered a sophistication to the *B&B* approach of [Abdallah 98].

d) *Beam Search* with a *beam-with* of 80; such a setting explores the same number of nodes as *DLSS** on average. The algorithm uses the same heuristic function as *DLSS** to select the beams i.e., h_{RCR}. We believe such an approach is similar to the *limited-expansion A** proposed in [Sun 96] and the *Beam Search* approach of [Shi 91].

The problem data was 1000 *FMS* descriptions, randomly obtained as follows. The number of jobs is uniformly obtained within the range [5..10], the number of resources in the system is calculated as the number of jobs divided by two. 25% , 50% and 25% of jobs have 3, 2 and one process plans respectively and 65% of tasks have alternate routing and multiple resources may be required to perform a task. Each operation is assigned a random *ground* cost from a uniform distribution [1..100]. The actual cost of each alternative is randomly obtained from a normal distribution of mean *ground* and 33% variance.

Each problem was solved by the four previously mentioned algorithms. The makespan obtained by *DLSS** was taken as the reference and compared with the rest of the solutions obtained. *Table 15* shows the results.

The first column indicates for the percentage of problems for which the algorithm obtains a better makespan than *DLSS**. The other columns show the descriptive statistics for the % relative increment (Rd) of the makespan obtained by each problem with respect with the solution makespan provided by *DLSS**.

	Problems improved	Minimum	Maximum	Mean	Std. dev.
*A** (a)	6.3%	-15.0	94.8	22.8	17.1
B&B (b)	5.4%	-15.9	125	26.0	19.6
Beam (c)	15.6%	-23.0	57.6	8.7	10.2

Table 15: Descriptive statistics.

We experienced tuning problems with the *A** algorithm, the variance of the search effort was high, making difficult a proper comparison; 24% of executions were halted due to memory problems. Results with *B&B* are clearly worst due to the fact that *B&B* bases its strategies on chronological optimisation of a first solution, rather than on accurate local improvement. When *DLSS* is* compared with *Beam Search* a less dramatic difference is obtained. Since both approaches follow a similar optimisation philosophy the difference is explained in terms of the backtracking recovery capacity of *DLSS**.

5.4 Comparison with some benchmarks problems from the literature.

We solved several concrete benchmark problems proposed in two papers. *Table 16* shows the comparison results for three *FMS* problems in [Lee 94]. This paper implements *A** with a non-admissible heuristic function. The results show considerable improvement. Additionally h_{RCR} results in a better heuristic function and *DLSS** allows us to increase the search effort without falling into *breadth-first* search. In [Lee 94] the authors reported that no further improvement of the solution could be made since further relaxing the *depth-first* component of the heuristic function resulted in the algorithm not finding a solution in reasonable time.

Problem	**Reported**	*DLSS**
Lee 94 (a)	426	329
Lee 94 (b)	298	254
Lee 94 (c)	273	237

Table 16: Comparison with benchmarks proposed in [Lee 94].

Table 17 shows results for three problems presented in [Xiong 98]. The optimal solution is given by a pure Best-First technique, while [Xiong 98] proposes a hybrid search technique between a *B&B* and *Best-First.* Results show *DLSS** finds the optimum solution with considerably less search effort (measured in terms of number of markings explored).

Problem	Makespan			Number of markings explored		
	BF	*BT-BF*	*DLSS**	*BF*	*BT-BF*	*DLSS**
1	58	62	58	3437	1687	431
2	100	104	100	9438	8045	856
3	134	148	134	23092	18875	1204

Table 17: comparison results with [Xiong 98].

Also, in this paper, they provide a model of a real Integrated Circuit sort and test floor in San Jose, CA. The system consists of 79 resources, and 30 jobs, each job has three tasks. No alternate routing is available and the total number of operations to schedule is 90. The best makespan reported in [Xiong 98] is said to be 30. *DLSS*(5,15)* found a makespan of 28 in 7 seconds. *Table 18* shows the operation starting times for the schedule obtained.

Operation	Start Time	Operation	StartTime	Operation	Start Time
Job_1_1	0.0	Job_28_2	4.0	Job_2_2	13.0
Job_2_1	0.0	Job_29_2	4.0	Job_8_2	13.0
Job_3_1	0.0	Job_1_2	5.0	Job_12_2	13.0
Job_4_1	0.0	Job_6_2	5.0	Job_19_3	13.0
Job_16_1	0.0	Job_10_1	5.0	Job_21_3	13.0
Job_17_1	0.0	Job_12_1	5.0	Job_18_2	14.0
Job_18_1	0.0	Job_3_2	6.0	Job_2_3	15.0
Job_19_1	0.0	Job_5_2	6.0	Job_7_2	15.0
Job_20_1	0.0	Job_13_1	6.0	Job_12_3	15.0
Job_23_1	0.0	Job_22_1	6.0	Job_14_2	15.0
Job_24_1	0.0	Job_27_3	6.0	Job_11_3	16.0
Job_25_1	0.0	Job_21_1	7.0	Job_13_3	16.0
Job_26_1	0.0	Job_8_1	8.0	Job_7_3	17.0
Job_27_1	0.0	Job_9_2	8.0	Job_15_1	17.0
Job_28_1	0.0	Job_10_2	8.0	Job_26_3	17.0
Job_29_1	0.0	Job_30_3	8.0	Job_28_3	17.0
Job_30_1	0.0	Job_3_3	9.0	Job_20_2	18.0
Job_4_2	1.0	Job_5_3	9.0	Job_24_2	18.0
Job_6_1	1.0	Job_7_1	9.0	Job_25_2	18.0
Job_9_1	2.0	Job_13_2	9.0	Job_8_3	19.0
Job_23_2	2.0	Job_19_2	9.0	Job_22_3	19.0
Job_30_2	2.0	Job_21_2	9.0	Job_14_3	20.0
Job_4_3	3.0	Job_23_3	9.0	Job_24_3	20.0
Job_11_1	3.0	Job_9_3	10.0	Job_29_3	21.0
Job_14_1	3.0	Job_11_2	10.0	Job_16_3	22.0
Job_17_2	3.0	Job_1_3	11.0	Job_15_2	22.0
Job_26_2	3.0	Job_17_3	11.0	Job_18_3	25.0
Job_27_2	3.0	Job_6_3	12.0	Job_20_3	25.0
Job_5_1	4.0	Job_10_3	12.0	Job_25_3	25.0
Job_16_2	4.0	Job_22_2	12.0	Job_15_3	26.0

Table 18. Schedule for the IC problem.

6 Summary.

In this chapter we have presented a *PN* based heuristic search algorithm *DLSS** that we believe overcomes the difficulties encountered with other approaches reviewed. *DLSS** builds on results studied by [Reyes 98]. The algorithm can be integrated with the branching scheme *CGS* presented in the previous chapter and allows the application of a *Best-First* heuristic search based on *PN* heuristic information without experiencing exponential cost. This results in an algorithm that is able to achieve a useful degree of optimality. The results indicate that *DLSS** is highly effective when solving case studies of the literature. When compared with other approaches, experimental tests indicate the superiority of our approach.

Chapter 7. An evolutionary Hybrid scheduler based on *PN* structures for *FMS*

1. Introduction.

The main body of this thesis has dealt with the definition of a scheduling methodology for *FMS* based on the application of systematic heuristic search algorithms within the state space defined by a *PN* based representation of a problem. To handle the complexity problem for large *FMS* instances and maximise the use of *PN* based heuristic search, we have presented an incomplete hybrid search algorithm *DLSS** that successfully handles large problem instances. In the literature consulted, two other interesting methodologies have been followed (see chapter 2, section 4.1). The first is based on problem decomposition techniques which is a well-known [Ashour 67] classic solution. The second is a relatively new technology: *Genetic Algorithms* (*GA*). As reviewed in chapter 2 genetic algorithms *GA*s belongs to the class of iterative optimisation methods based on random exploration of the search space.

From the literature review on these methods, we have concluded the following:

- The hybridisation between *splitting-up* approaches and random optimisation methods appears to be promising.
- Results from *GA* methods applied to manufacturing scheduling suggest that hybridisation of the *GA* main scheme with simulation of heuristics and systematic search algorithms. We believe this connects with the potential of *PN* as a simulation tool and *PN* based algorithms for scheduling *(DLSS*)*.

However, the idea of integrating *PN* capabilities with these methodologies is largely untried. We believe that *PN* are convenient to support the hybridisation of these methodologies as we will explain in this chapter.

The objective of the method that we present in this chapter is a preliminary integration of these two ideas, *GA* and decomposition/progressive construction of schedules with a *PN* representation of the *FMS* and *PN* scheduling based on simulation, heuristics and the *PN* based heuristic search algorithm *DLSS**.

To justify this integration, we present an evolutionary hybrid scheduler called *BSPC* that resulted in a promising method that obtains useful results, demonstrates the use of *PN* with *OR* methods and justifies further research.

This chapter is organised as follows: section 2 gives reviews other relevant work, highlighting the relationship between, splitting up approaches, schedule builder methods and *GA* approaches. Section 3 presents the algorithm and gives a case study. Section 4 provides preliminary experimental results.

2. Background.

2.1 *FMS* scheduling approaches based on decomposition.

The idea of splitting the scheduling problem into more tractable sub-problems has been tried many times; often in combination with other methodologies. We can classify these approaches, depending on how the problem is truncated, as temporal, spatial or hierarchical.

Temporal truncation: The recurrent idea is the *rolling-horizon*. This consists of generating successive partial schedules as production evolves, instead of generating a full schedule for all parts. A job release policy decides which job or jobs to schedule for the next production horizon [Yamamoto 77]. The scheduler then employs a search methodology restricted to these jobs. This general approach is implemented with variations on search algorithms and heuristics, for example [Bispo 92] employs a very restrictive *Beam Search* procedure, while [Liu 92] employs *A**.

Spatial truncation. The scheduling problem formulation is truncated into affordable sub-systems which are solved optimally usually by a systematic method. Then all schedules are joined and usually an iterative heuristic optimisation method is employed to solve conflicts among sub-problems. The truncation is based either on job decomposition [Chu 92] or disjunctive graph representation of machine sequences [Kruger 95] [Byeon 98] [Kruger 98].

Hierarchical truncation. A different approach separates *loading* from *sequencing* and integrates them in a hybrid scheduler. For example, [Shaw 88] [Shaw 88b] [Shaw 89]

generates the best possible route for each job using a linear planner based on *A** and then, following a non-linear planner methodology, combines the sub-solutions to construct the overall schedule by solving conflicts. The most interesting aspect of this work is the use of an iterative algorithm as a plan revision procedure that is also a construction procedure: whenever a conflict arises, one of the jobs causing the conflict is chosen and an alternative plan is proposed. If the new plan enhances the schedule, it is retained. A similar approach can be found in [De 88].

From the literature, we can identify the major weakness of a decomposition approach: it typically is composed of an expensive systematic search with a construction/decomposition strategy heavily based on heuristics. For example, the *Shifting bottleneck* algorithm [Adams 88] is based on a spatial decomposition approach where the single sequencing problem for one machine is solved in an iterative algorithm. Unfortuntely the selection of the next machine to schedule affects the final schedule, specially if it is based on a simple rule. The decomposition approach of [Shaw 88] employs optimal *A** search to schedule a single job, but is forced to employ an iterative construction algorithm based on local search in order to obtain the complete solution.

In an attempt to improve the construction phase recent work has incorporated random search within the general decomposition approach. For example, [Kim 99] modifies the work of [Shaw 88]. First they obtain all the possible routes for the processing a part of each job-type. The scheduling algorithm is a hybrid schedule construction methodology based on neighbourhood search and dispatching rules. The algorithm starts with an initial combination of routes and a random neighbourhood search seeks improvement.

The interest of this work is in the combination of two methodologies, optimal or heuristic search for *sequencing* and random search for *loading*. This hybridisation of heuristic or systematic search and stochastic elements has been seen in approaches that apply Genetics Algorithms to *FMS* problems as we will see below.

2.2 *FMS* scheduling based on Genetic algorithms.

Much of the early work on genetic algorithms used a universal internal representation of fixed-length binary strings with binary genetic operators to operate in a domain-independent fashion [Cheng 98]. However, such simple genetic algorithms are

difficult to apply directly and successfully in optimisation problems. This is the case for *FMS* where determining paths and machine allocation must be considered. As reviewed by [Ulusoy 98], for production scheduling two main approaches for chromosome representation are used: direct and indirect.

In direct, the chromosome models all the information relevant to solve the problem. The advantage is that the search is performed entirely by the *GA* (and consequently the schedule builder is very simple). Unfortunately more complex genetic operators are needed as are functions for repairing illegal chromosomes i.e. chromosomes that represent unfeasible schedules, and this happens because the chromosome representation can not handle all the constraints of the system. Besides, some authors point out that although these *GA´*s can explore large problem spaces and locate possible solution areas they cannot quickly converge on an optimal solution due to the complexity of the algorithms required to produce legal schedules [Uckun 93].

What *GA* seem best able to provide for *FMS* scheduling is high level decisions or strategies for the system. In other words, a simple genetic algorithm can do its work very well if an individual is considered to be a sequence of local decisions to be used as knowledge by a special purpose heuristic algorithm which is known as the *schedule builder*. A *GA* algorithm that follows this approach has an indirect representation.

In an indirect representation, the chromosome does not represent the total sequence and assignment of operations to machines. In other words, the *GA* does not explore the solution space defined by the scheduling problem; it doesn't build the solution directly but provides guidance on how to build the schedule. Here is where *PN* are of interest. First of all, because *PN* are a powerfull and well known modelling tool capable of including all the characteristiques of the *FMS*. The consequence of this is that the problem of unfeasible schedules during *GA* operation is eliminated. But also, as we will see, because approaches employing indirect chromosome representation integrate *GA* with previous and well established methodologies to act as schedule builders: simulation, dispatching rules and systematic heuristic search. Scheduling based on simulation and heuristic/dispatching rules has been largely applied using *PN* as a representation and the previous chapters of this thesis has demonstrated the potential of the integration of *PN* and *AI* based search methods.

The *GA* determines plans, machine selection and job ordering, whilst the schedule builder acts like as heuristic algorithm that solves the remaining sequencing problem (see for example [Uckum 93], [Holsapple 93] and [Ulusoy 97]). There is a

clear similarity between these approaches and the hierarchical decomposition methods of [Shaw 88] and [De 88]. In an attempt to reduce the gap between heuristic procedures and the systematic search employed in decomposition approaches, *GA's* have been used to enhance heuristic decision making in systematic procedures. The *GA* explores the *heuristic space* instead of the *problem space.* (see [Dundorf 95] and [Herrman 95].)

2.3 The proposed approach.

What we propose is to employ the same idea of integrating *GA* with heuristic techniques based on a spatial splitting approach using a *PN* model of an *FMS.* Rather than control a search algorithm by a *GA*, we employ an evolutionary scheme to progressively construct a final schedule from partial schedules, favouring those combinations that improve the schedule obtained. It is our opinion that this represents an interesting alternative to the iterative construction methods of [Shaw 88] [De 88] [Kim 99]. We justify the use of *PN* by:

a) Unlike previous research, mostly based in mathematical representation of the problem, more complex *FMS* representation can be faced. Aditionally, a good model not only allows us to implement schedule builders easily, but also help to track the status of lots and machines efficiently. Which is useful to heuristically guide the decision making part of the *GA* phase.

b) They allow easy structural truncation analysis and decomposition. For example, the top-down synthesis procedure given in chapter 3 for the parsing of *FmsML* descriptions easily support the identification of independent *sub-PN* that represent job descriptions. This potential has already been exploited. For example, in the *process* and *command* circuits of [Hillion 98] and [Proth 98] that separate sequencing from resource sharing. *PN* truncation has also been applied in *PN* representations of discrete event systems (not necessarily *FMS*) [Shen 1992] where the original *PN* model is truncated into two simple sub-nets which are solved by a *B&B* algorithm. The sub-schedules obtained are then joined, solving conflict among subsystems. [Chen 93] and [Chen 94] extended this work by modifying the branch and bound algorithm so that when solving a sub-net (sub-system) there was synchronisation with an already solved sub-system. It is interesting to see how this synchronisation is easily guided by the PN model, by means of identifying related or

conflicting transitions. The algorithm uses this synchronisation to guide the search and avoid futile schedules. Although results are only illustrative, they justify further research and the interest of employing *PN* as a representation tool for hybrids *GA* approaches.

c) A *PN* model allows the immediate and easy application of heuristic algorithms based on dispatching rules and simulation. This is an interesting feature for those *GA* based approaches for production scheduling that require an indirect representation where to simulate the effects of the *GA* output, as for example to evolve a combination of different heuristics [Herman 95] [Fujimoto 96] [Lee 97] [Jawahar 98]. Additionaly a simpler schedule builder based on dispatching rules can be subsituted by a *PN* based systhematic heuristic search algorithm such as *DLSS**.

d) The literature integrating *PN* representation with *GA* optimisation is very scarce. [Chiu 97] and later [Chu 98] propose an embedded *GA* in a *rolling horizon* scheduling approach. Work in progress is initially defined, then the total schedule is generated segment by segment, each segment being the result of running a *GA* search. The *PN* structure in terms of places and transitions is transformed into a direct chromosome representation. Two criticisms can be made to this work. First, it is a direct chromosome representation which means that chromosome needs to be transformed into a feasible sequence of transitions for the *PN* model to simulate the behaviour until the next time horizon. The second is that the *GA* merely substitutes a systematic *A** search employed in a similar *rolling horizon* approach [Liu 92] in an attempt to overcome intractability. While *A** allow *PN* analysis to guide the search, the *GA* proposed in [Chu 98] adopts a direct chromosome representation, which is not a *PN*. This cancels the possibility of employing *PN* heuristics and analysis.

