Ann Oper Res (2010) 181: 795–812
DOI 10.1007/s10479-010-0686-1

Applications of resource assignment and scheduling with Petri Nets and heuristic search

Gonzalo Mejía · Carlos Montoya

Published online: 29 January 2010

Abstract This paper introduces a new Petri Net based approach for resource allocation and scheduling. The goals are (i) minimize the number of required resources given a set of jobs, (ii) find both an assignment for all jobs in the span of a predefined shift and (iii) the sequence in which such jobs are executed. The studied problem was inspired from a complex real life manufacturing shop as described in this document. The modeling of the processes and jobs is carried out with Petri Nets due to their capability of representing dynamic, concurrent discrete-event dynamic systems. The resource assignment starts with an initial feasible solution (initial number of resources) and then follows with a re-optimization process aimed to further reduce the resource requirements. The algorithm is based on a modified Heuristic Search method previously presented. The algorithm was tested first on a number of instances from the literature and then on the aforementioned system (a car seat cover manufacturer). The proposed approach shows not only good results in terms of performance but also shows the potential of Petri Nets for modeling and optimizing real-life systems. An implementation phase at the first stages of the process is underway at the time of writing.

Keywords Petri Nets · Resource assignment · Scheduling · Heuristic search

1 Introduction

Resource allocation and scheduling problems are related to a number of topics which include the calculation of the number of required resources (workers and/or machines), and both the assignment and the scheduling of tasks to such resources. A number of applications of this are commonly found in every day life that different types of manufacturing and service systems such as manufacturing shops, airlines, hospitals, call-centers, banks, etc. (Meisels and Lusternik 1997). In some cases, such as the Employee Timetabling Problem (ETP) and the Workforce Scheduling Problem (WSP), the problem consists of establishing

G. Mejía (✉) · C. Montoya
Departamento de Ingeniería Industrial, Universidad de los Andes, Carrera 1 # 19-40 Office ML703, Bogotá, Colombia
e-mail: gmejia@uniandes.edu.co

the number of resources without subject to mostly cost and timing (shifts) constraints; other problems such as the Flow and Job Shop Scheduling Problem (FSSP and JSSP), and the Resource Constrained Project Scheduling Problem (RCPSP) consider precedence and resource constraints but with a predefined number of resources (Rodammer and Whit 1988; Bachman et al. 2002; Zhang and Bard 2006; Nhu et al. 2007).

In real manufacturing systems, the problem at hand is more difficult to solve: Generally, the resource requirements and the job and workforce schedule must be worked out at the same time as these two problems are interrelated. In addition, scheduling jobs is difficult, not only for its combinatorial nature but also because of complex routings, assembly operations, concurrent and parallel activities commonly found in real settings. In turn, such jobs are processed with workers with different skills and requirements. In many companies, establishing the number of resources on a daily or a weekly basis depending on the workloads together with a job schedule is a highly demanding task that typically uses up many man-hours.

The aim of this paper is to propose a methodology to handle both the problem of determining the minimum number of resources (machines and workers) required to execute a schedule of jobs within a time frame (shift) and the schedule of such jobs. Constraints for this problem include, but are not limited to, the qualifications or capabilities of each resource, precedence and routing constraints. The outputs of the algorithm are: (i) the number of resources of each type, (ii) the assignment of resources to tasks and (iii) the sequence and timing in which all tasks are processed on each resource.

This paper proposes an integrated Petri Net modeling and optimization approach. Having the Petri Net model, the optimization procedure starts with finding an initial solution consisting of both a number of resources and a feasible schedule (i.e. all tasks are finished within the allocated time span (shift)). Next, a re-optimization process based on a Heuristic Graph Search method attempts to reduce the initial number of resources found at the first stage. This Graph Search algorithm is a modified version of the A* Search (A*S) presented in previous work (Mejía and Montoya 2007).

The remainder of this paper is organized as follows: Sect. 2 reviews the basic concepts of modeling with Petri Nets; Sect. 3 describes the proposed method which includes the initial solution and the re-optimization process. Section 4 is devoted to the computational tests on the instances from the literature. Section 5 describes and discusses the modeling, testing and implementation of the methodology on a real life setting. Results of the computer tests are also shown in this section. Finally, the conclusions are presented in Sect. 6.

2 Modeling systems with Petri Nets

A Petri Net is a directed graph, with an initial state called the initial marking M_0. A Petri Net is a directed, weighted, bipartite graph consisting of two kinds of nodes, called places and transitions where arcs are directed either from a place to a transition or from a transition to a place. In graphical representation places are drawn as circles, transitions as bars or boxes. Places usually represent actions or conditions and transitions represent events. Tokens (black dots) reside in places and represent the truth of the condition or action associated with the corresponding place. A place containing at least one token is denoted as a marked place (Murata 1989).

A *marking* is defined as an array which contains in each of its positions the number of tokens at each place. Tokens move throughout the net by effect of transition firings. A tran-

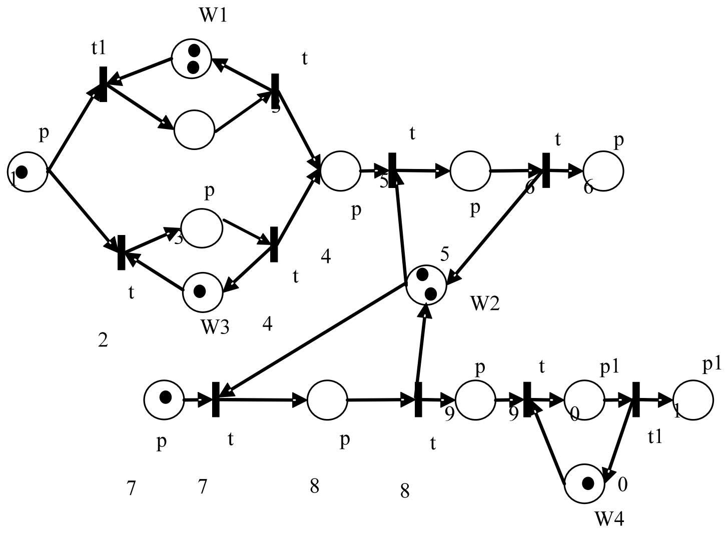

Fig. 1 Petri Net Model of a system with two jobs and four workstations

sition is enabled when all its input places are marked (contain at least one token).[1] If an enabled transition fires, one token is removed from all the transition input places and one token is put on each of the transition output places. A marking (state of the net) M' is reachable from another marking M if there exists a valid sequence of transition firings which converts M into M'. A Timed Place Petri Net (TPPN) considers time associated with places. In TPPNs, in addition to the token rule for transition firing, a transition is enabled when the time associated with each of its input places has been exhausted (Mejía and Odrey 2005).

