Branch and Bound-Based Scheduling of Tasks on Unrelated Parallel Multiprocessor Systems Using Petri Nets

ABOLFAZL JALILVAND[1], SOHRAB KHANMOHAMMADI[1], FEREIDOON SHABANI NIA[2]
[1]Faculty of Electrical and Computer Engineering
University of Tabriz
[2]Faculty of Electrical and Electronics Engineering
Shiraz University
IRAN

Abstract: This paper deals with scheduling n jobs in an m-machine job-shop environment to minimizes the maximum overall completion time of all jobs called make-span. In this multiprocessor scheduling problem we assume that the jobs are available at time zero and have no sequence- dependent setup times on machines. For solving the scheduling problem we develop a new Branch and bound algorithm which constructs its search tree gradually and doesn't need large size of memory. We will propose a lower bound cost for Branch and bound method. Furthermore for initializing the root node in the search tree a heuristic upper bound cost will be introduced which reduces the branch-and-bound computations. For applying the resultant optimum schedule on the manufacturing system a supervisor Petri net is introduced. The proposed methods will be verified through a computational experiment.

Key-Words: Manufacturing systems, Petri nets, Branch and Bound, Task scheduling, Make-span.

1 Introduction

The problem of assigning a given collection of tasks to a given set of processors with a minimum make-span is commonly referred to as the multiprocessor scheduling problem [1]. The search for a solution to the multiprocessor scheduling problem is performed with the aid of a *search tree* that represents the solution space of the problem, that is, all possible permutations of task–to–processor assignments and schedule orderings. The branch-and-bound (B&B) strategy has been successfully used for searching the solution space and finding optimal or near-optimal solutions to the problem of scheduling tasks on multiprocessor architectures [2-4].

The solution space is often represented by a search tree where each vertex in the search tree represents a specific task–to–processor assignment and schedule ordering, and one or many vertices represent the optimal solution whenever one exists. The *root vertex* of the search tree represents an empty schedule and each of its descendant vertices (children) represents the scheduling of one specific task on one specific processor. The children of each of these child vertices represent the scheduling of yet another task on one processor [2].

Scheduling jobs on unrelated parallel machines is a common problem, where a set of machines is available to process the jobs simultaneously. In the general case of unrelated machines, the processing time for a job on a machine is independent of its processing time on other machines. In a manufacturing plant, for example, many machines may be able to process a job but with different speeds depending on the machine's technology. In such a system assuming that each machine can operate on each of jobs, we encounter m^n distinct leaves (goal nodes) in search tree. Constructing this tree thoroughly needs $\sum_{k=1}^{n} m^k$ memory cells, so that even for low number of jobs and machines, it needs a large size of memory which causes run time to decrease in computer-based application of B&B algorithm.

In this paper we introduce a new method, which enables us to construct the search tree gradually, to solve this problem. Using this new algorithm, vertices are generated just when the B&B algorithm needs to explore them. A standard approach for evaluating effectiveness of a heuristic solution is to compare the value of the heuristic solution with a lower bound on the optimal solution [5]. For this reason, we establish a lower bound on the make-span. Furthermore for initializing the root node in the search tree a heuristic upper bound cost will be

introduced which reduces the branch-and-bound computations. For applying the resultant optimum schedule on the manufacturing system a supervisor Petri net is used.

2 The Multiprocessor System

The multiprocessor system consists of a set of processors (manufacturing cells) indexed by $i \in I = \{1,2,...,m\}$ and a set of jobs, indexed by $j \in J = \{1,2,...,n\}$ where each of jobs can be processed by each of the machines. It is assumed that the required processing time of each job is different for each of machines. Furthermore it is assumed that there are no sequence- dependent setup times on machines. In other word, the main problem is to determine that which job must be processed by any machine and once a set of jobs is assigned to a machine, it is not necessary to find the job sequence on that machine. We can summarize all of these processing times in a $n \times m$ time matrix as follows:

$$T = [t_{ji}]_{n \times m} \# \quad (1)$$

Where t_{ji} represents the processing time of j^{th} job i^{th} machine. If a machine isn't eligible to process a specific job it can be considered by setting the corresponding processing time in T to infinity. We introduce C_i as the completion time of the last job scheduled on i^{th} machine which can be defined as follows:

$$C_i = \sum_{j=1}^{n_i} t_{ji} \# \quad (2)$$

Where n_i is number of jobs scheduled on machine i and t_{ji} is the processing time of job j on machine i. By using C_i, the make-span ($C_{\max}$) can be defined as:

$$C_{\max} = \max_{i \in I}(C_i) \# \quad (3)$$

Where the objective of scheduling is to minimize the make-span.