3 *BSPC:* An Evolutionary Hybrid scheduler based in *PN* structures for *FMS*.

The scheduling methodology presented in this chapter integrates *PN* based heuristic search and structural analysis of the *PN* graph for a splitting up procedure. An evolutionary procedure is used to join and sequence sub-problems in which the *loading, routing* and part of the *sequencing* problem have already been determined.

The following steps describe the method:

- A splitting up approach is performed, by grouping jobs or lot sizes or both. The rationale is to use the *PN* graph to identify those jobs that conflict most. This gives a collection of affordable sub-problems.
- Each of these sub-problems is solved using various random settings of *DLLS**. This produces a collection of different, near optimal, solutions for each sub-problem.
- All these initial solutions can be understood as building blocks and form an initial building pool. Each individual has information on *how much of the overall problem it solves* and *how*. This information can be represented as a chromosome, for example: *Solve a single part type of JA using route* r' *and one part of JB using route* r'' is represented as *(1xJA, 1xJB)*. Associated with this chromosome is a sequence of firing transitions that completely determine the schedule over the *PN* model.
- An evolutionary algorithm selects the best partial solutions and combines them to produce new individuals that are closer to a solution. The chromosome structure evolves by creating new building blocks. This evolutionary procedure searches the space of possible combinations of sub-schedules and controls a final schedule construction procedure. This paradigm is conceptually different from the concept of chromosome evolution employed in [I. Lee 97] and [Sikora 96] where the chromosome structure varies from iteration to iteration, but is constant during each *GA* phase.
- When a new individual is constructed, *DLSS** or other specific dispatching algorithm based on *PN* simulation, acts as a schedule builder by solving the remaining conflicts. Note that there is a great deal of information inherited from the parents about the firing sequences associated with each individual. Already scheduled operations do not need to be considered by the schedule builder, which only solves conflicts amongst sub-schedules. This differs from approaches such as [Holsapple 93] where for each new individual, a complete schedule is produced using *Beam-Search*, without identifying previous good building blocks already scheduled.

3.1 Overview of the method.

We present an overview of the main stages of the *BSPC* algorithm in *fig. 65*. For the rest of the discussion, the term individual, sub-schedule and sub-problem are equivalent.

Step1 *Obtain initial* POPULATION
while *not termination condition found* **do**
Step2 SONS = *Combine* (POPULATION)
PARENTS = POPULATION
Step3 POPULATION = *Truncate* (SONS) ∪ Truncate (PARENTS)

Fig. 65: BSPC algorithm

Step 1 summarises the definition of the problem, the truncation of the lot-sizes and the application of the heuristic search algorithms to obtain the initial population. The algorithm enters an iterative procedure where a new set of individuals called *SONS* is obtained as a result of combining individuals ***(Step 2)***. *SONS* represents a set of sub-schedules that are closer to the final schedule. ***Step3*** updates *POPULATION* by eliminating the least fit members. There is no mutation in this initial approach.

In order to give a detailed explanation of each of the components of the algorithm consider the simple *FMS* description in *fig.66* , which is parsed as the *PN* of *fig.67*. The system manufactures two different jobs *JA, JB* and has three machines *M1, M2, M3*. JA consists of two tasks, the first can be achieved by two alternative operations. The first uses machine *M1* and takes three time units. The same operation can alternatively use *M2* taking four time units. The second *task* of *JA* can only be processed by *M2* taking five time units. The initial marking in *fig.67* indicates that the product requirements are two parts per job.

```
#system EXAMPLE;
#resource M1 #end;
#resource M2 #end;
#resource M3 #end;

#job A
  #task A1
    description = M1:3.0 | M2: 4.0;
    buffer = INF;
  #end;
  #task A2
    description = M1:5.0 ;
    buffer = INF;
  #end;
#end;
```

```
#job B
  #task B1
    description = M1:3.0 | M3: 5.0;
    buffer = INF;
  #end;
  #task B2
    description = M1:3.0 | M2:5.0 ;
    buffer = INF;
  #end;
#end;
```

Fig. 66: FMS example.

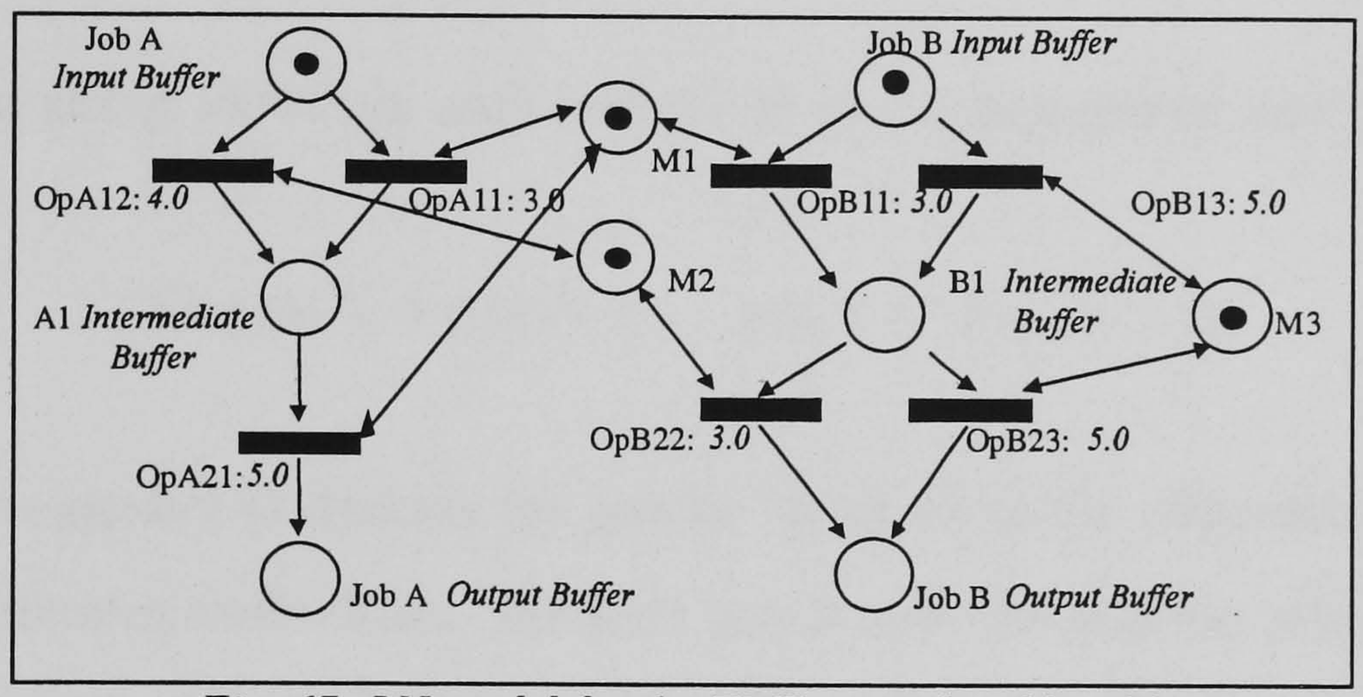


Fig. 67: PN model for the FMS example of fig. 66.

3.2 Truncating the problem.

The first step of the algorithm splits the problem into smaller sub-problems. A first approach might be to consider jobs in isolation and to include a collection of alternative paths for each job as the initial pool. This has the advantage that any lot-size can be obtained from this truncation, and also that they represent easy sub-problems, where no conflicts on the use of resources need to be solved. However, it implies that the effort in the iterative construction phase is greater.

A second choice is to reduce the lot-size by, for example, solving (1xJobA, 1xJobB). Different solutions for this problem will form the initial pool. Note that for an *FMS* including many jobs, a lot-size of one job represents an unaffordable problem for an admissible search algorithm so the use of *DLSS** is justified. The first individuals obtained are now more complex since they have a greater number of scheduling conflicts already solved.

3.3 Obtaining the initial solutions.

Each sub-problem is solved using a *PN* based heuristic search algorithm, (branch and bound, *A**, *DLSS** etc). A collection of different solutions for each problem, including the optimum, can be found in a reasonable amount of time because the size of the problem is small. The search process is guided by the *PN* definition, and heuristic information from the *PN* model is applied.

3.4 Representing each sub-schedule.

A fixed string identifies each individual in the population and looks like this:

$$G: p_1 \times J_1 + p_2 \times J_3 + \ldots + p_n \times J_n \rightarrow t_1, t_2, \ldots t_k$$

The first symbol G denotes the gender (M or F) of the individual and will later be used for combining individuals. the term $p_i \times J_i$ indicates that the schedule of p_i parts of job i are solved by this individual. Finally a sequence of transitions contains the solution to the sub-problem. By simulating such sequence in the *PN* model, we obtain a final marking and the completion time or makespan of the schedule. If we reduce the final marking to the tokens in places representing final output buffers, the number of tokens in these places equals the number of parts produced for each job as expressed in the chromosome. Finally, a makespan-based fitness function *(f*)* is defined to evaluate each individual.

3.5 Building larger solutions.

Once the initial population is obtained, the algorithm enters the evolutionary optimisation process that constructs the overall schedule from the partial solutions in the population.

The first phase of the algorithm combines individuals representing sub-schedules. The entire population forms the mating pool. The gender of each individual divides the mating pool into two disjoint sets named F and M. Combination among

individuals of F and M produces a set of new sub-schedules S. The procedure for obtaining S is ruled by the fitness function and is given in *fig.68.*

Algorithm *Combine(P)*

Receives: P the population of sub-schedules

Computes: *S* the set of sub-schedules resulting from combination of *F* and *M.*

Variables: *F,M* the set of sub-schedules from *P* .

$S = \varnothing$

(1) while *F* is not *empty* **do**

Determine $f \in F$ | *f*(f)* is *minimum*

Remove *f* from *F.*

(a) Determine $m \in M$ | *f*(m)* is *minimum* and *FEASIBLE(f,m) > 0*

s = JOIN (f ,m).

$S = S \cup s$

Remove *m* from *M.*

(b) Determine $m \in M$ | *FEASIBLE(f,m)>0* and *FEASIBLE(f,m)* is *minimum.*

s' = JOIN (f, m).

$S = S \cup s'$

(c) **If** no individual *s* or *s'* has been obtained **then**

Obtain an individual *m'* either from list *dead*

or solve *m´* as the remaining problem

s = JOIN (f, m´).

$S = S \cup s$

goto (1)

Fig. 68: Combination of sub-schedules.

Each individual $f \in F$ within the mating pool will be considered for combination. A priority list among individuals of F is established based on the fitness function.

Initially, *f* tries to combine with the best *m* (a) . If this is possible (the function *FEASIBLE* determines this), a new sub-schedule *s* is obtained from *f* and *m. s* is included in *S* and *m* is removed from *M.* Secondly (b), *f* tries to combine with the best *m* in terms of a fitness function based on the function *FEASIBLE* obtaining *s'.*

FEASIBLE(f,m) determines if *f* and *m* can be joined. If this happens it also gives an estimate of the potential of this combination which is used as a fitness function.

FEASIBLE is a *PN* based simulation function that concatenates the sub-schedules using the *Shortest Remaining Time* rule *(SRT)* for conflict resolution. The final marking obtained after simulation is compared with the goal marking M_F defined for the problem. If any final buffer-place contains more tokens than the same place in M_F then the joining of *f* and *m* is *over-solving* the problem and thus, its combination is not viable. *Fig 69.* shows the algorithm.

Algorithm ***FEASIBLE(f,m)***
Receives: *f, m* sub-schedules.
M_0 and M_F are the initial final marking.
Returns: Determines if the combination of *f,m* is possible and gives an estimate.
Variables: s_f the firing sequence of transitions associated with f.
s_m the firing sequence of transitions associated with *m*.
P the set of places of the PN modeling the system.
M the marking employed for simulation.

$M = M_0$
do
let t_f be the first transition of s_f
let t_m be the first transition of s_m
If $SRT(t_f) < SRT(t_m)$ **then**
Remove t_f from s_f.
Fire t_f in *M*
else Remove t_m from s_m
Fire t_m in *M*
while s_f or s_m are not null sequences

if $(M(p) \leq M_F(p) \ \forall p \in P)$ **then return** *success* and the makespan associated to *M*
else return *failure*

Fig. 69: Algorithm Feasible.

When the evolutionary phase has progressed to the point where the population is close to a final solution (schedule), it may be that, for a given *f* it is not possible to find any *m* that satisfies the conditions expressed in *1)* and *2)*. In these situations, and in order to force the creation of individuals that complete the search, an individual not in the population needs to be determined.

To deal with situations of this kind (see case (c) in *fig. 68)*, the approach includes a mechanism that preserves the best of previously existing sub-problems. A

limited list of *dead* individuals is maintained. When the process of selection discards individuals, each is considered for inclusion in a *dead* list. The smaller the sub-schedule, the higher chance it has to be included in this list. If an individual that solves the same sub-problem already exists in *dead,* the one yielding the best fitness function is kept. The rationale is to maintain the best solution to small sub-problems that can complete a sub-schedule F that is almost a final solution. If no individual that is *FEASIBLE* with f can be found both in the population and the *dead* list, an *artificial* sub-schedule m' must be obtained. m' is obtained by determining the remaining parts to be produced after f to achieve the goal. *DLSS** is used to solve m'. And a new individual representing a complete solution is obtained by combining m' and f.

It must be noted that this situation will only occur when f is almost a complete schedule and thus in the final stages of the algorithm. In such cases, m' will be a small sub-problem not previously considered.

3.6 Actualising the population.

Once the combination phase (step 2 in *fig 65*) has terminated, *SONS* contains a collection of new sub-schedules. Based on the fitness function $f^*(m)$ the best k individuals of *SONS* are retained. If the size of the population is g, the best $g - k$ individuals of *POPULATION* are also kept (step 3). Initially we must consider values of k equal to half the number of new sub-schedules in *SONS.* This forces the population to progress towards individuals that are closer to the final solution.

3.7 Joining two sub-schedules: *JOIN* procedure.

The purpose of joining two sub-schedules is to combine two operation sequences and minimise the idle time of the resulting schedule by filling the machine idle time gaps. This new schedule maintains the partial order of operations for both sequences (a problem already solved), and the total machine operation cost, since no other machine assignment is made, however, the total machine idle time should be reduced.

The simplest procedure is based in a heuristic-dispatching algorithm. A dispatching rule, *Shortest Remaining Time* assigns higher priorities to those operations

that can be applied early. The feasibility of the schedule is assured, since the *PN* model contains all the constraints of the system. The algorithm is described in *fig. 70.*

Algorithm ***JOIN(f,m)***
Receives: *f, m* sub-schedules.
M_0 is the initial final marking.
Returns: s_s the sequence to be constructed
Variables: s_f the firing sequence of transitions associated with f.
s_m the firing sequence of transitions associated with *m.*
M the marking employed for simulation.
n the number of jobs in the system.

$M = M_0$
do
Let t_f be the first transition of s_f.
Let t_m be a the transition yielding the smaller $SRT(t_m)$
of the first *n* enabled transition of s_m
If $SRT(t_f) < SRT(t_m)$ then
Remove t_f from s_f.
Add t_f to s_s.
Fire t_f in *M*
else Remove t_m from s_m
Add t_m to s_s.
Fire t_m in *M*
while s_f or s_m are not null sequences
return s_s as the final sequence

Fig. 70: Algorithm Join.

The sub-schedule s_m associated with each *m* controls the execution of the algorithm. At each iteration, the next potentially enabled transition t_m of s_m is examined in terms of the *SRT* rule. $SRT(t_m)$ is the time needed for t_m to become fireable under the *PN* marking *M*. The next *n* potentially enabled transitions of s_f are evaluated by the *SRT* rule, *n* being the number of jobs in the system. If any transition t_f has an $STR(t_f) < STR(t_m)$ then t_f is fired, otherwise t_m is fired. The transition selected is *fired* under *M* and added to the final list s_s.

The algorithm *JOIN*, although simple, produced interesting results, which suggest that a better dispatching algorithm employing limited look-ahead (such as *DLSS*)* and enabling alternate machine selection would improve the results.

3.8 Calculating the fitness function.

Associated with each individual is a fitness function f*(m) that is used during combination and to determine whether the individual is retained. As individuals represent sub-problems of various sizes the fitness function must be normalised. In fact, *f*(m)* is an upper bound on the final solution that we might get if we select *m* to be combined. *f*(m)* is calculated differently depending on wheter *m* is a new individual or it is already included in *POPULATION*.

a) If *m* is a new individual obtained, *f*(m)* is equal to the makespan of the schedule obtained as follows: First we simulate the schedule associated with *m* as many times as needed without over-solving the problem. If remaining operations to complete the number of parts are left (a simulaton of *m* produce more parts than the required by the problem), individuals from the population are considered, and their schedules simulated in the *PN*. When we have scheduled the totallity of parts, *f*(m)* is the makespan obtained.

b) When the individual *m* already belongs to the population, its *f*(m)* is recalculated at each iteration by the following expression:

$$f^*(m) = \frac{f^*(m) + \sum f^*(m_i) \;\; m_i \in S_m}{1 + |S_m|} - w \bullet \alpha + w' \bullet \beta$$

Where S_m is the set of new individuals that have been obtained by combining *m* with some other sub-schedule. Consequently *f*(m)* is recalculated at each iteration by calculating the average of *f*(m)* together with the sum of the fitness values of the new individuals in S_m. The rationale is to include in *f*(m)* an estimate of *how good m* is when combined with other sub-schedules. The term $-w \bullet \alpha$ is intended to favour sub-schedules whose combination produces new individuals which have not been rejected after the selection phase. *w* is a tuning parameter and α is the number of individuals of S_m successfully included in the population after truncation of *SONS* (see step 3 in *fig. 65*). On the other hand, there is a need for the population to progress towards larger

solutions, hence, *older* individuals are also penalised. The term $w' \bullet \beta$ models this, with β the number of iterations since m was created. The preliminary results were obtained with $w = w' = 1$.