Definition Timed Place Petri Net (TPN): A TPN is a 6-tuple $(P, T, P \times T, T \times P, M_0, \tau)$ where $P =$ set of places, $T =$ set of transitions, $P \cap T = \{\}$, $P \times T =$ a set of input arcs, $T \times P =$ a set of output arcs, $M_0 =$ initial marking, and $\tau =$ set of time delays associated with places.

This paper followed the methodology concept of S^{4}R nets (Systems of Sequential Systems with Shared Resources (Abdallah et al. 2002)) for modeling manufacturing systems. In S^{4}R nets, a number of concurrent serial processes (jobs) are modeled with an equal number of strongly connected state machines. Places belonging to such processes are denoted as "Operational Places" and can be timed if representing the execution of an operation, or untimed if representing a condition such as "ready", "in queue" or "finished". A kind of Operational Place is the Source Place (with no input transitions) that represent the condition "component, (part, job) available". Resource capacity constraints are incorporated to the model via "Resource Places". All resource places are untimed.

Figure 1 illustrates a Petri Net model of a manufacturing system with two machines and two jobs. Each job has a sequence to follow: Job 1 is processed first either at the workstation 1 or at the workstation 3 and then at workstation 2; job 2 is processed first at workstation 2 and then at workstation 4. The number of tokens at each resource place (places W1 to W4

[1] This is the concept of ordinary Petri Net in which all arc weights are set to 1.

Table 1 Description of Timed Place for the Petri Net in Fig. 1

p1	Job 1 in queue	p9	Job 2 in buffer waits for machine 1
p2	Job 1 processed by machine 1	p10	Job 2 processed by machine 1
p3	Job 1 processed by machine 3	p11	Job 2 finished
p4	Job 1 in buffer waits for machine 2	W1	Workstation 1 available
p5	Job 1 processed by machine 2	W2	Workstation 2 available
p6	Job 1 finished	W3	Workstation 3 available
p7	Job 2 in queue	W4	Workstation 4 available
p8	Job 2 processed by machine 2		

in Fig. 1) corresponds to the number of identical machines or workers in parallel available at each workstation at a given marking. Table 1 describes the places in the net.

3 The proposed resource assignment and scheduling methodology

3.1 The Resource Allocation A* Search algorithm (RAAS)

Several attempts have been made to reduce the effort to search the entire state space of Petri Nets. Perhaps the most widely used approach has been the application of modified versions of the commonly known A* search algorithm (Lee and DiCesare 1994; Xiong and Zhou 1998; Reyes-Moro et al. 2002; Low and Wu 2001; Yu et al. 2003; Kim et al. 2007; Mejía and Montoya 2008, 2009). The A* search algorithm is a branch and bound-like algorithm which expands only the most promising nodes of the Petri Net reachability graph, which contains all the possible states of the net. The root node of this graph is the initial marking M_0, and its successors are generated by firing one transition at a time, thus obtaining new markings (nodes).

The algorithm A* search expands branches of the net reachability graph according to a criterion established with the heuristic function $f(M) = g(M) + h(M)$. The function $g(M)$ is the actual cost from the initial marking M_0 to the marking M and $h(M)$ is an estimate of the cost from marking M to the desired final marking M_f. The function $h(M)$ is *admissible* if underestimates the true value of the cost to reach the final marking. The list of candidate markings is sorted values of $f(M)$ will have priority for expansion (Lee and DiCesare 1994).

3.2 The RAAS algorithm implementation

The A* Search algorithm guarantees optimality if the $h(M)$ function is admissible for all markings but has exponential complexity in both memory and time (Mejía and Odrey 2005). For this reason a number of improvements, mostly on limiting the nodes that are expanded at the expense of optimality, have been presented in the literature (Mejía and Odrey 2005; Xiong and Zhou 1998; Yu et al. 2003). The RAAS algorithm developed in this paper was adapted from the BAS (Beam A* Search) presented in previous work (Mejía and Odrey 2005) to further speed up the search. The BAS algorithm had improvements over the classic A* search that include: (i) tighter bounds for the $h(M)$ function, (ii) reduction of the search space by trimming out non-promising nodes, and (iii) limited backtracking. The BAS algorithm proved effective on classical Job Shop, Flexible Job Shop and Flexible Manufacturing Systems (FMS) scheduling problems both in solution quality and computer time for the makespan (Mejía and Odrey 2005) and total tardiness criteria (Mejía and Montoya

2008). For the classical JSSP instances. The average deviation from the optimal solution (makespan) was approximately 4%,[2] the highest deviation from the optimal solution was around 7% and in about 35% of the cases the optimal value was found. The RAAS algorithm changes the expansion strategy by cutting off those nodes whose value of $f(M)$ are greater than the shift length and limiting the number of iterations without finding a feasible schedule. This version of the algorithm was implemented in JAVA and run on a PC having 2 GB RAM.

3.3 The Resource Assignment and Scheduling algorithm (RAAS)

The primary idea of the algorithm is to start with a preliminary schedule, calculated with either simple dispatching rules or simple heuristics and an assignment of resources and then iteratively add resources until the shift duration constraint is met (feasible solution in which all jobs are finished within the shift span). Next, improve the schedule with the RAAS algorithm and iteratively decrease the number of resources until no further reduction can be achieved without violating the shift constraint.[3] An important assumption is that the shift length must be greater than both the maximum of the total operation times for all jobs and all the machine workloads. In this way, the algorithm is guaranteed to find feasible assignments of resources.

Inputs:

a. A manufacturing system with n jobs to be processed, all available at the beginning. Each job has a route that includes (some or all) precedence, buffer (intermediate storage) and resource eligibility constraints and known processing times. There may also be alternative routings, multiplicity of resources (one or more resources required to perform one operation), recirculation, non-identical parallel machines, and all features that can be modeled with Petri Nets (Murata 1989).
b. A predefined shift duration in which all jobs must be completed.
c. An initial distribution of resources of each type. In the case of the studied FJSSPs the number of initial resources of each type is set to one unit. This means that initially only one token is put in each resource place ($N_r = 1$).