3 Branch and Bound Algorithm

Branch and Bound is a common search technique for combinatorial optimization. B&B improves over exhaustive enumeration, because it avoids the exploration of those regions of the solution space, where it can be certified (by means of lower bounds) that no solution improvement can be found. B&B constructs a solution of a combinatorial optimization problem by successive partitioning of the solution space. The *branch* refers to this partitioning process (generating the child vertices); the *bound* refers to lower bounds that are used to construct a proof of optimality without exhaustive search (process of evaluating the cost of new child vertices). The exploration of the solution space can be represented by a search tree, where its nodes represent sets of solutions, which can be further partitioned in mutually exclusive sets. Each subset in the partition is represented by a child of the original node. Whenever a new vertex is generated which could lead to an optimal solution, it will be referred to as an *active vertex*. The power of the B&B strategy lies in alternating branching and bounding operations on the set of active vertices.

A *goal vertex* in the search tree represents a complete solution where all tasks have been scheduled on the processors. An "acceptable" complete solution is also called a *feasible* solution. An *intermediate vertex* represents a partially complete schedule. The *level* of a vertex is the number of tasks that have been assigned to any processor in the current schedule. The *cost* of a vertex is the quality of the schedule represented by the vertex.

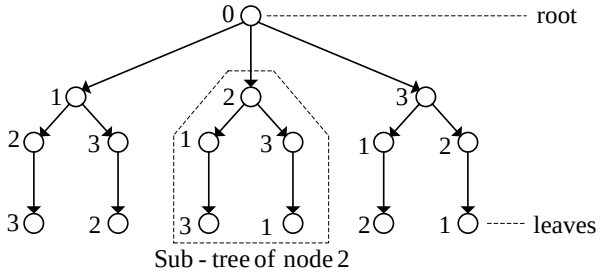

Fig.1: The structure of a tree.

The B&B algorithm starts by sequencing one of the available operations called branching the node. By branching a node, a new node is formed and the node is kept in the search space if its lower bound value is better than the upper bound value or vice versa. A heuristic is used to schedule the remaining operations for every node and the best solution found so far will be recorded as the upper bound value. The next node to be branched is the one with the best lower bound value, as it is deemed to have the best potential. As more nodes are branched, more and more operations will be sequenced, and the upper bound value will become smaller and smaller [6].

An algorithm that computes a lower bound on the cost of any solution in a given subset prevents further searches from a given node if the best cost found so far is smaller than the cost of the best solution that can be obtained from the node (lower bound computed at the node). In this case the node is killed and no children need to be searched; otherwise it is alive.

3.1 Upper-Bound Cost

It is well known that B&B computations can be reduced by using a heuristic to find a good solution to act as an upper bound prior to the application of the enumeration algorithm, as well as at certain nodes of the search tree [3]. An upper-bound cost (UBC) is used to initialize the root vertex. The more accurate the upper-bound cost is, the faster the B&B algorithm will get because more vertices can be pruned at each step. For the mentioned scheduling problem we introduce a heuristic UBC as follows:

Consider the jobs one by one from 1 to n. Related to each job, after adding the make-spans of scheduled machines to the corresponding processing times related the new job we select the machine with minimum overall time and consider its index as the machine which the corresponding job is assigned to it. When this process is completed we have a sequencing of machines related to jobs. Now we calculate the completion time of each machine and based on them we compute $C_{\max}$ of this sequencing as a UBC using (3).

3.2 Lower-Bound Cost

In this section we show that a lower bound cost (LBC) can be found as follows:

$$\text{LBC} = \frac{1}{m}\sum_{j=1}^{n}(\min_{i\in I} t_{ji})\ \# \qquad (4)$$

For proofing this relation as it was explained the make-span of each sequence is $C_{\max} = \max_{i\in I}\{C_i\}$ where C_i is defined by formulation (2). It can be shown that:

$$\frac{1}{m}\sum_{i=1}^{m} C_i \le C_{\max}\ \# \qquad (5)$$

Substituting (2) in (5) we have:

$$\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n_i} t_{ji} \le C_{\max}\ \# \qquad (6)$$

Considering that n_i is the number of jobs scheduled on machine i and based on that the total number of scheduled jobs on machines is $\sum_{i=1}^{m} n_i = n$ we can rewrite equation (6) as follows:

$$\frac{1}{m}\sum_{j=1}^{n} t_{ji} \le C_{\max}\ \# \qquad (7)$$

To minimize the make-span we must choose an optimum schedule to achieve $\min\{C_{\max}\}$. Assuming that $t^*_{ji}: j = 1,2,...,n$ is the processing time of the optimal sequencing we have:

$$\frac{1}{m}\sum_{j=1}^{n} t^*_{ji} \le \min\{C_{\max}\}\ \# \qquad (8)$$

$$\sum_{j=1}^{n}\min_{i\in I}\{t_{ji}\} \le \sum_{j=1}^{n} t^*_{ji}\ \# \qquad (9)$$

Using (8) and (9) we have:

$$\text{LBC} = \frac{1}{m}\sum_{j=1}^{n}\min_{i\in I}\{t_{ji}\} \le \min\{C_{\max}\}\ \# \qquad (10)$$

This indicates the truth of (4).