3.9 Termination of the algorithm.

For evaluation purposes, we have established the following condition for termination: The population size is kept constant until a first solution to the problem is obtained. This schedule is taken as a candidate solution to the problem and the population size is decreased by 1. If a new solution is found it is compared with the candidate obtained so far and the best kept. Notice that a schedule that is a solution does not allow further combination, this is why, for practical reasons we have decided to decrease the population. Consequently, a total of G solutions will be obtained before the algorithm terminates.

3.10 A case study.

Let us consider the problem of *Fig 66*. The problem is to schedule a number of 3 parts per job type. The problem is truncated by considering a single part for each job and we have optimally solved this problem. There are eight possible active schedules for this problem (see chapter 5, section 3), and *DLSS** was configured to find them all. *Table 19* shows these solutions which form the initial pool of individuals for the construction algorithm.

Individual	Chromosome	Makespan	*f*()*	Schedule
1	*F*: 1xJobA+1xJobB	8	24	*OpA11, OpB13, OpA21, OpA23*
2	*F*: 1xJobA+1xJobB	9	27	*OpA12, OpB11, OpA21, OpB22*
3	*M*: 1xJobA+1xJobB	9	27	*OpA12, OpB11, OpB23, OpA21*
4	*M*: 1xJobA+1xJobB	9	27	*OpA12, OpB13, OpA21, OpB22*
5	*F*: 1xJobA+1xJobB	10	30	*OpA11, OpB13, OpA21, OpA23*
6	*F*: 1xJobA+1xJobB	10	30	*OpA12, OpB13, OpA21, OpB23*
7	*M*: 1xJobA+1xJobB	11	33	*OpB11, OpA11, OpB23, OpA21*
8	*M*: 1xJobA+1xJobB	11	33	*OpB11, OpA11, OpB22, OpA21*

Table 19. Initial population of sub-schedules.

Notice how the fitness value is calculated for each individual as 3 times the makespan of the sub-schedule (see section 3.8).

After the first iteration of the algorithm, a total of 8 schedules were created resulting from combination. The four with the best value of f^* were selected to form part of the population. As a consequence, 4 individuals from the original population were discarded based on the fitness function f^*.

Table 20 shows the population after this first iteration of the algorithm.

Individual	Chromosome	Makespan	*f*()*
1	*F*: 1xJobA+1xJobB	8	22.5
2	*M:* 1xJobA+1xJobB	9	23.4
3	*M*: 1xJobA+1xJobB	9	24.2
4	*M*: 1xJobA+1xJobB	9	24.8
9	*F*: 2xJobA+2xJobB	13	21
10	*F*: 2xJobA+2xJobB	14	22
11	*F*: 2xJobA+2xJobB	15	23
12	*M*: 2xJobA+2xJobB	16	24

Table 20.Population after the first iteration.

Notice how the fitness function for individuals 1, 2, 3 and 4 is now less than 3 times their makespan. The reason is that their combination has produced successful sub-schedules. On the other hand, new individuals have been added to the population. The

individual with the best fitness function (9) also corresponds with the optimum solution for producing two parts of each job. This individual resulted from joining sub-schedules 1 and 4, *fig. 71* shows their Gantt chart. The resulting schedule after applying the algorithm *JOIN* is shown in *fig. 72.* Indices show the part number.

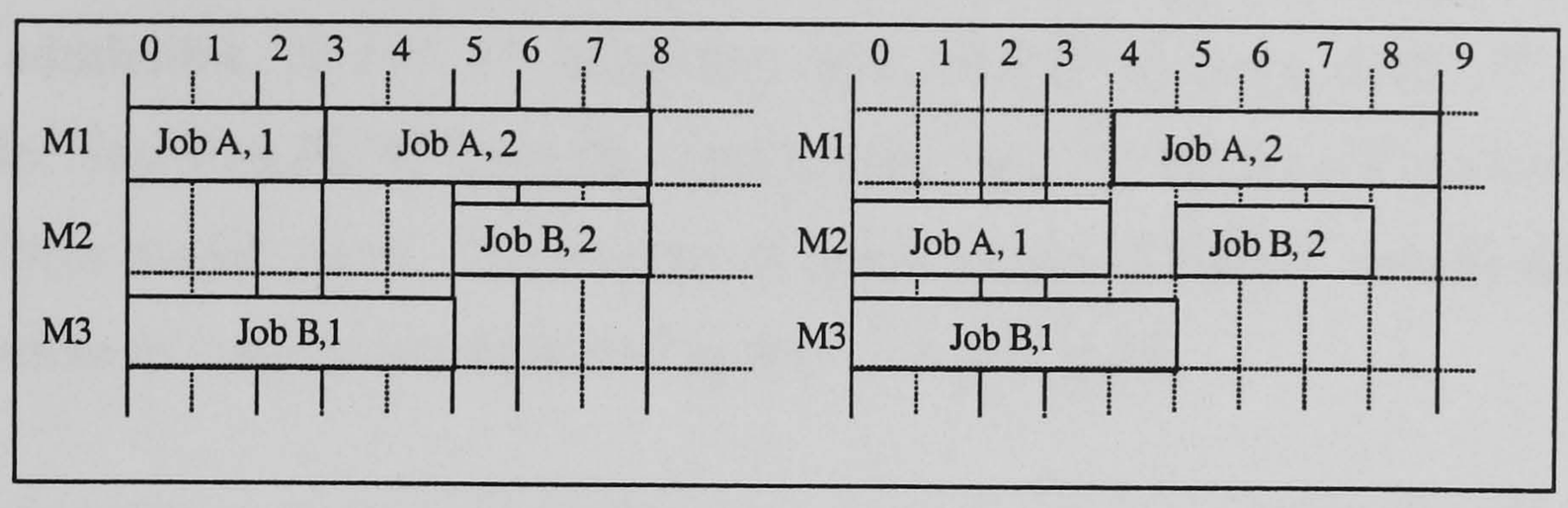


Fig. 71: Gantt Charts of sub-schedules 1 and 4.

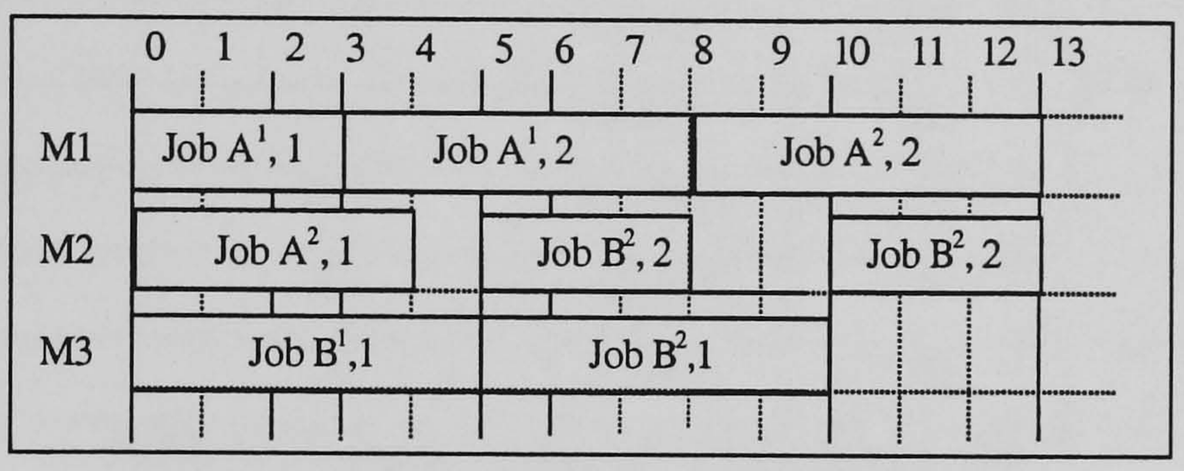


Fig.72: Gantt Chart for sub-schedule 5.

The next iteration of the algorithm produced several final solutions, the best solution (which is also the optimum solution for the problem) is found by the combination of schedules, 9 and 4. The makespan obtained was 18 and *fig. 73* shows its *Gantt* chart.

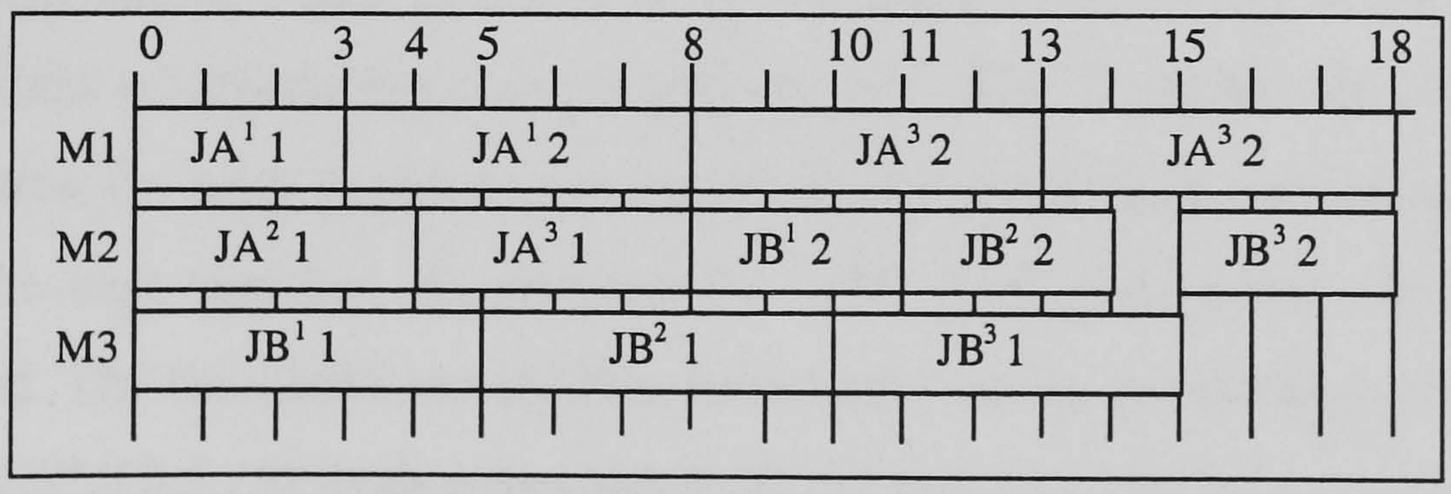


Fig.73: Gantt Chart for final solution.

4 Experimental results.

Table 21 shows the results obtained applying *BSPC* to the example problem of *fig. 66*. The number of parts per job varies from 2 to 6. Each problem was also solved using the admissible *cb-PN A** algorithm and *DLSS** when a pure *A** becomes unaffordable. Note that *BSPC* finds the same solution as *A**. The number of iterations of both algorithms is also given. The number of nodes explored for *A** and the population size and number of generations obtained by *BSPC* is also given.

Number of Parts per job.	Makespan *A**/ (*DLSS*(10,100))	*A** Iterations	*Makespan BSPC*	Population size /Iterations
2	13*	143*	13	8/1
3	18*	1029*	18	8/2
4	24*	17.787*	24	16/3
5	31	10.317	31	16/6
6	36	13.496	36	24/6

Table 21: Summary of results A(DLSS) vs. BSPC.*

We also solved the case-study proposed in [Lee 94]. The best solution that we have obtained for this problem using *DLSS** is 329. The best solution reported by [Lee 94] using *A** was 426. The problem consisted of scheduling ten parts for each of the five different jobs in the system.

We solved the problem using the *BSPC* algorithm with a pool size of 20 individuals that are obtained as the best 20 solutions obtained by *DLSS** solving the reduced problem of scheduling a single part per job. *Fig. 74* shows the evolution of an *obvious* solution for each generation in the sense that a schedule can be built by simply replicating the best member. At iteration five (the sixth generation) the first solution (334) is found. The final solution (332) is found after seven generations. Notice that, for example, *DLSS*(10,5,15)* finds a makespan of 334 and *DLSS*(20,20,20)* obtains *329*.

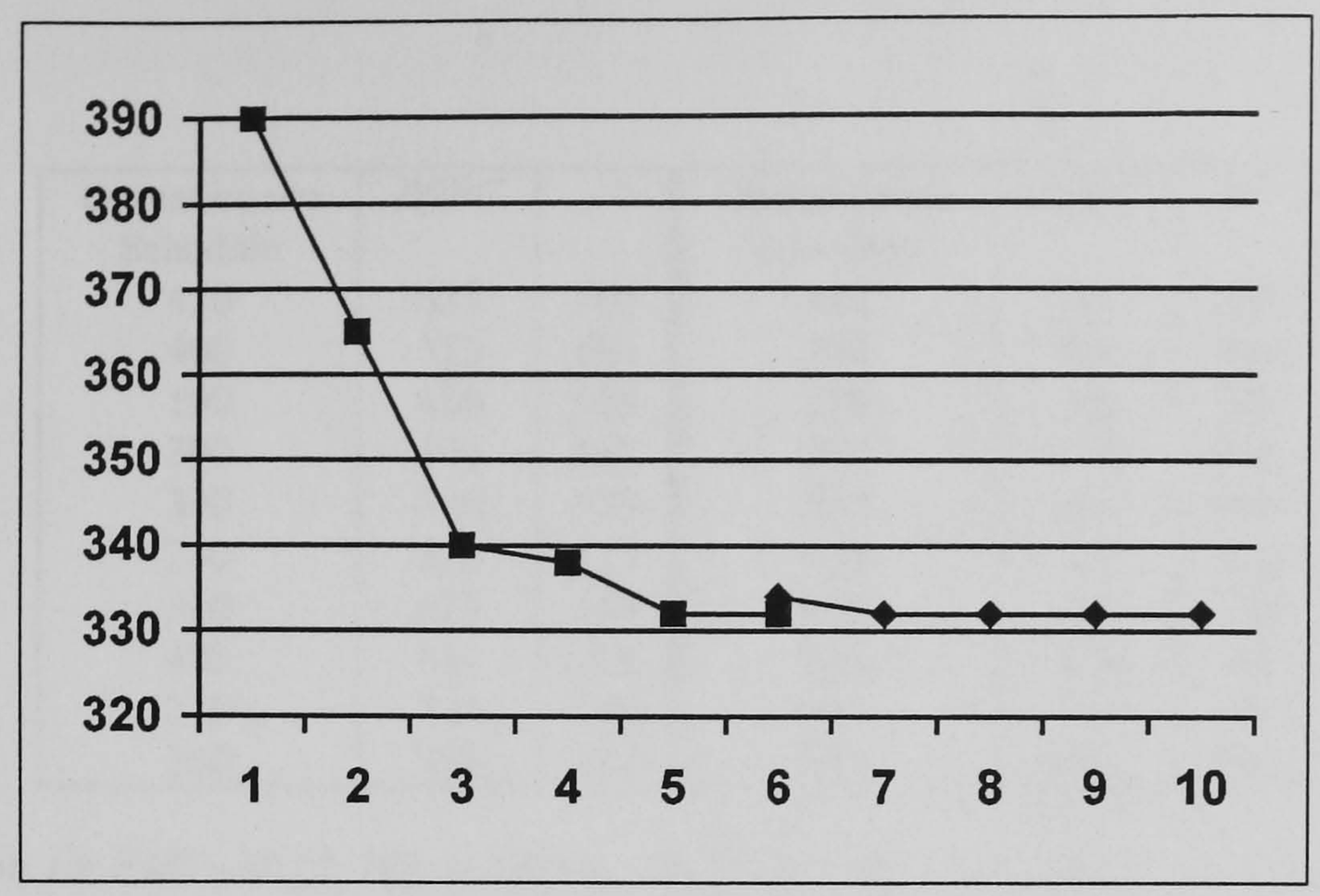


Fig. 74: Convergence of the algorithm

The second experiment consisted of generating 20 random *FMS* descriptions corresponding to systems where the optimum solution should be close to the theoretical lower bound $h^*(M_0)$. The lot sizes per job were set to 10. Each problem was solved by *BSPC* with an initial population of the best 20 solutions obtained by 36 random settings for *DLSS**. These problems were taken from [Reyes 99] were a preliminary version of *BSPC* employing *DWS* was presented. The initial sub-schedules were solutions to the *one-part-per job* problem. *Table 22* shows the results.

Lower bound	*BSPC*	Lower bound	*BSPC*
383.33	393	426.66	446
323.33	338	513.33	528
450	459	336.66	357
360	390	433.33	442
423.33	433	436.66	456
393.33	400	320	335
510	525	316.66	325
360	384	370	382
300	321	333.33	342
393.33	403	383.33	389

Table 22: Results for FMS with three machines, 10x5 jobs to schedule, and a random number of tasks between three and seven. The average number of operations to schedule was approximately 220.