Outputs:

a. The number of (identical) resources of each type.
b. A feasible schedule.

Steps: The general steps of the proposed methodology are the following:

1. Model the system using S^4R Petri Nets.
2. Find an initial solution feasible schedule and an initial value of the number of resources of type $r(N_r)$. The number of resources of each type corresponds to the initial number of tokens at each resource place.
3. Re-optimize to both minimize the number of resources and to improve resource utilization (analogous to minimize Makespan).

[2] When the optimal solution was available.

[3] Another tested strategy was: Start with a resource assignment and a schedule established with RAAS. Add iteratively resources until the shift constraint is met. This strategy proved to be a lot slower than the proposed in this paper.

3.4 Initial solution

The objective of this stage is to find an initial value for the number of resources given the set of tasks, resource capacity and technical constraints and length of the shift. These are the steps:

1. Find a preliminary schedule using a priority rule to sequence all tasks (jobs). Three dispatching rules were tested (SPT, LPT and MWR).[4] The rule that provided with the best objective function was selected as the initial solution.
2. In the case of that one or more jobs could not be completed in the required shift duration, increase in one unit the resource with the highest utilization rate ($N_r = N_r + 1$).
3. Repeat steps 2 and 3 until all jobs can be finished within the shift length. It is assumed that the sum of processing times of a job cannot exceed the shift duration. Otherwise there would not be a feasible solution.

3.5 Re-optimization process

In this stage, an attempt is made to selectively reduce the number of resources. Let *checked_resources* be a list of resources which have been reduced in size. This stage has three phases:

1. Re-sequence the operations of all tasks, finding a new sequence of transition firings. This phase is performed with the RAAS algorithm.
2. Originally this list is empty. Reduce in one unit the size of the resource type r with the lowest utilization rate ($N_r = N_r - 1$) that is not in the list ($r \notin$ *checked_resources*), and re-run the RAAS algorithm. Put r in *checked_resources*.
3. Re-check that all tasks can be completed within the shift duration. If yes, go directly to step 3. If no, re-set the size of the resource r with the lowest utilization rate ($N_r = N_r + 1$) and go to step 3. The re optimization process stops when the number of all resources cannot be further reduced (the shift limit is exceeded).

4 Computational experiments and results

The performance of the algorithm was evaluated in two parts: The first part (Sect. 4.1) involved a test on classical scheduling problems. The test included classical job shop and a set of flexible manufacturing systems scheduling problems; the second part (Sect. 5.2) corresponds to application of the proposed modeling methodology and the testing of the algorithm on a real manufacturing company.

4.1 Problems from the literature

In this section we study the performance of the algorithm on problems adapted from the literature. The primary idea is to show that this methodology can be easily adapted to a variety of manufacturing systems. The first set consists of classical Job Shop Scheduling Problems (JSSP) converted into Flexible Job Shop Scheduling problems (FJSSP). A JSSP (or a FJSSP) consists of a number of jobs that follow a pre-defined sequence of machines

[4]SPT (Shortest Processing Time); LPT (Longest Processing Time); MWR (Most Work Remaining).

which they must visit. Each machine can process one job at a time and all jobs and machines are available at the beginning of the planning horizon. A FJSSP extends the concept of the JSSP with a pool of identical parallel machines at each workstation. Twelve 10×10 (jobs $\times$ machines) JSSP instances were studied. Such instances were proposed by Lawrence (1984), Applegate and Cook (1991) and Adams et al. (1988). These instances were adapted for this test as follows: We start with the classical JSSP instance (one machine per station) and then we iteratively add (with the RAAS algorithm) machines to stations until the shift constraint is met. As parallel machines are added to workstations, the original JSSP instance is converted into a FJSSP.

The lengths of the shifts were defined as β times the maximum of the Total Work Time (TWT) of all jobs:

$$\textit{Shift Length} = \beta \times \max_j \left(\sum_i^m p_{ij} \right)$$

The TWT of job j is the sum of all processing times p_{ij} over all machines, indexed in i (1 to m), in the system. If job j does not visit machine i.

For each of the tested instances different shifts lengths were generated, varying the value of β. Shift lengths were set in such way that their values were greater or equal than the maximum of the Total Work Time (TWT) of all jobs and less than the optimal makespan value for the JSSP instance (where only one machine per work station is required). Thus the lowest value of the shift length was obtained when β is equal to 1. Other shift lengths were obtained by increasing the value of β ($\beta = 1$, 1.1, 1.2 and so forth). The selected *maximum value of* β is such that the corresponding shift length does not exceed the optimal makespan value for the JSSP instance.

For the JSSP instances, the results of the proposed algorithm (initial solution + re-optimization) are compared against the solutions provided by two algorithms: The General Shifting Bottleneck algorithm (GSB) (Implemented in the software Lekin® by Pinedo and Chao 1999) and the Flexible Job Shop Java Genetic Algorithm (FJSJGA) (Gutiérrez and Mejía 2009). The GSB algorithm was chosen due to its availability and its "general purpose" characteristic which makes it easily adapted to a number of settings (as it is the case of RAAS). The FJSJGA is a recent development that has been extensively tested with a large number of Flexible Job Shop instances from the literature and has reached the best known solutions (makespan) in the majority of the cases (the average deviation from the best known solution is around 2%). Notice that the GSB and the FJSJGA algorithms are aimed for scheduling purposes only and do not establish the number of required machines at each workstation. For both the GSB and the FJSJGA routines, the number of parallel machines at each workstation was set equal to the number of machines of each type found with the RAAS algorithm. Unlike the JSSP instances, for most classical 10×10 FJSSP instances the optimal solutions are not known.

The second set of scheduling problems can be characterized as Flexible Manufacturing Systems. Three instances proposed by Lee and DiCesare (1994) were studied. Each instance consists of a number of jobs with alternative routings, parallel machines, recirculation, multiplicity of resources and batch sizes. For these instances the optimal values are not yet known. For this reason, we used the best known solutions from Reyes-Moro et al. (2002). In previous work (Mejía and Odrey 2005), we showed that our Petri Net scheduler (BAS) reached solutions within 1% of the solutions of Reyes-Moro et al. (2002). For these problems, the *TWT* is route dependent and therefore the shift spans were set as α times the best known makespan values. The values of α were set between 0.4 and 0.95.