4 The New Branch and Bound Algorithm

In this section we introduce a new B&B algorithm based on a new method which constructs the search tree step by step. For solving the problem it is required to consider all possible combinations for assigning n distinct jobs to m individual machines. In our new approach, instead of constructing the whole of the search tree for mentioned multiprocessor system which needs $\sum_{k=1}^{n} m^k$ memory cells, we just need a $n\times 1$ vector $\boldsymbol{D}$ which reduces this high size of memory to only n memory cells. The key idea in proposed B&B algorithm is to produce the nodes of this search tree one by one just when the B&B algorithm needs to explore it. First we introduce an algorithm that can produce each of the possible combinations of m machines and the n jobs assigned to them, one by one.

Algorithm I:

1- Get n as the number of jobs and m as the number of machines.
2- Set $\boldsymbol{D} = [1\ \ 1\ \ \dots\ \ 1]^T$ as the initial value of $\boldsymbol{D}$ and set $f = n$ as a temporary flag.
3- Get $\boldsymbol{D}$ as the new sequence of machines which can be assigned jobs respectively.
4- Set $\boldsymbol{D}(f) = \boldsymbol{D}(f) + 1$.
5- IF $\boldsymbol{D}(f) > m$ and $f > 1$ THEN set $\boldsymbol{D}(f) = 1$ and $f = f - 1$ then go to step 4 ELSE go to step 6.
6- IF $\boldsymbol{D}(f) \le m$ THEN go to step 3 ELSE go to step 7.
7- End.

As it can be seen in this algorithm we only needs a $n\times 1$ vector and this decreases the memory size considerably. Now based on algorithm I we introduce a B&B algorithm to solve the mentioned scheduling problem.

Algorithm II:

1- Get n as the number of jobs, m as the number of machines and T as the matrix which includes machines processing times on jobs.
2- Set $\boldsymbol{D}=[1\ \ 1\ \ \ldots\ \ 1]^T$ as the initial value of $\boldsymbol{D}$ and set $f=n$ as a temporary flag.
3- Get $\boldsymbol{D}$ as the sequence of machines which are assigned jobs respectively and Calculate the make-span ($C_{\max}$) of machines based on formula (3) then set MC (minimum make-span) equal to $C_{\max}$.
4- Get $\boldsymbol{D}(1{:}\,f)$ as the sequence of machines which are assigned jobs respectively and calculate the make-span ($C_{\max}$) of machines.
5- IF $C_{\max} \geq MC$ THEN go to step 7 ELSE go to step 6.
6- IF $f=n$ THEN set $MC=C_{\max}$ and go to step 7 ELSE set $f=f+1$ and go to step 4.
7- Set $\boldsymbol{D}(f)=\boldsymbol{D}(f)+1$.
8- IF $\boldsymbol{D}(f)>m$ and $f>1$ THEN set $\boldsymbol{D}(f)=1$ and $f=f-1$ then go to step 7 ELSE go to step 9.
9- IF $\boldsymbol{D}(f)\leq m$ THEN go to step 4 ELSE go to step 10.
10- End.

This B&B method performs a depth first search in an exhaustive manner. In this algorithm the value of f in every stage indicates the level of the search tree. When the f^{th} element of vector $\boldsymbol{D}$ exceeds m, it denotes that the branching must be restart from the previous node. This process will be done by resetting the f^{th} element of $\boldsymbol{D}$ to 1 and decrementing f by 1. Creating a new node at the level f, the make-span of the path from root to this new node will be tested and if it isn't less than the current LBC that node is killed by incrementing the corresponding element at vector $\boldsymbol{D}$. Otherwise that node will be remained as a activate node and the branching will be continued by incrementing f and creating a new node at the next level of the search tree. The branching may be continued until it reaches a leaf where $f=n$. In this algorithm each node is constructed when it must be tested and there isn't need to construct the whole search tree at once.

Because their powerful modeling capability for a variety of systems, Petri Nets have been chosen as the modeling tool for modeling, in this paper. Next sections deal with Petri nets and using them for modeling the manufacturing systems [9]. Also to apply the optimum sequence obtained trough B&B algorithm, we introduce a supervisor Petri net.