The average relative difference from the lower bound is *3.8%* which is close to the results obtained with *DLSS** for problems of the same type in a similar experiment performed in [Reyes 00].

Table 23 represents another set of problems generated where the parallelism between machines is known to be low. The settings for *BSPC* were the same as for the previous experiment. Each problem was also solved by the *A** approach of [Lee 94].

Operations to Schedule	*BSPC*	*A**	Operations to schedule	*BSPC*	*A**
450	623	705	410	716	898
400	533	601	380	566	661
190	480	509	210	386	382
390	496	562	460	718	844
390	556	575	270	568	698
240	380	417	430	665	746
360	625	649	330	502	578
400	631	704	219	438	461
350	764	780	210	313	345
380	799	872	350	528	555

Table 23: Results for FMS with [5..10] machines, 10x [5..10] jobs to schedule, and a random number of tasks between three and seven.

The relative difference of makespans has a mean of *10.7* which improves the results obtained in [Reyes 98] when comparing *DWS** with *A** guided by the heuristic function of [Lee 94].

5 Summary.

This chapter has presented a preliminary hybrid scheduling algorithm for *PN* structures of *FMS* that is an alternative to the application of *DLSS** for large problems. The objective was the integration of *PN* capabilities – heuristic search based on state space representation and structural analysis- with several successfully employed techniques – *GA* optimisation ideas, and splitting up approaches – in a hybrid scheduling paradigm. It represents a promising alternative to approaches that employ splitting up with construction techniques [Shaw 88] [De 88] and [Kim 99]. In addition the approach supposes an improvement and generalisation of the preliminary results based on *PN* truncation proposed in [Shen 92] and [Chen 94]. Experimental results show that the performance is good compared with the current work integrating *PN* and heuristic search. Compared with *DLSS** the *BSPC* is likely to better handle the problem of local optima (see section 4.8 in this chapter) for irrevocable strategies based on local optimisation, since the adaptive construction can be seen as a global procedure to determine the order in which jobs are produced.

In relation to *GA* paradigms the dynamic evolution of the *chromosome* structure presents advantages over global approaches by identifying good structures in a

progressive schedule building methodology. The inheritance of partial schedules from individuals (no need to re-schedule using the schedule builder) is also an interesting property since it avoids repetition of previous work.

The approach retains the benefits of a *PN* implementation. It allows easy truncation analysis, the application of heuristic dispatching rules is immediate, and application of state space search methodologies is straightforward.

The preliminary results obtained suggest further research on the method. Particulary in terms of the integration of the three basic *GA* operators (selection, crossover and mutation) to the *BSPC* algorithm.

Chapter 8. Conclusion and future work.

1. Summary of the thesis.

The work presented in this thesis has resulted in the integration of *PN* theory with *AI* heuristic search techniques for the scheduling of *FMS*. We have studied different issues that impact this integration and concentrated on the development of *PN* based methods that are useful to guide *AI* based search algorithms. This has resulted in a *PN*-based scheduling algorithm that in the tests that we have conducted improves existing results. The following will summarise the work presented and briefly describe its contribution.

Chapter 2 has reviewed current theory and methods applied to *FMS* scheduling problems. We presented the integration of *PN* and *AI* search methods as a promising way to overcome the modelling difficulties of operational research approaches which lacks a powerful representation paradigm, and the less effective scheduling methods based on heuristic rules and simulation that can not cope with the complexity of *FMS*. Finally, we presented an exhaustive review of previous work that attempted to solve manufacturing scheduling problems by the integration of *PN* and *AI* based searches.

Chapter 3 presented a language for defining *FMS* called *FmsML* that allowed the automated synthesis of *PN* models that capture the main features of the *FMS*. It has been shown that a *PN* results in a natural way of representing a state-space search problem via reachability analysis. We have defined a class of *PN* (*cb-NETs)* that allow the easy inclusion of high-level constraints within *PN* models and which allow easy automatic synthesis of *PN* models for *FMS*. No previous work has considered buffer residence constraints such as *zero-wait* in *PN* based heuristic search for the scheduling of production processes. Examples were given to illustrate the modelling and scheduling of different storage policies.

Chapter 4 studied the development of a new heuristic function based on *PN* structures that model *FMS* descriptions. The heuristic function plays an important role in *PN* based search algorithms. The new heuristic function developed has resulted in an *admissible*

and easy to calculate heuristic function that takes into account the current state of the system expressed by the marking of the *PN*. This heuristic function is clearly superior to previous heuristics which are either not admissible nor based in *PN* information [Lee 94] [Yim 96] [Sun 94] [Inaba 98] or are computationaly expensive and require mathematical expression relaxation [Jeng 98]. In addition it can be applied to problems where multiple parts per job are considered. Experimental tests have shown the benefits of our heuristic function in terms of predicting *what has to be done* based in *what has been done*. This has resulted in greater search reduction when, compared with other approaches, and has eliminated tuning difficulties observed in other work.

Chapter 5 studied the search space that is defined by *PN* models of *FMS* and has proposed methodologies to achieve effective search reduction. The first part studied the identification of intermediate states (markings) which have been previously unexplored, and has proposed heuristic tests based on the *PN* markings that our experiments showed to improve previous methods. We believe this work has resulted in a better understanding of an issue that has not been sufficiently addressed in the literature. The second part has presented a branching scheme based on *PN* analysis that controls the generation of candidate markings by not generating partial schedules that are known not to yield optimum schedules. To the best of our knowledge, no other previous work has applied such a methodology. The experimental results have shown the effectiveness of the approach, in particular for scheduling approaches that do not admit historical records of previous explorations e.g. *(B&B)* and for incomplete search procedures where the number of candidate schedules is limited and thus it is interesting to avoid futile schedules.

Chapter 6 has dealt with the development of an affordable incomplete search algorithm that can be integrated with the heuristic *PN* based information presented in the previous chapters. Previous *PN-based* search algorithms are analysed and their shortcomings reviewed. The problems occur because they rely excessively on *depth-first* search methods [Lee 94] [Yim 96] [Xiong 98] due to the vast search space, thus inhibiting the successful application of *PN* information to guide the search. Although attempts have been made to relax such *depth-first* dependency, either they cannot successfully fight the combinatorial explosion [Jeng 98b] or severely restrict the recovery capability of the search [Sun 94] [Inabla 98]. The novel incomplete search algorithm *DLSS** is built from

this analysis of previous methods and the study of two strategies based on *PN* heuristics and *FMS* operation assumptions. The first exploits the *PN* analysis capabilities of the system to obtain heuristic dispatching rules that are employed to identify very unpromising alternatives; the second assumes a limited backtracking capability based on the hypothesis that the performance of the *FMS* does not depend on a single decision made at an earlier stage, but on a complex and difficult to identify strategy. The experimental results showed that the algorithm was easy to tune and performed polynomially, thus avoiding the combinatorial explosion. It showed very promising results in terms of optimality for well-balanced systems, and produced better schedules than previous methods for more general *FMS* formulations. The superiority over previous *PN* approaches [Lee 94] [Xiong 98] was demonstrated by the solving of various benchmark problems.

Chapter 7 presented a novel approach that extended the results obtained in *PN* based heuristic search with the current trends from *OR* towards stochastic methods. We believe that it represents an interesting result in the rare literature on *PN* based scheduling that employs operation research methods, and provides an improvement and generalisation of the preliminary results based in *PN* truncation proposed in [Shen 92] [Chen 93] and [Chen 94]. It represents a promising alternative to the recent approaches that employ splitting up with construction techniques [Shaw 88] [De 88] [Kim 99]. Experimental results show that performance is close to current works integrating *PN* and heuristic search. We believe this justifies the viability of the approach. Finally, we believe that the method demonstrates the benefits of a *PN* based implementation as *PN's* allow easy truncation analysis, the application of heuristic dispatching rules is immediate, and application of state space search methodologies is straightforward.

2. Current and future work.

Current and future work at the moment is concentrated in two directions. The first is devoted to improve the algorithms presented in this thesis. The second is more directed to the application of the algorithms to real-time scheduling and to real industrial systems.

2.1 Further improvement and development of *DLSS** and *BSPC* algorithms.

The following are a collection of affordable research directions that concentrate in the improvement and further analysis of the *PN* based scheduling algorithms, *DLSS** and *BSPC* that we believe of interest and that are based on the findings of our work.

2.1.1 A sophistication of *DLSS**.

Geometry of *SF*.

In our implementation of the search frame *(SF),* we have considered *max_nodes(l)* to be equal to a fixed constant for every level. This defines a rectangular shape that can allocate a maximum number of markings given by the expression *[top - bottom] • max_wide.* A second approach increases the number of nodes to consider as we approach the *top* level. Since nodes included in *bottom* level represent the limit for backtracking, it seems to be interesting to reduce the choices as we are further from *top* level. It will be interesting to study if these variations in the geometry of *SF* enhance *DLSS** capability of selecting the most promising markings.

Study of different heuristics for advance of *SF* in *DLSS*.*

The study of the rules that control the advance of *SF* in *DLSS** algorithm is an interesting framework within which to perform empirical experiments, and many ideas can be tried. The following is a collection of other different approaches that we believe of interest:

- Compare the overall quality of markings in *SF* (such as the mean and std. deviation); with a certain reference value based on previous history.
- Study the degradation of the heuristic function along *bottom* and *Top* levels of *SF*.
- Consider at least a minimum number of nodes in *SF,* either at each level, or in a subset of levels.

It is clear that two parameters will be relevant: a) the heuristic *f(M)* value of the markings and b) the number of nodes contained in *SF*; but others may be considered.

Inclusion criterion in *SF*.

Further complex analysis could be performed to determine if a marking would be considered for exploration. Note that our approach only considers markings of the same level, but other levels can be taken into account. For example, include a marking in a level not only if it improves the quality of the level, but also if the marking represents an improvement over markings from lower or upper levels. Additionally, we could improve the heuristic rejection of transitions by incorporating a large number of traditional dispatching and heuristic rules [Y-D. Kim 90] in *HST*.

DLSS* **and deadlock avoidance policies.**

*DLSS** is implements itself a deadlock avoidance policy. Obviously, previous works on the deadlock problem in *FMS* with the use of *PN* analysis may be suitable for use in *DLSS**. For example, [Abdallah 98b] describes a deadlock avoidance policy based on *PN* information and dispatching rules. *DLSS** may substitute for such a simple dispatching rule approach. [Hsieh 94] employ *PN* analysis to derive an algorithm that checks whether the execution of a control action is valid to maintain a deadlock free policy, independently of the dispatching policy employed. Similar work on these lines is found in [D'Souza 93] [Banaszak 90] [Ferrarini 98] [Wiswanadham 90]. Incorporating these techniques into *DLSS** may improve the capability of the algorithm to avoid exploration of unfeasible schedules by an earlier detection of deadlock situations.

Minimisation of the local optima effect caused by the backtracking limitation of *DLSS**

In chapter 6, section 4.8, we showed that *DLSS** suffered from a kind of *SPT* strategy, scheduling the fastest jobs first, leaving the longest for the end. However, despite the capability of h_{RCR} to attack this problem, we showed that modifying the heuristic function to avoid this *SPT* strategy improved the results. It will be of interest to integrate well-know heuristic strategies for job sequencing. For example, the work proposed in [Santos 96] reviews several heuristics for sequencing jobs in flow shop systems (*FSS*). Such heuristics are based on the evolution of the operation times within a job description. It should be noted that in a *FSS* all the jobs follow the same path, and usually the problem is reduced to finding a sequence of jobs, given a fixed dispatching

policy, for example a *FIFO* policy [Santos 95]. Establishing a parallel, we can consider the *FMS* problem as finding a sequence of jobs, given a scheduling policy, which is *DLSS**. The sequencing policy will add a component of global strategy that balances the tendency of an incomplete *best-first* strategy to fall into local optima.

2.1.2 A full *Genetic Algorithm BSPC* approach.

Despite the fact that *BSPC* presented in chapter 7 is an adaptive algorithm, one cannot claim that the approach falls into the class of Genetic Algorithms. The following subsections will discuss the integration of the three basic *GA* operators (selection, crossover and mutation) to the *BSPC* algorithm. We suggest integration with different *GA* methodologies that have been successfully applied to manufacturing scheduling problems.

Probabilistic selection of individuals for reproduction.

The way reproduction is performed in *BSPC* is deterministic, although based on the fitness function. This could be modified in order to consider a random selection of pairs of individuals based on the probability distribution defined by the fitness function.

Crossover.

BSPC defines a unique operator: the *schedule joiner*. It takes two partial sub-schedules and obtains a new schedule that inherits all the characteristics of both parents. A general crossover that inherits partial information from the individuals could be introduced.

Obviously, non-homogeneous individuals form the population. This is due to *BSPC* using a progressive construction method. The crossover operator optimises existing plans and schedules for parts and introduces new variants, the schedule builder identifies good building blocks and creates new individuals that are closer to the final solution. Consequently, the dual application of operators contributes in a different manner to the final solution.

The work in [Hsu 96], although not focussed on manufacturing scheduling scenarios, provides an interesting analysis of *GA* performance and insight on our

approach. The main idea is to employ different genetic operators that are priced according to their contribution to the enhancement of the solution. Their analysis leads to the conclusion that certain *GA* operators (crossover, mutation, interchange) have advantages in different stages of search or for different population groups. For example, they state that crossover may be ineffective in earlier stages because of bad genes that are present in the parents. Additionally, individuals with higher fitness values have a higher probability of breeding better offspring.

In other words the two operators crossover-schedule builder and the schedule joiner are executed in each generation but the population over which they are applied changes both in type and size. The appropriateness and usefulness of crossover and mutation varies with population and, consequently, stage of search. On the other hand, the schedule joiner will be used for those sub-schedules identified as good building blocks so their combination contributes towards rapid convergence on the final solution.

Mutation.

Mutation is perhaps the most confusing operator from the work applying *GA* to production scheduling. Most of the work employs traditional mutation operators in others, for example [I. Lee 97], the mutation implements a neighborhood movement which is accepted if it supposes an enhancement of the chromosome, if not, *simulated annealing* is employed. In [Kanet 91] mutation is simply an elimination of a random member of the population and its substitution by a randomly generated chromosome. Some work simply does not consider mutation [Dundorf 95].

As said previously the aim of mutation is to introduce random variations that may have not been explored with the aim of avoiding local optima. It is worth noting that, when an indirect representation is employed, a fixed deterministic heuristic algorithm is in charge of building the schedule. This means that the chance of climbing local peaks may be higher. Introducing a certain amount of random behaviour in the schedule builder may be an elegant alternative to mutation. The crossover operator employed in [Yamada 92] based on *Giffler & Thompson's* algorithm, employs such an approach by randomly determining the next operation that the algorithm selects at different stages of the algorithm. In [Holsalpple 93] the *Beam Search* schedule builder includes a random behaviour, hence guaranteeing that even two identical chromosomes identical after *Beam Search* would obtain different schedules.

[Ombuky 88] propose that the chromosome not only represents the schedule in terms of the order of jobs in the machines but also employ a gene to evolve the heuristics that may be used during the *GA* phase. This gene is obviously affected by mutation and crossover. They call this *genetized-knowledge*. This is clearly related to the problem of which combination of heuristic rules assigned to resources solves the problem best.

What we propose is an integrated approach between a chromosome that represents a solution, and a chromosome that represents how to obtain the solution. Instead of evolving the combination of different heuristic rules, it would be interesting to explore which different schedule builders/joiners are available to be used during the *GA* search, from simple dispatching rules, to more complex heuristics (*Giffler & Thompson's* algorithm, *Shifting botleneck* procedure) and heuristic search algorithms (*Beam Search, DLSS*)*. As stated before, [Yamada 92] and [Holsalpe 93] identify a degree of uncertainty in these algorithms: (e.g. breaking ties between equally valued heuristics, selection of the next operation in heuristic procedures, generation of branches in tree search). It is possible to define different alternatives to handle these situations and which could be coded as part of the chromosome. Consequently, the partial schedules expressed by individuals evolve, but so does the knowledge about *which* method to be applied as a schedule builder/joiner and *how* it is to be applied.

2.2 Application to real industrial systems.

At almost any conference on the field, the existing gap between practice and theory in the application of *OR* results in industry becomes evident. A challenging research direction will be the development of a *PN* based simulation and control tool in cooperation with industrial partners. The *PN* based techniques developed here will be integrated in such product as an attempt to compete with current and popular available commercial tools such as *WITNESS* in *UK* and *ARENA* in *USA* offering two main advantages: first, that it is a *PN* based tool, which is a well known methodology for practitioners and second that offers powerful scheduling methodologies confronted with the heuristic dispatching rules that are normally employed.

The first steps to achieve this goal and a more affordable lines of research are the following:

2.2.1 Increase the complexity of the *FMS* formulation.

Cb-NETS have allowed to incorporate constraints of the *FMS* such as buffer policies and material stability constraints which are typically difficult to model with other representation paradigms such as mathematical programming. However, it will be of interest to adapt the algorithms developed in this work to more powerful *PN* extensions. *Cb-NETS* can be further extended to obtain a systematic *Hybrid High Level Petri nets* that can model the detailed manufacturing characteristics not considered here, such as part assembly/disassembly, dependant set-up times, tool changeover, pallet loading, AGV routing and path planning, and more complex models of machine operations. Although the work reviewed that integrates *PN* and heuristic systematic search have not considered these characteristics, there exists a large number of works focusing in the simulation and analysis part of production processes that have proposed powerful *PN* extensions to often model real systems [Camurri 93] [Yan 98] [Peng 98]. However, this work pays little attention to complex search techniques based on *PN*. We believe that the adaptation of *CGS* and *DLSS** would not suppose a major challenge and we believe that the performance of the systems will be increased.