Tables 2 and 3 show the results obtained for the job shop instances. Each JSSP instances consists of 10 jobs and 10 workstations with one machine each. Column 2 shows the shift length, while columns 3, 4 and 5 show respectively the rule(s) that obtained the best initial solution, the number of initially added machines and the initial makespan obtained with such rule(s). The next columns show the final results obtained by the RAAS algorithm: Column 6 shows the number of added machines per workstation; column 7 the total number of machines added, and column 8 shows the makespan and CPU time (sec). Columns 10 and 11 show respectively the results of the GSB and FJSJGA algorithms. We did not include computer times for such algorithms as these are not comparable: The RAAS runs the scheduling routine several times until the shift length constraint is met whereas the GSB and FJSJGA algorithms run their scheduling routines once with the assignment of resources defined by RAAS as input. Tables 2 and 3 show for each of the tested instances the results for the minimum ($\beta = 1$) and the maximum shift lengths (β max varies from instance to instance) respectively. The number of additional machines vs. different values of shift lengths (varying the values of β) can be seen in Fig. 2 for selected instances. Tables 4 and 5 illustrate the results of the RAAS algorithm on the more complex Lee and DiCesare (1994). These tables follow the same format as Tables 2 and 3 and are self explanatory. In this case showed results considered a minimum ($\alpha = 0.4$) and maximum ($\alpha = 0.95$) shift length. Also Fig. 3 shows for the Lee and DiCesare (1994) instances (denoted here as ld3, ld4a and ld4b). These results considered all the shifts lengths generated for each of the tested instances.

4.2 Analysis

The above results show that, considering all the job shop instances shown in Tables 2 and 3, in 75%, 10% and 20% of all of such problems the best initial solution was obtained with the MWR, LPT and SPT dispatching rules respectively. On the other hand, it is possible to obtain the same initial solution with different dispatching rules. For example for the ABZ5 instance in Table 2, the same initial solution was obtained with the MWR, SPT and LPT dispatching rules.

The results in Tables 2 and 3 show the validity of the proposed approach: The RAAS algorithm outperformed the GSB in 22 out of 24 instances and the FJSJGA in 20 out of 24 with the same number of machines. Also, the values of makespan obtained were greater than the predetermined shift length in 16 of the 24 tested instances for GSB and in 15 out of 24 for the FJSJGA.

Notice also that the shorter the shift length, the better the RAAS algorithm seems to perform as compared to the GSB and the FJSJGA routines. These can be seen in the results from Tables 2 and 3. For short shift lengths (see Table 2), the RAAS algorithm outperforms the GSBA and FJSJGA. The deviations are 6% and 5% respectively from the makespan value of the RAAS algorithm. This suggests that the RAAS algorithm performs best in tight shift lengths. In Table 3 (longer shifts), the results do not appear conclusive as on average the results are very similar. The makespan from the GSBA and the FJSJGA are in average 2% and 0% respectively in comparison to the makespan value obtained with the RAAS algorithm.

Tables 4 and 5 show the results when the shift spans were fixed to 0.95 and 0.4 times the values of makespan obtained by the DLSS algorithm proposed by Reyes-Moro et al. (2002). Considering the FMS instances shown in Tables 4 and 5, in the 50%, 0% and 50% of all such problems the best initial solution was obtained with the MWR, LPT and SPT dispatching rules respectively. If in such analysis it is included all shift lengths generated for each FMS instance, in the 32%, 17% and 53% of all such problems the best initial solution was obtained with the MWR, LPT and SPT dispatching rules respectively.

Table 2 Results for the minimum shift length (Shift length = $(\beta = 1) \times \max(TWT)$ for the JSSP instance)

Problem	Shift length	RAAS algorithm							FJSJGA	GSB
		Initial solution			Final solution					
		Best rule			Additional machines	Machines added per workstation	$C_{\max}$	CPU (sec)	$C_{\max}$	$C_{\max}$
		Rule	Additional machines	$C_{\max}$						
ABZ5	859	MWR, SPT, LPT	23	859	15	$(1, 4, 5, 6, 8, 9) + 1$; $(2, 3, 10) + 2$; $(7) + 3$*	859	189	952	859
ABZ6	742	MWR	11	742	10	$(1, 2, 3, 4, 7, 8, 9) + 1$; $(6) + 2$	742	97	752	762
LA16	717	SPT	11	717	9	$(1, 4, 5, 6, 7, 8, 10) + 1$; $(3) + 2$	717	98	731	741
LA17	646	SPT	12	646	7	$(1, 2, 6, 7, 8, 9) + 1$; $(4) + 2$	646	93	646	650
LA18	829	SPT	4	819	3	$(2, 3, 4) + 1$	808	32	804	862
LA19	617	MWR, SPT	20	617	13	$(1, 2, 3, 7, 8, 9, 10) + 1$; $(4, 5, 6) + 2$	617	201	672	632
LA20	756	LPT	10	756	9	$(1, 2, 4, 5, 6, 7, 8, 9, 10) + 1$	756	120	763	756
MT10	655	SPT	16	655	15	$(4, 5, 6, 9, 10) + 1$; $(1, 2, 3, 7, 8) + 2$	655	125	713	655
ORB1	695	MWR, SPT	18	695	16	$(4, 7, 8, 9, 10) + 1$; $(2, 3, 5, 6) + 2$; $(1) + 3$	695	104	795	706
ORB2	681	MWR	11	649	9	$(1, 2, 3, 4, 5, 6, 8, 9, 10) + 1$	651	135	726	778
ORB3	648	SPT	29	648	24	$(8) + 1$; $(1, 2, 7, 9) + 2$; $(3, 4, 5, 6, 10) + 3$	648	162	701	648
ORB4	753	LPT	14	753	11	$(2, 3, 4, 5, 6, 7, 8, 9, 10) + 1$; $(1) + 2$	753	140	774	787
ORB5	584	SPT	28	584	22	$(1, 2, 5, 7) + 1$; $(3, 4) + 2$, $(8, 9) + 3$; $(6) + 4$	584	168	641	584