2.2.2 Integration with results in *PN* applied to manufacturing control.

PN applied to manufacturing processes is a vast area of research. There are different sub-areas that concentrate in different problems and at different level of abstraction. Examples of these are human-system interfaces, user supervisory control, deadlock detention, control of forbidden states, transformation to *PN* structures to ladder logic diagrams (LLD) for Programable logic controllers *(PLC's)*. Interesting ideas and approaches needs to be integrated within the overall framework.

2.2.3. Adaptation of *DLSS** to on-line scheduling scenarios.

The natural evolution of the *PN* based scheduling methodology presented in this thesis is its extension to on-line and real-time scheduling scenarios. A preliminary extension is straightforward and is currently under development. The scheduling architecture changes slightly from the one depicted in *fig. 4* chapter 3. In fact, two new modules need to be added, a stochastic *PN* simulator [Murata 89] and a control module. The on-line architecture is shown in *fig. 75*.

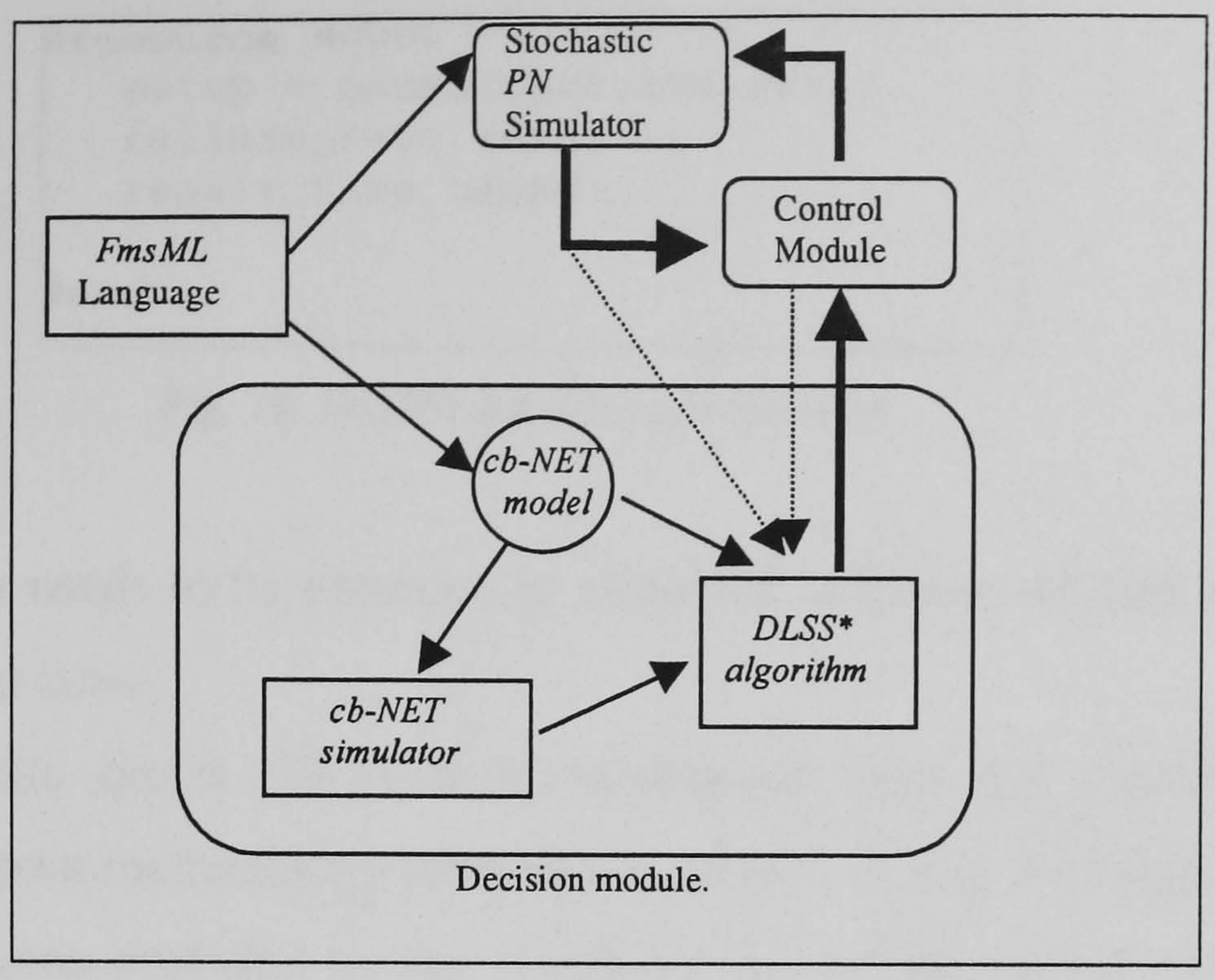


Fig.75: PN based scheduling architecture.

The user models the *FMS* description using *FmsML* which is parsed into two *PN* models. A first model is an extension of a stochastic *PN* which aims to model the behaviour of the real system. The second model is the deterministic *cb-NET* model. *DLSS** is the scheduling algorithm of the decision module. It receives the current state of the system from the *Stochastic PN simulator* and produces a complete or partial schedule in the form of the sequence of transitions to be fired. This sequence is applied by the control module to the *Stochastic PN simulator* and the performance obtained is monitored with the aim to meet unforeesen changes. If deviations from the expected behaviour are observed, the current state is passed to the decision module as the initial marking and a new schedule is obtained.

In order to achieve these objectives, the following changes or extensions need to be made to each of the following modules:

***FmsML* language:**

The *FmsML* needs to be extended to allow the possibility of defining general time distributions such as normal and exponential. Normal distributions will usually be employed to define operation times and set-up times, while exponential distributions allow the simulation of machine/resource failures. *Fig. 76* shows an example of a resource definition.

```
#resource Robot
    setup = gauss(mean,std_dev);
    failure_rate =exp(…);
    repair_time gauss(…)

#end;
```

Fig. 76: FmsML definition of a resource.

FmsML also needs to be extended to allow the definition of part demand rates and part cancellation rates.

The stochastic timed *PN* is then constructed from the *FmsML* definition following the top-down methodology presented in chapter 3. It is in charge of firing the sequence of transitions provided by the scheduler, but obtaining the operation times from the time distribution expressed. Also, it fires those transitions that will model new arrival or cancellation of parts, as well as unavailability of resources.

Decision module (*DLSS scheduling algorithm):**

Initially, the *PN-based* scheduling algorithms do not need modification. They still work over a deterministic *PN* model and obtain a sub-optimal schedule that achieves the production objectives.

However, instead of generating a full schedule, *DLSS** may be tuned to concentrate effort in local improvement. This is a common procedure in *look-ahead* methods that are related to *AI* search algorithms. The application of *AI* game search algorithms is an interesting alternative that frequently appears in scheduling methodologies based on *AI* paradigms (Expert Systems and knowledge based approaches) for on-line and real time scheduling. The application of the decision constrained by a limited search horizon (response time) or by the non-deterministic nature of a real system (for example, machine failure, product demand reorder, and the stochastic nature of the processes) that cause deviations from expected behaviour.

The parallel between games and real-time scheduling problems has been identified in the literature [Korf 90] [Dario 96]. In essence, a real-time scheduling problem can be seen as *computer-real world* game search. The computer devises a short-term strategy based on the expected behaviour of the system (model representation and deterministic simulation). The decisions *(moves)* are applied to the real system, which is obviously ruled by uncertainties both in processing times and unexpected events. Deviations from the expected behaviour represent the real system

move and the decision module *plays* now by reviewing the strategy. This continuous invocation of the decision module makes it inappropriate to devise a long strategy or to spend to much time worrying about obtaining a complete solution.

Even if deviations from the deterministic behaviour are not produced, we can identify the limited search horizon as a result of computational or information limitations [Korf 90] or, for example, avoid the generation of an unaffordable number of alternatives. Korf proposed several real-time heuristic search algorithms based on *A**. For example, time-limit *A** applies a pure *A** until a time limit is reached; threshold-limited *IDA** algorithm applies an iteration of the *IDA** [Korf 85] algorithm with a threshold which is always a constant amount greater than the heuristic estimate of the current state. At the end of each iteration, a move will be made in the direction of the most promising marking, each such move is an irrevocable decision and starts a new *A** or *IDA** iteration.

It is worth noting that the application of *DLSS** to an on-line scenario is straightforward. This is cannot be said for a *B&B* approach, for example. As stated in [Korg 90], a related drawback is that they must search all the way to a solution before making a commitment to how good a node (path) is. Actually, *B&B* finds solutions quickly basing the optimisation procedure on chronological backtracking. In other words, before a possible alternate choice at depth k is considered, the complete search sub-tree from the current state of depth k to the goal marking needs to be explored. On the contrary, *DLSS** can, for example, increase the *search frame* size and be interrupted when the time limit expires, returning the best partial schedule generated so far. We have implemented a preliminary version of *DLSS** that stops when a certain number of operations have been scheduled and integrated with a *stochastic PN* simulation that only considers non-determinism in operation time, and we are obtaining interesting results.

The implementation, testing and validation of the previous proposed *PN* based *on-line* scheduling methodology, will be of great interest to study a real situation. The scheduling algorithms developed in this thesis will be tested against traditional scheduling methods based on dispatching rules and simulation optimisation over *PN* models. Advances in this research direction aim to provide effective decision modules to be integrated in knowledge based architectures that employ *PN* as a representation paradigm [Martinez 89].

References.

References.

[Abdallah 98b] I. B. Abdallah and H. A. ElMaraghy: *Deadlock Prevention and Avoidance in FMS: A Petri Net Based Approach.* International Journal of Advanced Manufacturing Technology, 14 (704-715) 1998.

[Abdallah 98] I. B. Abdallah, H. A. ElMaraghy and ElMekkawy: *An Efficient Search Algorithm for Deadlock-free Scheduling in FMS using Petri Nets*. Proc. IEEE Int. Conf. On Robotics and automation. Leuven, Belgium, (1793-1798) 1998

[Adamou 93] M. Adamou, A. Borjault, S. N. Zerhouni, *Modelling and control of flexible manufacturing assembly systems using object oriented Petri Nets*. IEEE International Workshop on Emerging Technol and Factory Automat. Cairns, Australia, (164-168) 1993.

[Adams 88] J. Adams, E. Balas and D. Zawack. *The Shifting Bottleneck Procedure for Job Shop Scheduling*. Management Science. 34 (391- 401) 1998.

[Aemsa] AEM,S.A. Accesorios Eléctricos y Mecánicos S.A. Valencia, Spain. *www.aemsa.es.*

[Ahn 93] J. Ahn, W. He, and A. Kusiak. *Scheduling with Alternative Operations*. IEEE Transactions on Robotics and Automation. 9 (297-303) 1993.

[Ammons 88] J. C. Ammons, T. Govindaraj and C. M. Mitchell. *Decision models for Aiding FMS scheduling and Control. IEEE Transactions on Systems Man and Cybernetics*. 18 (744-756) 1998.

[Arjona 96] E. Arjona-Suarez and E. Lopez-Mellado. *Synthesis of coloured Petri Nets for FMS task specification*. International Journal of Robotics and Automation, 11 (3) 1996.

[Ashour 67] S. Ashour. *A Decomposition approach for the machine scheduling problem.* The International Journal of Production Research, 6 (109-122) 1967.

[Aytug 98] H. Aytug, S. Bhattacharyya and G. J. Koehler. *Genetic learning through simulation: an investigation in shop floor scheduling*. Annals of Operations Research. 78 (1-29) 1998.

[Azzopardi 94] D. Azzopardi and S. Lloyd. *Scheduling and simulation of Multi-product Batch Process Plant through Petri Net modeling.* IEE Conference on Advanced Factory Automation. Oct. 1994.

[Bagchi 91] S. Bagchi, S. Uckun, Y. Miyabe and K. Kawamura. *Exploring Problem-Specific Recombination Operators for Job-Shop Scheduling*. Proc. 4th Int. Conf. On Gen. Algorithms. San Diego, CA, (10-17) 1991.

[Banaszak 90] Z. A. Banaszak & B. H. Krogh. *Deadlock Avoidance in Flexible Manufacturing Systems with Concurrently Competing Process Flows*. IEEE Transactions on Robotics and Automation. 6 (724-734) 1990.

[Basnet 94] C. Basnet and J. H. Mize. *Scheduling and control of flexible manufacturing systems: a critical review*. International Journal of Computer Integrated Manufacturing. 7 (340-355) 1994.

[Bean 94] J. C. Bean. *Genetic Algorithms and Random Keys for Sequencing and Optimization.* ORSA Journal of Computing. 6 (154 –160) 1994.

[Benjaafar 92] S. Benjaafar. *Intelligent simulation for flexible manufacturing systems: An integrated approach*. Computers in Industrial Engineering. 22 (297-311) 1992.

[Bensana 88] E. Bensana. G. Bel and D. Dubois. OPAL: *A multi-knowledge based system for industrial job-shop scheduling*. International Journal of Production research. 26 (795 – 819) 1988.

[Berrada 86] M. Berrada and K. E. Stecke. *A branch and bound approach for machine load balancing in Flexible Manufacturing Systems*. Management Science. 32 (1316-1335) 1986.

[Biegel 90] J. E. Biegel and J. J. Davern. *Genetic algorithms and Job Shop Scheduling.* International Journal of Computers and Industrial Engineering. 19 (81- 91) 1990.

[Bispo 92] C.F.G. Bispo, J.J.S Sentiero and R. D. Hibberd. *Adaptive Scheduling for High-Volume Shops*. IEEE Transactions on Robotics and Automation. 8 (696-706) 1992.

[Bistline 98] W.G. Bistline, S. Banerjee and A. Banerjee. *RTSS: An interactive decision support system for solving real time scheduling problems considering customer and job priorities with schedule interruptions*. Computers Ops. Research. 25 (237-247) 1998.

[Bona 90] B. Bona, P. Brandimarte, C. Greco and G. Menga. *Hybrid Hierarchical Scheduling and Control Systems in Manufacturing.* IEEE Transactions on Robotics and Automation. 6 (673-686) 1990.

[Boutet 98] F. Boutet and G. Motet. *Use of Constraints in Petri Nets for Modelling and Solving Scheduling Problems in Preliminary System Design*. Proceedings IEEE International Conference on Systems Man and Cybernetics, San Diego, California, (576-581) 1998.

[Brucker 94] P. Brucker, B. Jurisch and B. Sievers. *A branch and bound algorithm for the job-shop scheduling problem*. Discrete Applied Mathematics. 49 (107-127) 1994.

[Bruno 86] G. Bruno, A. Elia and P. Laface. *A rule-based system to schedule production.* IEEE Computer. Vol 19 (32-40) 1986

[Bu 88] M. I. Bu-Hulaiga and A. K. Chakravarty. *An object-oriented knowledge representation for hierarchical real-time control of flexible manufacturing.* International Journal of Production Research. Vol. 26 (777-793) 1988.

[Byeon 98] E.S. Byeon, S. David Wu and R.H. Storer. *Decomposition Heuristics for Robust Job-Shop Scheduling.* IEEE Transactions on Robotics and Automation. 14 (303-313) 1998.

[Calvez 98] S. Calvez, P. Aygalinc and P. Bonhomme. *Transient Functioning Mode of Manufacturing Systems with Operation Time Constraints*. The International Journal of Advanced Manufacturing Technology. 14 (694 -703) 1998.

[Camurri 91] A. Camurri, P. Franchi and F. Gandolfo. *A Timed Colored Petri Nets Approach To Process Scheduling*. IEEE 1991 (304 – 309)

[Camurri 93] A. Camurri, P. Franchi, F. Gandolfo and R. Zaccaria: *Petri Net Based Process Scheduling: A model of the Control System of Flexible Manufacturing System*. Journal of Intelligent and Robotic Systems. 8 (99-123) 1993.

[Caramihai 98] S. I. Caramihai, I. Dumitrache and A. M. Stanescu. *Real-time Supervision for Intelligent Manufacturing Supported by T-Temporal Petri Net Models*. Proceedings IEEE International Conference on Systems Man and Cybernetics, San Diego, California, (582-587) 1998.

[Carlier 89] J. Carlier and E. Pinson. *An Algorithm for solving the Job-Shop problem*. Management Science. 35 (164 -165) 1989.

[Chakravarty 97] A.K. Chakravarty, H.K. Jain, J.J. Liu and D. L. Nazareth. *Object-oriented Domain Analysis for Flexible Manufacturing Systems*. Integrated Computer Aided Engineering. 4 (290-309) 1997.

[Chan 97] F.T.S. Chan, A. Kazerooni and K. Abhary. *A Fuzzy approach to Operation Selection.* Engineering Applications of Artificial Intelligence. 10 (345-356) 1997.

[Chan 90] D-Y Chan and D. D. BedWorth. *Design of a scheduling system for flexible manufacturing cells*. International Journal of Production Research, 28 (2037-2049) 1990.

[Chang 97] J.W. Chang and Y-P- Luh. *Integration of scheduling and control in a Job Sho*p. Journal of the Chinese Institute of Engineers. 20 (67-76) 1997.

[Chang 89] Y-L Chang, H. Matsuo, R. Sullivan. *A bottleneck-based beam search for job scheduling in a flexible manufacturing system*. International Journal of Production Research. 27 (1949-1961) 1989.

[Chen 98] H. Chen, C. Chu and J.M Proth. *An Improvement of the Lagrangean Relaxation Approach for Job Shop Scheduling: A Dynamic Programming Method.* IEEE Transactions on Robotics and Automation. 14 (786-795) 1998

[Chen 96] M Chen. *A Mathematical Programming Model for AGVS planning and control in Manufacturing Systems*. Computers industrial Engineering. 30 (647-658) 1996.

[Chen 93] Q. Chen and J.Y.S. Luh: *Task level optimum scheduling by truncated Petri nets applied to operation of multi-robot work*cell. IEEE 1993 (326-331)

[Chen 95] S. C. Chen and M. D. Jeng. FMS scheduling Using Backtracking-Free Heuristic Search Based on Petri Net State Equations. Proc. Int. Conf. On Systems, Man and Cybernetics. (2153 – 2158) 1995.

[Chen 94] Q. Chen, J.Y.S Luh, L. Shen. *Complexity Reduction for optimization of deterministic Timed Petri-Net Scheduling by Truncation*. Cybernetic and Systems: An international Journal. 25 (643-695) 1994.

[R. Chen 96] R. Chen, M. Gen and Y. Tsujimura. *A tutorial survey of job-shop scheduling problems using Genetic Algorithms – I Representation*. Computers Industrial Engineering 30 (983 – 997) 1996.

[C-L. Cheng 96] C-L. Cheng, R. V. Neppali and N. Aljaber. *Genetic Algorithms applied to the continous flow shop problem*. Computers Ind. Engnineering. 30 (919- 929) 1996.

[Cheng 96] C.C. Cheng and S. F. Smith. *A constraint Satisfaction Approach to Makespan Scheduling. AI*PS-96, (45-52) 1996.

[Cheng 98] R. Cheng and M. Gen. *An evolution programme for the resource-constrained project scheduling p*roblem. International Journal of Computer Integrated Manufacturing, 1998, 11 (274-287) 1998.

[Chetty 96] O.V.K. Chetty and O. C. Gnanasekaran. *Modelling, Simulation and Scheduling of Flexible Assembly Systems with Coloured Petri Nets*. International Journal of Advanced Manufacturing Technology, 11 (430-438) 1996.

[Chincholkar 96] A. K. Chincholkar and O.V. Krisnaiah Chetty. *Stochastic Coloured Petri Nets for modelling and Evaluation, and Heuristic Rule Base for Scheduling of FMS*. International Journal of Advanced Manufacturing Technology, Vol. 12 (339-348) 1996.

[Chiu 97] Y.F. Chiu and L.C. Fu. *A GA Embedded Dynamic Search Algorithm over a Petri Net Model for an FMS Scheduling*. Proceedings of the 1997 IEEE International Conference on Robotics and Automation. (513-518) 1997.

[Christofides 87] N. Christofides, R. Alvarez-Valdes and J.M. Tamarit. *Project scheduling with resource constraints: A branch and bound appraoch*. European Journal of Operational Research. 37 (262-273) 1987.

[Chryssolouris 85] G. Chryssolouris, and S. Chan. *An Integrated Approach to Process Planning and Scheduling*. Annals of the CIRP, 34 (413-416) 1985.

[Chryssolouris 88] G. Chryssolouris, K. Wright, J. Pierce and W. Cobb. *Manufacturing systems operation: Dispatch rules versus intelligent control. Robotics & Computer-Integrated Manufacturing*, 4 (531- 544) 1988.

[Chryssolouris 94] G. Chryssolouris, K. Dicke, and M. Lee. *An Approach to Real-Time Flexible Scheduling*. The international Journal of Flexible Manufacturing Systems, 6 (235-253) 1994.

[Chryssolouris 91] G. Chryssolouris, K. Dicke and M. Lee. *An approach to short interval scheduling for discrete parts manufacturing*. International Journal of Computer Integrated Manufacturing. 4 (157 – 168) 1991.

[Chu 98] C. Chu, J.M. Proth and C. Wang. *Improving job-shop schedules through critical pairwise exchanges*. International Journal of Production Research. 36 (683-694) 1998.

[Chu 92] C. Chu, M. C. Portmann and J. M. Proth. *A splitting-up approach to simplyfy job-shop scheduling problems*. International Journal of Production Research. 30 (859 – 870) 1992.

[Y-Y. Chu 98] Y-Y. Chu, L-C. Fu and M-W Lin. *Petri net based modeling and GA Based scheduling for a Flexible manufacturing system*. Proceedings of the 37th IEEE Conference on Decision & Control. (4346-4347) 1998.

[Co 88] H.C. Co, T. J. Jaw and S. K. Chen. *Sequencing in Flexible manufacturing Systems and other Short Queue-lengh System*. Journal of Manufacturing Systems. 7 (1 – 7) 1988.

[Colombo 97] A. W. Colombo, R. Carelli and B. Kuchen: *A Temporised PN approach for Design, Modelling and Analysis of Flexible Production Systems*. International Journal of Advanced Manufacturing Technology. 13 (214-226) 1997.

[Cossins 92] R. Cossins, P. Ferreira. *Celeritas: a coloured Petri net approach to simulation and control of flexible manufacturing systems*. International Journal of Production Research, 30 (1925-1956) 1992.

[Coves 98] C. Coves, D. Crestani and F. Prunet. *How to manage coverability graphs construction: and overview*. Proceedings IEEE International Conference on Systems Man and Cybernetics, San Diego, California, 1998 (541- 546).

[Damasceno 98] B.C. Damasceno, X.Xie. *Scheduling and Deadlock Avoidance of a Flexible Manufacturing System*. Proc. Of the 28th Int. Conf. On Systems, Man and Cybernetics. (564-569) 1998.

[Dario 96] D. R Dario, J. Toncich Chess: *A Methodology for Resolving Scheduling and Dispatch Problems in FMSs*. The International Journal of Flexible Manufacturing Systems, 8 (23-43) 1996.

[Das 97] S. K. Das, P. Nagendra. *Selection of routes in a flexible manufacturing facility.* International Journal of Production Economics. 48 (237-247) 1997.

[De 88] S. De. *A Knowledge-Based Approach to Scheduling in an F.M.S.* Annals of Operations Research. 12 (109-134) 1988.

[De 90] S. De and A. Lee. *Flexible Manufacturing system (FMS) scheduling using filtered beam search.* Journal of Intelligent Manufacturing . 1 (165-183) 1990.

[Della Croce 95] F. Della Croce, R. Tadei and G. Volta. *A genetic algorithm for the Job Shop Problem.* Computers and Operation Research. 22 (15-24) 1995.

[Demeulemeester 92] E. Demeulemeester and W. Herroelen. *A branch and bound procedure for the multiple resource-constrained project scheduling problem.* Management Science. 38 (1803-1818) 1992.

[Dicesare 91] F. Dicesare and A. A. Desrochers, *Modelling control and Performance analysis of automated manufacturing systems using Petri Nets*. Control and Dynamics Systems. Vol. 47 (121- 172) 1991.

[Donath 88] M. Donath and R. J. Graves. *Flexible assembly systems: an approach for real-time scheduling and routeing of multiple products*. International Journal of Production Research . 26 (1903-1919) 1988.

[Dorndorf 95] U. Dorndorf and E. Pesch. *Evolution based learning in a Job Shop Scheduling Environment.* Computers and Operation Research. 22 (25-40) 1995.

[Doulgeri 87] Z. Doulgeri, R. D. Hibberd, T. M. Husband. *The scheduling of Flexible Manufacturing Systems*. Annals of the CIRP, 36 (343 – 346) 1987.

[D'Souza 93] K. A. D'Souza and S. K. Khator. *A Petri Net approach for modelling controls of a computer-integrated assembly ce*ll. International Journal of Computer Integrated Manufacturing. 6 (302 – 310) 1993.

[Dutilleul 98] S.C. Dutilleul, J-P. Denat. *P-time Petri Nets and the Hoist Scheduling Problem*. Proc. of the 28th Int. Conf. On Systems, Man and Cybernetics. 1 (558-563) 1998.

[ElMekkawy 98] T. ElMekkawy, I. Ben Abdallah, and H. ElMaraghy. *A heuristic Search Approach for Deadlock-Free Scheduling in FMS's Using petri Nets and AI Techniques.* Proc. of 1998 ASME Computers in Engineering Conference, Atlanta, GA, USA, September 1998

[Falkenauer 91] E. Falkenauer and S. Bouffouix. *A Genetic Algorithm for Job Shop*. Proceedings of the 1991 IEEE International Conference on Robotics and Automation. Sacramento, California. (824 – 829) 1991.

[Feldmann 98] K. Feldmann and A. W. Colombo. *Material Flow and Control Sequence Specification of Flexible Production Systems Using Coloured Petri Nets*. International Journal of Advanced Manufacturing Technology. 1998, 14 (760 – 774) 1998.

[Ferrarini 98] L. Ferrarini and M. Maroni. *Deadlock Avoidance Control for Manufacturing Systems with multiple Capacity Resources*. International Journal of Advanced Manufacturing Technology. Vol. 14 (729 – 736) 1998.

[Fikes 71] R. E. Fikes and N. J. Nilsson, *STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving*. Artificial Intelligence, 2 (189-208) 1971.

[Fox 82] M. S. Fox, B. Allen, and G. Strohm. *Job-Shop scheduling: an investigation in constraint-directed reasoning*. Proceedings of the National Conference on Artificial Intelligence. (155-158) 1982

[Fox 84] M. S. Fox and S. F. Smith. *ISIS – a knowledge based system for factory scheduling*. Expert Systems. 1 (25- 49) 1984.

[Fujimoto 96] H. Fujimoto, K. Yasuda, Y. Tanigawa and K. Iwahashi. *Applications of Genetic Algorithm and Simulation to Dispatching Rule-Based FMS Scheduling*. Computers Industrial Engineering. 30 (969-981) 1996.

[Genrich 81] H.J. Genrich and K. Lautenbach. *System modeling with high-level Petri Nets.* Theoretical Computer Science. 13 (109-136) 1981.

[Glass 96] C.A. Glass, C. N. Poots. *A comparison of local search methods for flow shop scheduling.* Annals of Operations Research. 63 (489-509) 1996.

[Godberg 89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* 1989 Addison-Wesley Publishing.

[Gupta 89] Y.P. Gupta, M.C. Gupta and C.R. Bector. *A review of scheduling rules in flexible manufacturing systems.* International Journal of Computer Integrated Manufacturing, 2 (357-377) 1989.

[Gusikhin 93] O. Y. Gusikhin, and A. S. Kulinitch. *Animated AI-Based simulation in production scheduling.* IFIP Transactions. B-Applications in technology, 11 (165-176) 1993.

[Hariri 97] A. M. A. Hariri and C.N. Potts. *A branch and bound algorithm for the two-stage assembly scheduling problem.* European Journal of Operational Research, 103 (547-556) 1997.

[Harmonosky 91] C. M Harmonosky and S. F. Robohn. *Real-Time scheduling in computer integrated manufacturing: a review of recent research.* International Journal of Computer Integrated Manufacturing. 4 (331-340) 1991.

[Hatono 91] I. Hatono, K. Yamagata and H. Tamura. *Modeling and On-Line Scheduling of Flexible Manufacturing Systems Using Stochastic Petri Nets.* IEEE Transactions on Software Engineering, 17 (126-132) 1991.

[Heiner 99] M. Heiner, P. Deussen and J. Spranger. *A Case Study in Design and Verification of Manufacturing system Control software with hierarchical Petri Nets.* International Journal of Advanced Manufacturing Technology. 15 (139 – 152) 1999.

[Herrman 95] J. W. Herrmann, C-Y. Lee. *Solving a Class Scheduling Problem with a Genetic Algorithm.* ORSA Journal of Computing. 7 (443 – 452) 1995.

[Herrman 95b] J. W. Herrmann, C-Y. Lee and J. Hinchman. *Global Job Shop Scheduling with a Genetic Algorithm.* Production and Operation Management. 4 (30 – 45) 1995

[Hillion 98] H.P. Hillion and J.M. Proth. *Using Timed Petri Nets for the scheduling of Job-Scheduling of Job-Shop Systems.* Engineering Costs and Production Economics. 17 (149-154) 1998.

[Hillion 87] H.P. Hillion, J.M. Proth and X.L. Xie. *A Heuristic Algorithm for the Periodic Scheduling and Sequencing Job-Shop Problem.* Proc. Of the 28th Conference on Decision and Control. (612-617) 1987

[Hillion 89] H.P. Hillion and J-M Proth. *Performance evaluation of Job-Shop Systems Using Timed Event-Graphs.* IEEE transactions on Automatic Control, 34 (3 – 9) 1989.

[Holland 81] J.H. Holland. *Genetics Algorithms and adaptation.* Technical Report No. 34. Ann Arbor: University of Michigan, department of Computer and Communication Sciences.

[Holsapple 93] C. W. Holsapple, V. S. Jacob, R. Pakath and J. S. Zaveri. *A Genetics-Based Hybrid Scheduler for Generating Static Schedules in Flexible Manufacturing Co*ntexts. IIIE Transactions on Systems, Man and Cybernetics. 23 (953-972) 1993.

[Hsieh 94] F-S. Hsieh and S-C. Chang. *Dispatching-Driven Deadlock Avoidance Controller Synthesis for Flexible Manufacturing Systems.* IEEE Transactions on Robotics and Automation. 10 (196-209) 1994.

[Hsu 96] C-C. Hsu, S-I. Yamada, H. Fujikawa and K. Shida. *A Fuzzy Self-Tuning Parallel Genetic Algorithm for optimisation.* Computers Industrial Engineering. 30 (883- 893) 1996.

[Hu 95] G. H. Hu, Y. S. Wong and H. T. Loh. *An FMS scheduling and Control decision Support System based on Generalised Stochastic Petri Nets.* International Journal of Advanced Manufacturing Technology, 10 (52-58) 1995.

[Hunag 92] H.P. Huang and P.C. Chang. *Specification, modelling and control of a flexible manufacturing cell.* International Journal of Production Research, 30 (264-277) 1992.

[Hutchison 91] J. Hutchison, K. Leong, D. Snyder and P. Ward. *Scheduling approaches for random job shop flexible manufacturing systems.* International Journal of Production Research, 29 (30-41) 1991.

[Hwan 89] S. S. Hwan and A. W. Shogan. *Modelling and solving and FMS part selection problem.* International Journal of Production Research. 27 (1349 – 1366) 1989.

[Inaba 98] A. Inaba, F. Fujiwara, T. Suzuki, S. Okuma: *Timed Petri Net based scheduling for Mechanical Assembly - Integration of Planning and scheduling.* IEICE Transactions on Fundamentals of Electronics Communication and Computer Sciences. Vol. E-81A (615-625) 1998.

[Ishii 96] N. Ishii and M. Muraki. *An extended dispatching rule approach in an on-line scheduling framework for batch process management.* International Journal of Production Research. 34 (329 – 348) 1996.

[Ishii 91] N. Ishii and J. J. Talavage. *A transient based real-time scheduling algorithm in FMS.* International Journal of Production Research. 29 (2501-2520) 1991.

[Iwata 80] K. Iwata, Y. Murotsu and F. Oba. *Solution of Large-Scale Scheduling Problems for Job-Shop Type Machining Systems with alternative Machine Tools.* Annals of the CIRP, 29 (335-338) 1980.

[Jain 97] A. K. Jain and H. A. ElMaraghy. *Production scheduling/rescheduling in flexible manufacturing systems. International journal of production research.* 35 (281-309) 1997.

[Jawahar 98] N. Jawahar, P. Aravindan and S. G. Ponnambalam. *A Genetic Algorithm for Scheduling Flexible Manufacturing Systems.* International Journal of Advanced Manufacturing Technology, 14 (588-607) 1998.

[Jeng 93b] M-D. Jeng, J.T. Lin and U-P. Wen. *Algorithms for sequencing Robot Activities in a Robot-Centered Parallel-Processor Workcell.* Computers and Operations Research. 20 (185- 197) 1993.

[Jeng 98b] M.D. Jeng, W.D. Chiou and Y-L Wen. *Deadlock-Free Scheduling of Flexible Manufacturing Systems Based on Heuristic Search and Petri Net Structures.* Proc. Of the 28th Int. Conf. On Systems, Man and Cybernetics. (26 – 31) 1998

[Jeng 98] M.D. Jeng and S.C. Chen. *A Heuristic Search Approach Using Approximate Solutions to Petri Net State Equations for Scheduling Flexible Manufacturing Systems.* International Journal of Flexible Manufacturing Systems, 10 (139-162) 1998.

[Jeng 99] M. D Jeng and S. C. Chen. *Heuristic Search Based on Petri Net Structures for FMS scheduling.* IEEE Transactions on Industry Applications. 35 (196-202) 1999.

[Jeng 96] M. D. Jeng, S. C. Chen and C. S. Lin. *A search approach based on the Petri Net Theory for FMS Scheduling.* 1996 IFAC 13th Triennial World Congress. San Francisco (55-60) 1996.

[Jeng 93] M-D. Jeng and F. Dicesare. *A Review of Synthesis Techniques for Petri Nets with Applications to Automated Manufacturing Systems.* IEEE Transactions on Systems, Man and Cybernetics. 23 (301-312) 1993.