$(1, 4, 5, 6, 8, 9) + 1$; $(2, 3, 10) + 2$; $(7) + 3$ means that 1 machine was added to workstations 1, 4, 5, 6, 8, 9; 2 machines were added to workstations 2, 3 and 10; and 3 machines were added to work station 3

CPU: Computer time in seconds. $C_{\max}$: Value of makespan. TWT: Total Work Time of a job

Table 3 Results for the maximum shift length (Shift length $= \beta \times \max(TWT)$ for the JSSP instance)

Problem	Shift length	RAAS algorithm							FJSJGA	GSB
		Initial solution			Final solution					
		Best rule			Additional machines	Machines added per workstation	$C_{\max}$	CPU (sec)	$C_{\max}$	$C_{\max}$
		Rule	Additional machines	$C_{\max}$						
ABZ5	1203	SPT	3	1163	3	$(3, 5, 7) + 1$*	1163	22	1169	1183
ABZ6	928	MWR	2	887	2	$(1, 6) + 1$	887	16	911	903
LA16	897	MWR	3	858	1	$(2) + 1$	896	17	864	923
LA17	776	MWR	3	713	2	$(2, 4) + 1$	749	23	745	729
LA18	663	LPT	15	663	10	$(1, 2, 3, 4, 5, 6, 7, 8, 9, 10) + 1$	663	150	708	669
LA19	833	MWR	3	820	2	$(6, 9) + 1$	829	18	818	911
LA20	870	MWR	4	851	3	$(2, 4, 9) + 1$	849	32	867	880
MT10	917	MWR	4	899	3	$(1, 2, 3) + 1$	913	31	897	906
ORB1	1043	SPT	4	1042	4	$(1, 2, 3, 5) + 1$	1036	22	1013	1035
ORB2	886	MWR	4	839	2	$(1, 4) + 1$	885	25	846	1092
ORB3	972	MWR	8	955	5	$(1, 3, 4, 5, 6) + 1$	972	42	948	958
ORB4	979	SPT	5	966	3	$(1, 2, 4) + 1$	973	35	917	1023
ORB5	876	MWR	5	863	3	$(2, 7, 10) + 2$	866	29	874	875

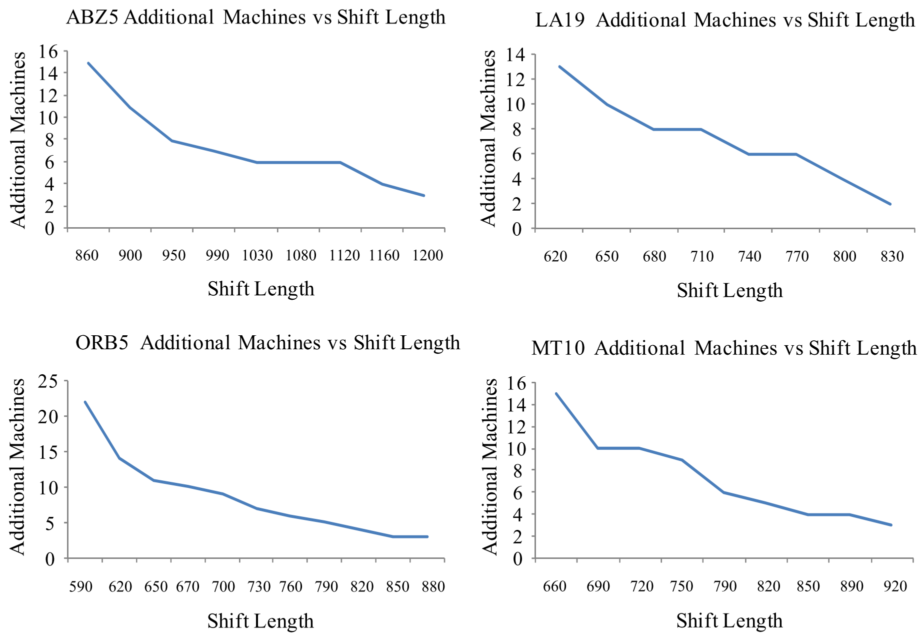

Fig. 2 Comparison between shift lengths and number of added machines for job shop instances

Table 4 Results for the FMS Instances with a shift span equals to ($\alpha = 0.4$) times the makespan values found with the DLSS algorithm (Reyes-Moro et al. 2002)

Problem	Shift length	RAAS algorithm						
		Initial solution			Final solution			
		Best rule			Additional machines	Machines added per workstation	$C_{\max}$	CPU (sec)
		Rule	Additional machines	$C_{\max}$				
ld3	132	SPT	7	117	6	$(3) + 1$; $(2) + 2$; $(1) + 3$	125	38
ld4a	102	MWR	20	101	14	$(2) + 1$; $(4, 5) + 2$; $(1, 3, 6) + 3$	100	342
ld4b	95	SPT	24	94	14	$(1) + 2$; $(3, 4, 5, 6) + 3$	95	434

CPU: Computer time in seconds. $C_{\max}$: Value of makespan

These tables show that very few machines were added when the value of α was close to 1.0 ($\alpha = 0.95$). With a very reduced shift length ($\alpha = 0.4$) the algorithm still found a feasible solution at the expense of adding many machines. For example, for the ld3 instance of Table 4 the RAAS algorithm obtained that 6 machines must be added to complete the all the operations within the defined shift length. The reason why the algorithm was able to add such a large number machines was that these FMS problems had a large number of jobs (about 50) and the value of the obtained makespan was far greater than the TWT of the jobs. As such the shift length was greatly reduced without generating unfeasible solutions. In the