[Jianjun 92] Y. Jianjun, Z. Feng, D. Jiati and C. Chuaijun: *Intelligent Manufacturing Cell Controller IMCC-E* . Human aspects in Computer Integrated Manufacturing. 1992 IFIP, Elsevier Science Publishers. pp. 745 - 755

[Jones 95] A. Jones, L. Rabelo and Y. Yih. *A hybrid approach for real-time sequencing and scheduling.* International Journal of Computer Integrated Manufacturing. Vol. 8 (145 – 154) 1995

[Jordan 98] C. Jordan. *A two-phase genetic algorithm to solve variants of the batch sequencing problem.* International Journal of Production Research. 36 (745-760) 1998

[Kanet 91] J. J. Kanet and V. Sridharan. Progenitor: *A genetic algorithm for production scheduling.* Wirtschaftsinformatik 1991 (332-336)

[Khansa 96] W. Khansa, J-P Denat , S. Dutilleul. *P-time Petri Nets for manufacturing systems.* Procedings of IEE WODES'96, Edinburg, Scotland, August 1996, (94-102).

[Karabuk 93] S. Karabuk and I. Sabuncouglu. *A beam search based algorithm for schedules machines and AGVs in a FMS.* Proc. 2nd Industrial Engineering Research Conference. Los Angeles. CA. 1993. (308-312).

[Kazerooni 89] A. Kazerooni, F.T.S Chan and K. Abhary. *A fuzzy integrated decision-making support system for scheduling of FMS using simulation.* Computer Integrated Manufacturing Systems. 10 (27 – 34) 1997

[Khoshnevis 89] B. Khoshnevis Q. Chen. *Integration of Process Planning and Scheduling functions.* Proceedings IIE Integrated Systems Conference & Society for Integrated Manufacturing. (415 – 420) 1989.

[M.H. Kim 94] M. H. Kim and Y.D. Kim, *Simulation based real-time scheduling in FMS.* Journal of manufacturing Systems. 13 (85-93) 1994.

[Y-D. Kim 94] Y-D Kim and C. A. Yano. *A due date-based approach to part type selection in flexible manufacturing systems.* International Journal of Production Research, 32 (1027-1043) 1994.

[Y-D. Kim 90] Y.-D. Kim. *A comparison of dispatching rules for job shops with multiple identical jobs and alternative routings,* International Journal of production Research, *28(5), 953-962.1990*

[Kim 99] K-H Kim and P.J. Egbelu. *Scheduling in a production environment with multiple process plans per job.* International Journal of Production Research. 37 (2725-2753) 1999.

[Kimemia 85] J. Kimemia and S. B. Gershwin. *Flow optimisation in flexible manufacturing systems.* International Journal of Production Research. 23 (81 – 96) 1985.

[Kimemia 83] J. G. Kimemia, S. B. Gershwin. *An algorithm for the computer control of production in a Flexible Manufacturing System.* IIE Transactions., 15 (353-3629) 1983.

[Kobayashi 95] S. Kobayashi, I. Ono and M. Yamamura. *An Efficient Genetic Algorithm for Job Shop Scheduling Problems.* Proc. 6th International Conference on Genetic Algorithms (506-511) 1995

[Kochikar 93] V.P Kochikar and T.T. Narendran: *State space approach to qualitative analysis of FMSs.* Computer integrated Manufacturing Systems. 6 (9-17) 1993.

[Konstans 98] N. Konstans, S. Lloyd, H. Yu. *Rule based Petri Net Modelling and scheduling of an FMS.* Proc. of the 14th NCMR Conference 1998.

[Kopfer 97] H. Kopfer and D. C. Mattfeld. *A Hybrid Search Algorithm for the Job Shop Problem. .* Proceedings of the first International Conference on Operations and Quantitative management. 2 (498-505) 1997.

[Korf 85] R. E. Korf . *Depth-First Iterative-Deepening: An Optimal Admissible Tree Search.* Artificial Intelligence. 27 (97-109) 1985.

[Korf 90] R. E. Korf. *Real - Time heuristic search..* Artificial Intelligence. Vol. 42 (189-211) 1990.

[Kruger 98] K. Kruger, Y. N. Sotskov and F. *Werner Heuristics for generalized shop scheduling problems based on decomposition.* International Journal of Production Research. 36 (3013-3033) 1998.

[Kruger 95] K. Kruger, N. V. Shakhlevich, Y. N. Sotskov and F. Werner. *A Heuristic Decomposition Algorithm for Scheduling Problems on Mixed Graphs.* Journal of the Operational Research Society. 46 (1481- 1497) 1995.

[Kubek 95] J.M. Kubek and G. Motet. *Application of Temporal Constrained Predicate Nets to Production Systems.* Symposium on Emerging Technologies and Factory Automation. INRIA/IEEE. (69-80) 1995.

[Kuo 98] C-H. Kuo, H-P. Huang and M-C. Yeh. *Object-Oriented Approach of MCTPN for Modelling Flexible Manufacturing Systems.* International Journal of Advanced Manufacturing Technology. 14 (737 – 749) 1998.

[Kusiak 89] A. Kusiak. *KBSS: A knowledge and optimisation based system for manufacturing scheduling.* Proceedings of the International Industrial Enginnering Conference & Societies' Manufacturing and Productivity Symposium. (694 – 699) 1989

[Kusiak 89b] A. Kusiak, and S. S. Heragu. *Expert systems and Optimization.* IEEE Transactions on Software Engineering. 15 (1017-1020) 1989.

[Lee 95] C. Y Lee and J. Y. Choi. *A genetic algorithm for job sequencing problems with distinct due dates and general early-tardy penalty weights.* Computers and Operation Research. 22 (857-869) 1995.

[I. Lee 97] I. Lee, R. Sikora and M. J. Shaw. *A genetic algorithm based approach to Flexible flow-line scheduling with variable lot sizes. IEEE Transactions on systems, man and cybernetics. 27 (36-54) 1997.*

[C.Y. Lee 97] C.Y. Lee, L. Lei, and M. Pinedo. *Current trends in deterministicc scheduling*. Annals of Operations Research. 70 (1-41) 1997.

[Lee 98] S. Lee, J. Bok and S. Park. *A New algorithm for Large-Scale Scheduling Problems: Sequence Branch Algorithm*. Industrial & Engineering Chemistry Research, 37(4049-4058) 1998.

[Lee 92] D. Y. Lee and F. Dicesare. *FMS Scheduling Using Petri Nets and Heuristic Search*. Proceedings of the 1992 IEEE International Conference on Robotics and Automation, Nice, France, (1057-1062) 1992.

[Lee 92b] D-Y. Lee and F. Dicesare. *Petri Nets and Heuristic Search for Periodic Scheduling*. IEEE 1992 (998-1003).

[Lee 94] D. Y. Lee and F. Dicesare: *Scheduling Flexible Manufacturing Systems Using Petri Nets and Heuristic Search*. IEEE Transactions on Robotics and Automation. 10 (123-132) 1994.

[Lee 94b] D. Y. Lee and F. Dicesare. *Integrated Scheduling of Flexible Manufacturing Systems Employing Automated Guided Vehicles*. IEEE Transactions on Industrial Electronics, 41 (602- 609) 1994

[Lee 97] C-Y. Lee, S. Piramuthu and Y. K Tsai. *Job Shop Scheduling with a genetic algorithm and machine learning*. International Journal of Production Research. 35 (1171 – 1191) 1997.

[Li 97] D. C. Li, C. Wu and K. Y. Tomg. *Using an unsupervized neural network and decision tree as knowledge acquisition tools for FMS scheduling*. International Journal of System Science. 28 (977-985) 1997.

[Lin 97] J. T. Lin and C. C. Lee: *A Petri net-based integrated control and scheduling scheme for flexible manufacturing cells*. Computer integrated Manufacturing Systems. 10 (109-122) 1997.

[Lin 95] J. T. Lin and C. C. Lee: *A CTPN based scheduler for a flexible Manufacturing Cell*. Journal of the Chinese Institute of Engineers, 18 (655-672) 1995.

[Liu 89] P-S. Liu and L-C, Fu. *Planning and Scheduling in a Flexible Manufacturing System Using a Dynamic Routing Method for Automated Guided Vehicles*. Proc 1989 IEEE International Conference on Robotics and Automation, (1584-1589) 1989.

[Liu 92] P-S. Liu and L. C. Fu. *Hierarchical dynamic scheduling for FMS*. Proc. 3rd Int. Conf. On Computer Integrated Manufacturing. (393-402) 1992.

[Liu 93] C.M. Liu and F. C. Wu: *Using Petri nets to solve FMS problems*. International Journal of Computer Integrated Manufacturing. 6 (175-185) 1993.

[Liu 97] J. Liu. B.L. MacCarthy. *A global MILP model for FMS scheduling. European Journal of Operational Research*. 100 (441-453) 1997.

[Lloyd 95b] S. Lloyd, H. Yu and C.V. Balakrishnan: *Petri Net Modelling and Schedulling of Flow Shop Systems with Different Storage Policies*. Proc. The 8th IFAC Symposium on Information Control Problems in Manufacturing Systems, Beijing, China, 11-13 Oct. 1995.

[Lloyd 95] S. Lloyd, H. Yu. and N. Konstants. *FMS scheduling Using Petri Net Modeling and Brach & Bound Search*. Proc. IEEE International Symposium on Assembly and Task Planning, Pittsburgh, USA, (141-146) Aug. 1995.

[Lo 91] Z-P. Lo and B. Bavarian. *Job Scheduling on Parallel Machines Using Simulated Annealing*. Proceedings of the 1991 IEEE International Conference of Systems, Man and Cybernetics. (391- 396) 1991.

[Logendran 97] R. Logendran and A Sonthinen. *A Tabu search-based approach for scheduling job-shop type flexible manufacturing systems*. Journal of the operational Research Society, 48 (264-277) 1997.

[Logendran 94] R. Logendran, P. Ramakrishna and C. Sriskandarajah. *Tabu search-based heuristics for cellular manufacturing systems in the presence of alternative process plans*. International Journal of Production Research, 32 (273-297) 1994.

[Maccarthy 93] B. L. Maccarthy and J. Liu. *Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling*. International Journal of Production Research. 3 (59-79) 1993.

[Maley 88] J. G. Maley, S. Ruiz-Mier and J. J. Solberg. *Dynamic control in automated manufacturing: a knowledge integrated approach.* International Journal of Production research. 26 (1739 – 1748) 1988.

[Malo-Tamayo 98] A.J. Malo-Tamayo, D. Gaviño-Contreras, A. Ramirez-Treviño. *Petri Net Based Control for the Dynamic Scheduling of a Flexible Manufacturing Cell.* Proceedings IEEE International Conference on Systems Man and Cybernetics, San Diego, California, (553-557) 1998.

[Mamalis 96] A. G. Mamalis and I. Malagardis. *Determination of due dates in job-shop scheduling by simulated annealing.* Computer Integrated Manufacturing Systems. 9 (65 – 72) 1996.

[Martinez 89] J. Martinez, P. Muro, M. Silva, S.F. Smith and J.L. Villaroel. *Merging Artificial Intelligence Techniques and Petri Nets for real Time scheduling and Control of Production Systems.* Artificial Intelligence in Scientific Computation. R. Huber editor. (307-313) 1989.

[Martinez 87] J. Martinez, P. Muro and M. Silva. *Modeling, validation and software implementation of Production Systems using High Level Petri* Nets. Proc. 1987. IEEE International Conference on Robotics and Automation. (1180-1185) 1987.

[Mason 90] T. Mason and D. Brow. *Lex & yacc.* O'Reilly & Associates, Inc. 1990.

[Maturana 97] F. Maturana, P. Gu, A. Naumann and D. H. Norrie. *Object-oriented job-shop scheduling using Genetic algorithms.* Computers in industry. 32 (281 – 295) 1997.

[Murata 86] T. Murata, N. Komoda, K. Matsumoto. *A Petri-Net Based Controller for Flexible and Maintainable Sequence Control and its Applications in Factory Automation.* IEEE Transactions on Industrial Electronics. 33 (1-8) 1986.

[Murata 89] T. Murata: Petri Nets*: Properties, Analysis and Applications.* In proceeding of the IEEE, (541-580) 1989.

[Murata 96] T. Murata, H. Ishibuchi and H. Tanaka. *Genetic algorithms for flowshop scheduling problems.* Computers Industrial Engineering. 30 (1061- 1071) 1996.

[Murata 96b] T. Murata, H. Ishibuchi and H. Tanaka. *Multi-objective genetic algorithm and its applications to flowshop scheduling.* Computers Industrial Engineering. 30 (957- 968) 1996.

[Muth 63] Muth, J.F., and Thompson, G.L. *Industrial Scheduling.* Englewood Cliffs, NJ: Prentice Hall. 1963.

[Nakano 91] R. Nakano and T. Yamada. *Conventional Genetic Algorithm for Job Shop Problems.* Proc. Of the 4th International Conference on Genetic Algorithms. (474 – 479) 1991.

[Narahari 85] Y. Narahari and N. Viswanadham. *A Petri Net Approach to modeling and analysis of Flexible manufacturing systems.* Annals of Operations Research. 3 (449-472) 1985.

[Nasr 90] 75b. N. Nasr and E. A. Elsayed. *Job shop scheduling with alternative machines.* International Journal of Production Research, 28 (1595-1609) 1990.

[Newell 72] A. Newell, and H.A. Simon. *Human problem Solving.* Prentice-Hall, Englewood Cliffs, NJ. 1972.

[Nilsson 82] N. J. Nilsson. *Principles of artificial intelligence.* Springer 1982.

[Noronha 91] S.J. Noronha and V.V.S Sarma. *Knowledge based approaches for scheduling problems: A survery.* IEEE Transactions on Knowledge and Data Engineering. 3 (160-171) 1991.

[Norbis 96] M. I. Norbis and J.M. Smith. *A multiobjective, multi-level huristic for dynamic resource constrained scheduling problems.* European Journal of Operational Research, 33 (30-41) 1996.

[Numerical 92] Numerical Recipes in C: *The art of scientific computing.* 1988-1992. (ISBN 0-521-43108-5) http://www.nr.com

[O'grady 87] P.J. O'Grady and U. Menon. *Loading a flexible manufacturing system.* International Journal of Production research. 25 (1053 – 1068) 1987.

[O'grady 88] P.J. O'Grady and K. H. Lee. *An intelligent cell control system for automated manufacturing.* International Journal of Production research. 26 (845 – 861) 1988.

[Ombuki 88] B. M. Ombuki, M. Nakamura and K. Onaga. *An evolutionary Scheduling Scheme based on gkGA approach to the Job Shop Scheduling Problem.* IEICE Trans. Fundamentals, E81-A, (1063 – 1071) 1988.

[Onaga 91] K. Onaga, M. Silva, T. Watanabe. *On Periodic Schedules for Deterministically Timed Petri Net Systems*. IEEE Proceedings 1991 (210-215).

[Ow 88] P. S. Ow and Thomas E. Morton. *Filtered beam search in scheduling*. International Journal of Production Research. 26 (35-62) 1988.

[Park 89] S. C. Park, N. Raman and M.J. Shaw. *Heuristic learning for pattern directed scheduling in a Flexible Manufacturing System*. Proceedings of the Third ORSA/TIMS Conference on Flexible Manufacturing Systems: Operations Research Models and Application. (369 – 376) 1989.

[Parrish 90] D. J. Parrish. *Flexible Manufacturing*. 1990 Butterworth-Heinemann Ltd.

[Pearl 84] J. Pearl. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley 1984.

[Peng 98] C. Peng, F. F. Chen. *Real Time Control and Scheduling of Flexible Manufacturing Systems: An Ordinal Optimisation based Approach.* International Journal of Advanced Manufacturing Technology, 14 (775-786) 1998.

[Pinedo 95] M. Pinedo. *Scheduling: theory, algorithms and systems*. Prentice Hall, 1995.

[Proth 97] J.M. Proth, L. Wang and X. Xie. *A Class of Petri Nets for Manufacturing system Integration*. IEEE Transactions on Robotics and Automation. 13 (317-326) 1997.

[Proth 96] J.M. Proth and X.L. Xie, *Petri Nets: A tool for design and Management of Manufacturing systems*. John Wiley & Sons. 1996

[Proth 98] J-M Proth and N. Sauer. *Scheuling of piecewise constant product flows: A Petri Net approach*. European Journal of Operational Research. 106 (45- 56) 1998.

[Proth 96b] J-M Proth and N. Sauer. *Continuous approach of scheduling problems based on Petri Nets*. Proceedings of the IEEE. (717 – 723) 1996.

[Rajan 90] V. N. Rajan and S. Y. Nof. *A game-theoretic approach for co-operation control in multi-machine workstations*. International Journal of Computer integrated Manufacturing, 3 (47-59) 1990.

[Raju 93] K. R. Raju and O.V. K. Chetty: *Priority Nets for scheduling Flexible Manufacturing Systems*. Journal of Manufacturing Systems. 12 (326-340) 1993

[Ram 90] B. Ram, S. Sarin, and C.S. Chen. *A model and a solution approach for the machine loading and tool allocation problem in a flexible manufacturing cell*. International Journal of Production Research. 28 (637 – 645) 1990.

[Ramaswamy 97] S. Ramaswamy, K. P. Valavanis and S. Barber. *Petri Net Extensions for the Development of MIMO Net Models of Automated Manufacturing Systems*. Journal of Manufacturing Systems. 16 (175 – 191) 1997.

[Ranky 90] P.G. Ranky 1990. *Flexible Manufacturing Cells and Systems in CIM*. 1990 CIMware Limited.

[Ravichandran 86] R. Ravichandran, A. K. Chakravarty. *Decision Support in Flexible Manufacturing Systems Using Timed Petri Nets*. Journal of Manufacturing Systems, 5 (89-101) 1986.

[Reeves 95] C. R. Reeves. *A genetic algorithm for FlowShop Sequencing*. Computers and Operation Research. 22 (5-13) 1995.

[Reyes 98] A. Reyes, H. Yu & G. Kelleher. *Applying new search methodologies for scheduling FMS using PN*. Proceedings of the sixteenth workshop of the UK Planning and Scheduling Special Interest Group. Huddersfield, UK. September 1998.

[Reyes 99] A. Reyes, H. Yu & G. Kelleher. *Petri Nets, Heuristic Search and Natural Evolution: A Promising Scheduling Algorithm for Job Shop Systems*. In Proceedings of the 3rd International Symposia on Intelligent Industrial Automation. Genoa Italy, June 1999.

[Reyes 00] A. Reyes, H. Yu & G. Kelleher. *Advanced Scheduling Methodologies for Flexible Manufacturing Systems using Petri Nets and Heuristic Search.* Accepted for presentation at the IEEE International Conference on Robotics and Automation. ICRA200 San Francisco, 24-28 April 2000.

[Reyes 00b] A. Reyes, H. Yu & G. Kelleher. *A PN reachability graph branching scheme with application to the scheduling of Flexible Manufacturing Systems.* Accepted for presentation at IFAC International Conference on Control Systems Design 2000, Bratislava, 19-22 June 2000.

[Reyes 00c] A. Reyes, H. Yu, G. Kelleher and S. Lloyd. *Petri Nets based scheduling of Flexible Manufacturing Systems using Hybrid Heuristic Search.* UKACC International Conference on Control 2000. Cambridge, 4-7 September 2000.

[Rodammer 88] F. A. Rodammer and K. P. White. *A Recent Survey of Production Scheduling*. IEEE Transactions on Systems Man and Cybernetics. 18 (841-851) 1988.

[Rogers 91] R. V. Rogers and K. P. White. *Algebraic, Mathematical Programing and Network models Of the Deterministic Job-Shop Scheduling Problem*. IEEE Transactions on Systems Man and Cybernetics. 21 (693-697) 1991.

[Sabuncuoglu 88] I. Sabuncuoglu and D. L. Hommertzheim. *Schedule Generation in A Flexible Manufacturing System: A knowledge-based approach.* Decision Suport Systems. 4 (157-166) 1988.

[Sabuncuoglu 92] I. Sabuncuoglu and D. L. Hommertzheim. *Dynamic dispatching algorithm for scheduling machines and automated guided vehicles in a flexible manufacturing system.* International Journal of Production Research. 30 (1059-1079) 1992

[Sahraoui 87] A. Sahraoui, H. Atabackche, M. Courvoisier, R. Valette. *Joining Petri Nets and Knowledge based Systems for Monitoring purposes*. Proc. IEEE Conf. Robotics and Automation. (1160-1165) 1987.

[Sakamo 94] S. Sakamoto, J. Koga, Y. Shimocaichi and S. Matsumoto*: Scheduling of a Batch Plant by Petri Net*. Journal of Chemical Engineering of Japan. 27 (241-244) 1994.

[Sakawa 96] M. Sakawa, K. Kato and T. Mori. *Flexible scheduling in machining center through genetic algorithms*. Computers Ind. Engineering. 20 (931-940) 1996.

[Santos 95] D. L. Santos, J. L. Hunsucker and D. E. Deal. *Flowmult: Permutation sequences for flow shops with multiple processors*. Journal of Information and Optimization Sciences. 16 (351-366) 1995.

[Santos 96] D. L. Santos, J. L. Hunsucker and D. E. Deal. *An evaluation of sequencing heuristics in flow shops with multiple processors*. Computers in Industrial Engineering. 30 (681 – 692) 1996.

[Sarin 87] S. C. Sarin and C. S. Chen. *The machine loading and tool allocation problem in a flexible manufacturing system*. International Journal of Production Research. 25 (1081 – 1094) 1987.

[Sarkar 91] U. K. Sarkar, P.P. Chacrabarti, S. Ghose and S. C. De Sarkar. *Reducing reexpansions in iterative deepening search by controlling cutoff bounds*. Artificial Intelligence. 50 (207-221) 1991.

[Sauve 87] B. Sauve and A. Collinot. *An expert system for scheduling in a flexible manufacturing system*. Robotics and Computer-Integrated Manufacturing. 3 (229-233) 1987.

[Sawik 93] T. Sawik. *Scheduling of Machines and Vehicles in an FMS: Single-Level Versus Multi-Level Approach*. Automatyka. 64 (541-568) 1993.

[Schmidt 92] Schmidt and Kreutzfeld : *ESPRIT-Project 245/ FLEXPLAN: Integrated Process Planning and Workshop Scheduling*. Human Aspects in Computer Integrated Manufacturingodern Tools for Manufacturing Systems. pp. 699-709 Elsevier Science Publishers. 1992.

[Shakhlevich 96] N. V. Shakhlevich, Y. N. Sotskov and F. Werner. *Adaptive scheduling algorihtm based on mixed graph model*. IEE Proceedings on Control Theory and Applications. 1996 Vol. 143, pp. 9-16.

[Shanker 85] K. Shanker and Y-J. J. TZEN. *A loading and dispatching problem in a random flexible manufacturing system*. International Journal of Production Research, Vol. 23 No. 3, 579-595, 1985.

[Shaw 88b] M. J. Shaw. *A Pattern-Directed Approach to FMS scheduling*. Annals of Operations Research. 15 (353 – 376) 1988.

[Shaw 89] M. J. Shaw and A. B. Whinston. *An Artificial Intelligence Approach to the Scheduling of Flexible Manufacturing Systems*. IIE Transactions. 21 (170 – 183) 1989.

[Shaw 88] M. J. Shaw. *Knowledge-based scheduling in flexible manufacturing systems: an integration of pattern-directed inference and heuristic search*. International Journal of Production Research. 26 (821 – 844) 1988.

[Shaw 92] M. J. Shaw, S. Park and N. Raman. *Intelligent scheduling with Machine Learning Capabilities: The induction of Scheduling Knowledge.* IIE Transactions. 24 (156 – 168) 1992.

[Shen 88] S. Shen and Y-L. Chang. *Schedule Generation in A Flexible Manufacturing System: A knowledge-based approach.* Decision Suport Systems. 4 (157-166) 1988.

[Shen 92] L. Shen, Q. Chen and J.Y.S. Luh: *Truncation of Petri net models for simplifying computation of optimun scheduling problems.* Computers in Industry. 20 (25-43) 1992.

[Shih 91] H.M. Shih and T Sekiguchi: *A Timed Petri Net and beam search based on-line FMS scheduling system with routing flexibility.* Preceedings of the 1991 IEEE International Conference on Robotics and Automation. Sacramento California- (2548-2553) April 1991.

[Shukla 96] C. S. Shukla and F. F. Chen. *The state of the art in intelligent real-time FMS control: a comprehensive survey.* Journal of Intelligent Manufacturing, 7 (441-455) 1996.

[Sikora 96] R. Sikora. *A genetic algorithm for integrating lot-sizing and sequencing in scheduling a capacited flow line.* Computers Industrial Engineering. 30 (969-981) 1996.

[Silva 89] M. Silva and R. Valette, *Petri Nets and Flexible Manufacturing.* In Advances in Petri Nets 1989, Lecture Notes in Computer Science, Springer-Verlag. Vol. 424 (374-417) 1989.

[Soo 92] L. G. Soo, Yoon Jung-Mo: *An Empirical Study on the Complexity Metrics of Petri Nets.* Microelectronics and Reliability 32 (323-329) 1992.

[Spachis 79] A.S. Spachis and J.R. King. *Job Shop scheduling heuristics with local neighbourhood search.* International Journal of Production Research. 17 (507-526) 1979.

[Stecke 83] K. Stecke. *Formulation and solution of nonlinear integer production planning problems for flexible manufacturing systems.* Management Science. 29 (273-288) 1983.

[Steffen 86] M. S. Steffen. *A Survey of Artificial Intelligence-Based Scheduling Systems.* Proceedings of 1986 Fall Industrial Engineering Conference. Norcross, Georgia. (395-404) 1986.

[Sun 94] T. Sun, C. Cheng and L. Fu: *A Petri Net Based Approach to Modeling and Scheduling for an FMS and a Case Study.* IEEE Transactions on Industrial Electronics, 41 (593-601) 1994.

[Sutdhiraksa 96] S. Sutdhiraksa and R. Zurawski. *Evaluation of scheduling approaches for Robotic Assembly Tasks using Petri Nets.* Proceedings of ISIE'96 (475-480) 1996.

[Tadei 95] R. Tadei, F. Della Croce and G. Menga. *Advanced search techniques for the Job Shop Problem: A comparison.* Operations Research. 29 (179-194) 1995.

[Tamura 89] H. Tamura, K. Yamagata; I. Hatono. *Decision Making for Flexible Manufacturing -OR and/or AI approaches in Scheduling* -. Systems analysis modeling simulation. 6 (363-371) 1989.

[Tanida 92] T. Tanida, T. Watanabe, M. Yamauchi and K. Onaga. *Priority-List Scheduling in Timed Petri Nets.* IEICE Transactions. Fundamentals. E75-A, (1394 – 1406) 1992.

[Tsukiyama 92] M. Tsukiyama, K. Mori, and T. Fukuda. *Dynamic scheduling with Petri-Net Modeling and Constraint-based Schedule Editing for Flexible Manufacturing Systems.* Lecture Notes in Control and Information Sciences. Vol. 180, (913-922) 1992.

[Tzafestas 93] S.Tzafestas, and A. Triantafyllakis, *Deterministic Scheduling in Computing and Manufacturing Systems: A Survey of Models and Algorithms*, Mathematics and Computers in Simulation. 35 (397-434), 1993.

[Uckun 93] S. Uckun, S. Bagchi, K. Kawamura and Y. Miyabe. *Managing Genetic Search in Job Shop Scheduling.* IEEE Expert. October 1993 (15-24)

[Ulusoy 97] G. Ulusoy, F. Sivrikaya, and U. Bilge. *A Genetic algorithm approach to the simultaneous scheduling of machines and automated guided vehicles.* Computers and Operations Research. 24 (335 – 351) 1997.

[Uzam 98] M. Uzam and A. H. Jones. *Discrete event Control System Design Using Automation Petri Nets and their Ladder Diagram Implementation.* The International Journal Advanced Manufacturing Technology. 14 (716 – 728) 1998.

[Vasquez 91] A. Vasquez and P. B. Mirchandani. *Concurrent Resource Allocation for Production Scheduling*. Proceedings of the 1991 IEEE International Conference on Robotics and Automation. (1060-1066) 1991.

[Villarroel 88] J.L. Villarroel, J. Martinez, M. Silva, *GRAMAN: A graphic system for manufacturing system design*. Proceedings of IMACS symposium on System Modelling and Simulation, September 1988 (311-316).

[Viswanadham 90] N. Viswanadham, Y. Narahari, and T. L. Jhonson. *Deadlock Prevention and Deadlock Avoidance in Flexible Manufacturing Systems Using PN models*. IEEE Transactions on Robotics and Automation. 6 (713-723) 1990.

[Wang 96] L. Wang and X. Xie. *Modular Modelling Using Petri Nets*. IEEE Transactions on Robotics and Automation, 12 (800-809) 1996.

[L. C. Wang 96] L. C. Wang: *Object-oriented Petri nets for modelling and analysis of automated manufacturing systems*. Computer integrated Manufacturing Systems. 10 (111-125) 1996.

[Wang 98] L-C. Wang and S-Y. Wu. *Modeling with colored Timed Object-Oriented Petri Nets for Automated manufacturing systems*. Computers ind. Engineering. 34 (463-480) 1998.

[Werner 95] F. Werner and A. Winkler. *Insertion techniques for the heuristic solution of the job-shop problem*. Discrete Applied Mathematics. 58 (191-211) 1995.

[Wilhelm 85] W. E. Wilhelm H-M Shin. *Effectiveness of alternate operations in a flexible manufacturing system*. International Journal of Production Research. 23 (65 – 79) 1985

[Witness] Witness Simulation. Lanner group. http://www.lanner.com/usa.

[Wu 89] S-Z D. Wu and R. A. Wysk. *An application of discrete event simulation to on-line control and scheduling in Flexible Manufacturing*. International Journal of Production Research. 27 (1603 – 1623) 1989.

[Xiong 97] H-H. Xiong and M-C. Zhou. *Deadlock-free Scheduling of an Automated Manufacturing System Based on Petri Nets*. Proceedings of the 1997 IEEE International Conference on Robotics and Automation, Albuquerque, New Mexico. (945-950) 1997.

[Xiong 98] H.H. Xiong and M. Zhou. *Scheduling of Semiconductor Test Facility via Petri Nets and Hybrid Heuristic Search*. IEEE Transactions on Semiconductor Manufacturing, 11 (384,393), 1998.

[Xue 98] Y. Xue, R.M. Kieckhafer and F.F. Choobineh. *Automated construction of GSPN models for flexible manufacturing systems*. Computers in Industry, 37 (17-25) 1998..

[Yamada 92] T. Yamada and R. Nakano. *A genetic algorithm applicable to large-scale job shop problems*. Proceedings 2nd International Conference on Parallel Problem Solving form Natura. Elsevier Science Publishers, North-Holland. (1992).

[Yamamoto 77] M. Yamamoto. *An approximate solution of machine scheduling problems by decomposition method*. International Journal of Production Research. 15 (599-608) 1977.

[Yan 98] H-S. Yan, N-S Wang, J-G Zhang, X-Y. Cui: *Modelling scheduling and simulation of flexible manufacturing systems using extended stochastic high-level evaluation Petri nets*. Robotics and Computer-Integrated Manufacturing. 14 (121-140) 1998.

[Yan 97] H-S. Yan, N-S Wang, X-Y. Cui, J-G Zhang,: *Modelling scheduling and control of flexible manufacturing systems by extended high-level evaluation Petri nets*. IIE Transactions. 29 (147-158) 1997

[Yim 94] Yim, Nelson and Murata: Predicate-*Transition Net Reachability testing using Heuristic search*. Transactions IEE Japan Vol. 114-C, (907-913) 1994.

[D-S. Yim 94] D-S Yim and T. A. Barta. *A Petri Net-Based Simulation Tool for the Design and Analysis of Flexible Manufacturing Systems*. Journal of Manufacturing Systems, 13 (251-26) 1994

[Yim 93] D. S. Yim and R.J. LINN. *Push and pull rules for dispatching automated guided vehicles in a flexible manufacturing system*. International Journal of Production Research. 31 (43-57) 1993.

[Yim 96] S.J. Yim and D.Y. Lee. *Multiple Objective Scheduling for Flexible Manufacturing Systems Using Petri Nets and Heuristic Search. IEEE Int. Conf. On Systems , Man, And Cybernetics*. Information Intelligence and Systems. (2984-2989) 1996.

[Yu 97] H. Yu, G. Kelleher, A. Ayesh. *Planning through Petri Nets*. Proc. of the 16th Workshop of the UK planning and scheduling interest group. (83-90) 1997.

[Yung 88] S. Yung, D. Wu and R. A. Wysk. *Multi-pass Expert Control System - A Control/Scheduling Structure for Flexible Manufacturing Cells*. Journal of Manufacturing Systems. 7 (107-114) 1988.

[Zamani 97] R. Zamani and Li-Yen Shue. *The application of an admissible heuristic algorithm to project scheduling problem*. International Journal of Computer Applications in Technology, 10 (308-315) 1997.

[Zhang 93] H-C. Zhang. IPPM - *A Prototype to Integrate Process Planning and Job Shop Scheduling Functions*. Annals of the CIRP, 42 (513 – 518) 1993.

[Zhang 90] D. Zhang. *ROPES: A tool for generating Robot Plans*. Proceedings of IECON'90. (210-215) 1990.

[Zhang 92] Du Zhang: *Planning using Timed Pr/T Nets*. JAPAN/USA Symposium on Flexible Automation, 2 (1179-1183) 1992.

[Zhou 93] M. C. Zhou, K. McDermott and P. A. Patel. *Petri Net Synthesis and Analysis of a Flexible Manufacturing System Cell*. IEEE Transactions on Systems, Man, and Cybernetics. 23 (523-531) 1993. .

[Zhou 93b] M. C. Zhou and F. DiCesare, *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems*. Boston, MA: Kluwer, 1993.

[Zhou 92] M. C. Zhou, F. Dicesare, A. A. Desrochers. *A Hybrid Methodology for Synthesis of Petri Nets Models for Manufacturing Systems*. IEEE Transactions on Robotics and Automation, 8 (350-361) 1992.

[Zimmermann 99] A. Zimmermann and G. Hommel. *Modelling and Evaluation of Manufacturing Systems Using Dedicated Petri Nets*. International Journal of Advanced Manufacturing Technology. 15 (132 – 138) 1999.

[Zurawski 94b] R. Zurawski. Systematic *Construction of Functional Abstractions of Petri Nets Models of Flexible Manufacturing Systems*. IEEE Transactions on Industrial Electronics. 41(584 – 592) 1993.

[Zurawski 94] R. Zurawski and M. Zhou. *Petri Nets and Industrial Applications: A Tutorial. IEEE Transactions on Industrial Electronics*. 41 (567-583) 1994.