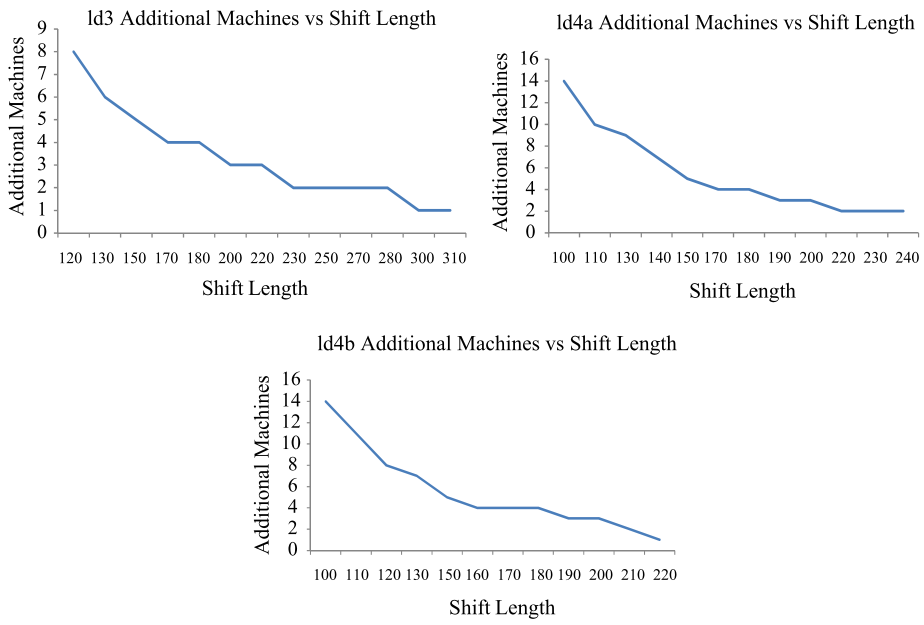

Fig. 3 Comparison between shift lengths and number of added machines for the FMS instances

Table 5 Results for the FMS Instances with a shift span equals to ($\alpha = 0.95$) times the makespan values found with the DLSS algorithm (Reyes-Moro et al. 2002)

Problem	Shift length	RAAS algorithm						
		Initial solution			Final solution			
		Best rule			Additional machines	Machines added per workstation	$C_{\max}$	CPU (sec)
		Rule	Additional machines	$C_{\max}$				
ld3	242	SPT	1	312	1	$(3) + 1$	286	11
ld4a	242	SPT	2	237	2	$(5, 6) + 1$	219	37
ld4b	226	MWR	2	214	1	$(5) + 1$	216	45

CPU: Computer time in seconds. $C_{\max}$: Value of makespan

case of the JSSP instances, a very shortened shift may have a duration shorter than the TWT of some job and therefore it would be impossible to find a feasible solution.

5 An application in a complex system: manufacturing of car cover seats

The methodology proposed in this paper was inspired from a company that produces car seat covers in Colombia (South America). The company is both supplier of large car assembly plants ("serial production", which is the term coined in the company) and of customers who order small quantities or special orders of seat covers (make-to-order production). This study focuses on the make-to-order production. In a typical day the company receives up to 30 special orders which must be processed within one calendar day; the schedule is calculated daily. All unfinished jobs at the end of the day are interrupted and re-scheduled for the next day. Thus, the plant can be considered as empty (no jobs) at the beginning of each day.

Despite the fact that the company uses ERP software, the company suffers from poor of coordination at the final assembly (sewing station) which in turn results in waste of resources. Sometimes, the sewing workers remain idle for hours due to the lack of components and sometimes the workforce is insufficient to meet due dates. Additional sewing personnel can be brought in easily with a 24-hour notice. The plant management has determined that it is mandatory to have a model that predicts both the workforce requirements and the schedule of jobs.

5.1 The manufacturing process and the Petri Net modeling

The manufacturing process itself has many features that complicate the scheduling task: These comprise both concurrent and alternative routings, assembly operations, machine eligibility constraints, batching, non identical parallel machines and so forth. Furthermore the company has very tight due dates. In fact, the company just dropped the implementation of a commercial scheduling software package because such a package did not fit their needs. The company has facilitated the information of products, machines, routing sheets and shift durations and expects that this research proves effective. At the time of writing the project has been underway for 6 months and the company is currently implementing a prototype in the first stages of the process. Most of the time has been on data collection, interfacing with the ERP software and modeling the system.

The current manufacturing process starts with four processes that can be performed concurrently: leather cutting, vinyl cutting, leather punching and carpet cutting. Leather cutting can either be made manually by cutting workers or by a CNC cutting machine. Vinyl cutting is done manually by the cutting workers if the batch size of the job is less than four (4) sets of cover seats or otherwise with a sheet cutting machine. In the first case, the vinyl and the leather pieces are cut concurrently by one cutting worker. In case of machine cutting, such a machine can cut up to 60 vinyl sheets at the same time. After the vinyl cutting operation, a punching operation is done on the cut vinyl parts. Next, a first assembly operation is performed. Here polyurethane foam pads are attached to all leather and vinyl parts. The next operation, called “setup” by the plant personnel, consists of picking plastic frames from the warehouse and get them ready for the final assembly. In this operation the leather-foam and vinyl parts are sewn onto the car seats. The critical activities (bottleneck) are the leather cutting and the sewing operations.

We selected three typical days in which about 30 jobs had to be processed. Such jobs were downloaded from the ERP system and input to our algorithm via a spreadsheet application. The company runs two eight-hour shifts and the primary objectives are (i) meeting the hard due date of 24 hours for each job and (ii) establishing the schedule for the cutting and sewing stations.

Typical process times for the operations are shown in Table 6. The process time depends on many parameters that include the process itself (manual or automated), the batch size, the complexity (difficulty) of the cutting operations, the configuration (“deluxe” or “regular”) and the number of seats per car.

Most products follow a standard route, although variations are not uncommon. For example, not all products require vinyl cutting (e.g. the “full leather” covers) and only a few require leather punching.

The number resources currently available are: 7 leather cutting workers, 1 worker for carpet cutting, 1 worker at the leather punching press, 1 worker at the vinyl cutting table,

Table 6 Typical processing times

Process	Time (hours)
Leather cutting (manual)	1.5–3
Leather cutting (CNC)	0.5–1.5
Vinyl cutting (machine)	2–4
Leather punching	0.5–1.5
Vinyl punching	0.5
Setup	1
Sewing	2–5

2 workers at the first assembly and 10 sewing lines with 5 workers each. The cutting and sewing workers specialize in either serial or make-to-order production.

In addition to all the above technical constraints, other issues further complicate the scheduling: sometimes jobs require reprocessing, raw materials and components are not always available, new products are developed almost every month, etc. Classical exact optimization techniques such as mathematical programming, branch and bound are unlikely to deal with the complexity of this system; on the other hand, heuristic methods are usually difficult to maintain in real settings since they are very rigid and problem specific. As for the company, having information as to the number of workers at the cutting and sewing operations and their schedule, seems the critical thing. For the above reasons, the Petri Net approach combined with the Graph Search optimization algorithms was adequate for this problem.

Figure 4 illustrates the manufacturing process of a standard cover seat as modeled with a Petri Net. Places "px" are operational places whereas places "Wx" are resource places. In this model, one token represents one set of cover seats. The proposed Petri Net model consisted of about 84 places and 64 transitions. Table 7 shows a description of each place.

5.2 Computer tests

The testing was performed with the production orders of three typical days with 31, 35 and 45 jobs. Two shift lengths were set: 16 hours, which is the daily available time, and 13 hours. Additional resources (workers) were added to the manned stations (no additional CNC machines were added). The sewing line was considered as a single resource. The initial configuration was set to one resource to each workstation.

The primary objective of this testing was to compare the actual number of resources at each station with the solution provided by the RAAS algorithm. The scheduling methods (RAAS and the company's) were not compared since: (i) neither the exact schedule nor the performance indicators were available from the records of the company, (ii) the old scheduling method also contemplated the skills and preferences of workers, (iii) rework was not considered in RAAS. The following table shows the results of the computer testing.

Table 8 shows only the number of machines or workers of the stations that require more than one resource.

The above results show that the actual configuration of workers is greater than the suggested by the RAAS algorithm. It is expected as there are many aspects that hinder the execution of the schedule (see Sect. 5.1). These include reprocessing, fabrication of spare components and loss of worker productivity due to external factors. As in the case of the actual plant, the critical resources are the cutting workers in terms of capacity and throughput.

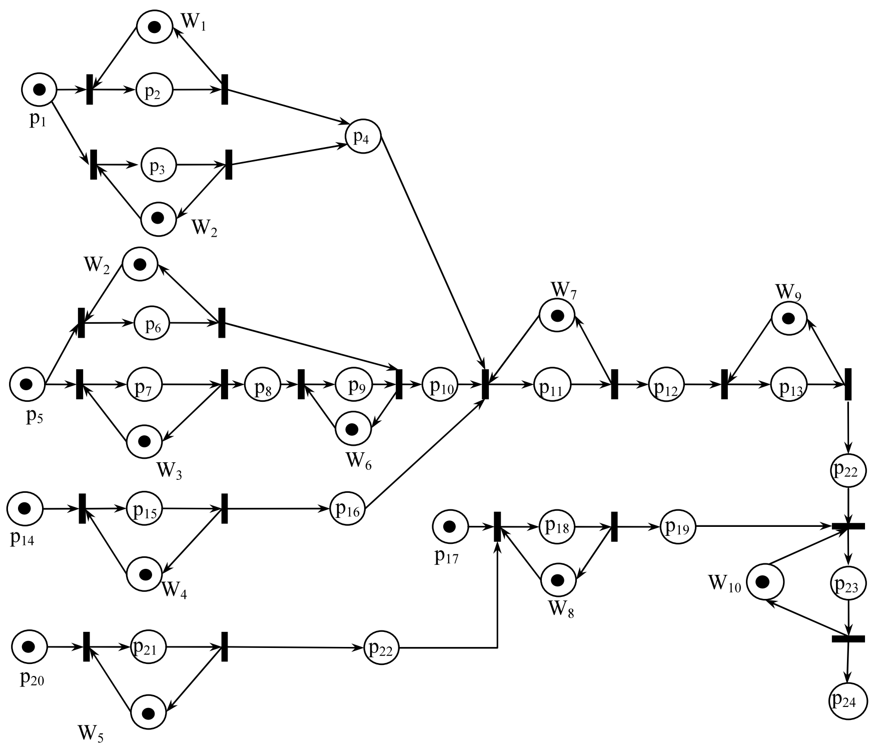

Fig. 4 Petri Net model of the routing sheet of a standard cover seat

Table 7 Description of places for the Petri Net in Fig. 2

p1	Leather sheets ready	p18	Setup operation
p2	Leather sheet CNC cutting	p19	Carpet sheets ready
p3	Leather sheet manual cutting	p20	Carpet cutting
p4	Leather components cut	p21	Carpet parts ready
p5	Vinyl sheets ready	p22	Leather and vinyl parts ready
p6	Vinyl sheet machine cutting	p23	Final assembly (sewing)
p7	Vinyl components cut	p24	Seat cover ready
p8	Vinyl punching	W1	CNC station available
p9	Vinyl components ready	W2	Leather cutting workers available
p10	Leather parts ready for punching	W3	Vinyl cutting station available
p11	Leather punching	W4	Leather punching station available
p12	Leather components punched	W5	Carpet cutting station available
p13	Foam pad attachment	W6	Vinyl punching station available
p14	First sub-assembly ready for sewing	W7	First assembly station available
p15	Frame components in stock	W8	Setup worker available
p16	Frame processing (cutting)	W9	Framing worker available
p17	Frames and carpet components ready	W10	Sewing station available

Table 8 Results for the case of study

Problems (number of jobs)	Shift span (min)	Total number of resources: Workstation	Total number of resources: Number of resources	$C_{\max}$ (min)
31	960	W2	4	960
		W7	2	
		W10	3	
31	780	W2	11	780
		W7	4	
		W9	5	
35	960	W2	5	960
		W7	3	
		W9	2	
		W10	3	
35	780	W2	13	780
		W5	2	
		W7	3	
		W9	3	
		W10	5	
45	960	W2	8	960
		W7	3	
		W9	2	
		W10	5	
45	780	W2	18	780
		W5	2	
		W7	5	
		W9	2	
		W10	7	

The second most critical resources are the sewing lines and the third is the first assembly station.

Another conclusion is that as the number of jobs increases the number of cutters increases more rapidly than the number of sewing lines. This figure increased from 13 to 18 cutters when the number of jobs changed from 35 to 45 in the 13 hour (780 min) shift. In the same situation, the number of sewing lines changed from 5 to 7. This result has implications in the actual management of resources. Although the sewing line seems less critical, notice that a sewing line employs 5 workers. Another issue is that finding additional capacity for this resource is not easy due to several reasons that include lack of satellite companies ("maquilas") and both lack of sewing machines and trained workers. On the other hand, according to the plant supervisor, finding additional trained cutting workers is not as difficult, but there is a limitation on the number of available cutting tables (seven at the time of writing).

Summarizing, the proposed RAAS method can be indeed a tool for workforce planning on real settings. The analysis, however, carried out for this manufacturing shop leads to different conclusions as compared to the problems from the literature. Clearly here, adding resources to stations imply additional costs, worker training, outsourcing and space and machine limitations that do not exist on the literature instances. The final word on these issues is on the plant management.

6 Conclusions

This paper has presented a Petri Net based methodology for resource assignment and scheduling. The methodology was inspired from a real problem in a manufacturing plant of car seat covers. Petri Nets become ideal for the studied problem since they can capture the essence of the interactions between workers and machines, complex precedence relationships, and concurrent activities. Usually traditional resource assignment approaches only take into account only time windows and worker skills constraints. Accurate resource assignment must also include job precedence relationships and resource—job interactions.

This paper shows the validity of Petri Nets for resource assignment and scheduling. The results on the classical instances suggest that the methodology used to both identify the critical resources and schedule jobs was indeed effective. The algorithm outperformed the well-known General Shifting Bottleneck (GSB) and the FJSJGA in most of the studied cases from the literature.

The methodology also showed its value on a real system. Here we can affirm that (i) Petri Nets are capable to model real system with an adequate detail, (ii) the RAAS algorithm can be effectively applied to a real-life setting and (iii) the further research will be focused on issues of online scheduling for complex systems such as order arrival and cancellation, changes in the processing times and machine breakdowns.

This project is in a prototype stage. The development of the scheduling system and its implementation at this system has several issues to be resolved. A full scale scheduling platform still requires has many issues: Database management, security and permissions, job status handling, training and adapting and changing the schedule to name a few. The latter seems to be very important for the plant schedulers: Usually the computer generated schedule requires manual changes to incorporate tacit rules. Such rules are very difficult and impractical to implement in any system. At the time of writing, the scheduling system provides only an off-line schedule at the cutting stages and all schedule changes are done manually.

Acknowledgements This paper is part of the project N° 33016 "*Redes de Petri para modelado y programación de producción aplicados al contexto colombiano*" ("*Petri Nets for modeling and scheduling applied to the Colombian context*") funded by the CIFI (Centro de Investigación de la Facultad de Ingeniería) at the Universidad de los Andes (Colombia).

References

Abdallah, I. B., Elmaraghy, H. A., & Elmekkawy, T. Y. (2002). Deadlock-free scheduling in flexible manufacturing systems using Petri-Nets. *International Journal of Production Research*, *40*(12), 2733–2756.

Adams, J., Balas, E., & Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management Science*, *34*, 391–401.

Applegate, D., & Cook, W. (1991). A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, *3*(2), 149–156.

Bachman, A., Cheng, T., & Janiak, A. (2002). Scheduling start time dependent jobs to minimize the total weighted completion time. *Journal of the Operational Research Society*, *53*, 688–693.

Gutiérrez, E., & Mejía, G. (2009). Scheduling complex manufacturing systems using a genetic algorithm. In *20th international conference on production research*, Shanghai, China, 2–6 August 2009. In CD-ROM.

Kim, Y., Tatsuya, T., & Tatsuo, N. (2007). FMS scheduling based on timed Petri Net model and reactive graph search. *Applied Mathematical Modelling*, *31*(6), 955–970.

Lawrence, S. (1984). *Resource constraint scheduling: an experimental investigation of heuristic scheduling techniques*. GSIA, Carnegie Mellon University.

Lee, D., & DiCesare, F. (1994). Scheduling flexible manufacturing systems using Petri Nets and heuristic search. *IEEE Transactions on Robotics and Automation*, *10*(2), 123–131.

Low, C., & Wu, T.H. (2001). Mathematical modeling and heuristic approaches to operation scheduling problems in an FMS environment. *International Journal of Production Research*, *39*(4), 689–708.

Meisels, A. & Lusternik, N. (1997). Experiments on networks of employee timetabling problems. In *LNCS: Vol. 1408*. *Pract. theo. autom. timetab. II*, Toronto, Canada (pp. 130–141). Berlin: Springer.

Mejía, G. E., & Montoya, C. E. (2007). Resource assignment and scheduling using Petri Nets and heuristic search. In *Proceedings of the ICPR 19 (International conference on production research)*, Valparaíso, Chile.

Mejía, G., & Montoya, C. (2008). A Petri Net algorithm for minimizing total tardiness in flexible manufacturing systems. *Annals of Operations Research*, *164*, 63–78.

Mejía, G., & Montoya, C. (2009). Scheduling manufacturing systems with blocking: a Petri Net approach. *International Journal of Production Research*, *47*(22), 6261–6277.

Mejía, G., & Odrey, N. (2005). An approach using Petri Nets and improved heuristic search for manufacturing system scheduling. *Journal of Manufacturing Systems*, *24*(2), 79–92.

Murata, T. (1989). Petri Nets: properties, analysis and applications. *Proceedings of the IEEE*, *77*(4), 541–580.

Nhu, B., Ho, J., Cing, T., & Edmund, M. (2007). An effective architecture for learning and evolving flexible job-shop schedules. *European Journal of Operational Research*, *179*(2), 316–333.

Pinedo, M., & Chao, X. (1999). *Operations scheduling with applications in manufacturing and services*. New York: McGraw-Hill.

Reyes-Moro, A., Yu, H., Kelleher, G., & Lloyd, S. (2002). Integrating Petri Nets and hybrid heuristic search for the scheduling of FMS. *Computers in Industry*, *47*, 123–138.

Rodammer, F., & Whit, K. (1988). A recent survey of production scheduling. *IEEE Transactions on Systems, Man, and Cybernetics*, *18*(6), 841–851.

Xiong, H., & Zhou, M. (1998). Scheduling of semi-conductor test facility via Petri Nets and hybrid heuristic search. *IEEE Transactions on Semiconductor Manufacturing*, *11*(3), 384–393.

Yu, H., Reyes, A., Cang, K., & Lloyd, S. (2003). Combined Petri Net modelling and AI based heuristic hybrid search for flexible manufacturing systems—part 1, Petri Net modelling and heuristic search. *Computers and Industrial Engineering*, *44*, 527–543.

Zhang, X., & Bard, J.F. (2006). A multi-period machine assignment problem. *European Journal of Operational Research*, *170*(2), 398–415.