5 Petri Nets

A Petri Net (PN) is a 5-tuple, $PN=(P,T,F,W,M_0)$ where [10]:

$P=\{p_1\ \ p_2\ \ \ldots\ \ p_m\}$ is a finite set of places.
$T=\{t_1\ \ t_2\ \ \ldots\ \ t_n\}$ is a finite set of transitions.
$F\subseteq(P\times T)\cup(T\times P)$ is a set of arcs (flow relations)
$W: F\rightarrow\{1\ \ 2\ \ 3\ \ \ldots\}$ is a weight function.
$M_0: P\rightarrow\{0\ \ 1\ \ 2\ \ \ldots\}$ is the initial marking .
$P\cap T=\Phi$ and $P\cup T\neq\Phi$.

A Petri net structure $N=(P,T,F,W)$ without any specific initial marking is denoted by N, and a Petri net with the given initial marking is denoted by (N,M_0).

The dynamical behavior of a system is modeled by changing the state or marking in Petri nets according to the following (firing) rules:

1- A transition *t* is said to be enabled if each input place *p* of *t* is marked with at least $w(p,t)$ tokens, where $w(p,t)$ is the weight of the arc from *p* to *t*.
2- An enabled transition may or may not fire depending on whether or not the event actually takes place (firing conditions are ok).
3- Firing of an enabled transition *t* removes $w(p,t)$ tokens from each input place *p* to *t* and adds $w(t,p)$ tokens to each output place *p* of *t*, where $w(p,t)$ and $w(t,p)$ are the weights of the arcs from *p* to *t* or *t* to *p* respectively.

In graphical representation of a Petri net, places are represented by circles and transitions are shown by hollow bars. The relationship between places and transitions is represented by directed arcs. Also each Petri net can be denoted by two input and output matrices related to transitions of Petri net. For example The Petri net of Fig. 2 depicts the firing of a transition.

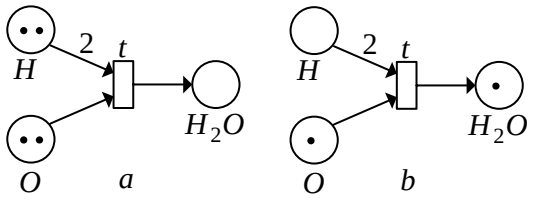

Fig.2: Transition (firing): (a) Marking before firing (b): Marking after firing.

In un-timed Petri net one can prohibit controlled transition from firing but cannot force the firing of a transition at a particular time. In a timed Petri net controlled transitions are forced to fire, by considering the time dependent firing functions. In timed Petri nets, each transition has its specific time

which determines the transition's holding time. When a transition is fired during its holding time the network's marking is not changed and as soon as its holding time elapsed the marking of network will be changed based on the former mentioned firing rules [11-12].

6 Applying B&B Using Petri Nets

For modeling the mentioned multiprocessor job-shop system it is considered that each machine has an input buffer and an output buffer as shown in Fig. 3. In this work to simplify the model the wait times are included in machining times.

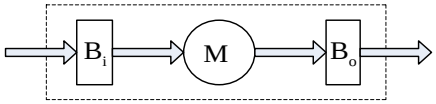

Fig. 3: Schematic diagram of a machine

The Petri net model of each machine is designed regarding its input and output buffers as shown in Fig. 4.

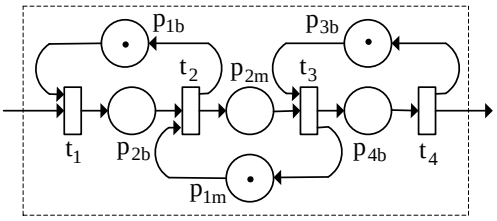

Fig.4: Petri net model of the machine.

In this model the places are defined as:

p_{1b}: the input buffer is empty.
p_{2b}: the input buffer is full.
p_{3b}: the output buffer is empty.
p_{4b}: the output buffer is full.
p_{1m}: the machine is idle.
p_{2m}: the machine is busy.

Also each transition in Fig. 4 is defined as:

t_1: A part enters the input buffer.
t_2: A part enters the machine.
t_3: The part exits the machine.
t_4: The part leaves the output buffer.

Based on this Petri net model we can model the multiprocessor system by considering *m* identical models which are parallel together. For applying the optimum sequencing obtained by B&B algorithm we introduce a Petri Net model. Fig. 5 shows this Petri net model. In this model we consider a place ($p_j : j = 1,2,...,n$) to each job. Regarding that each job can be assigned to each of the machines hence we consider *m* output arcs from each of these places (one output arc is related to one machine). To apply the optimum sequence it is needed to determine which job will be assigned to which machine. For this purpose we consider *m* additional places $p_{ji} : i = 1,2,...,m$ related to each job j (totally $n \times m$ additional places). We can apply the desired sequencing by proper putting one token in one place of each of these n sets of places.

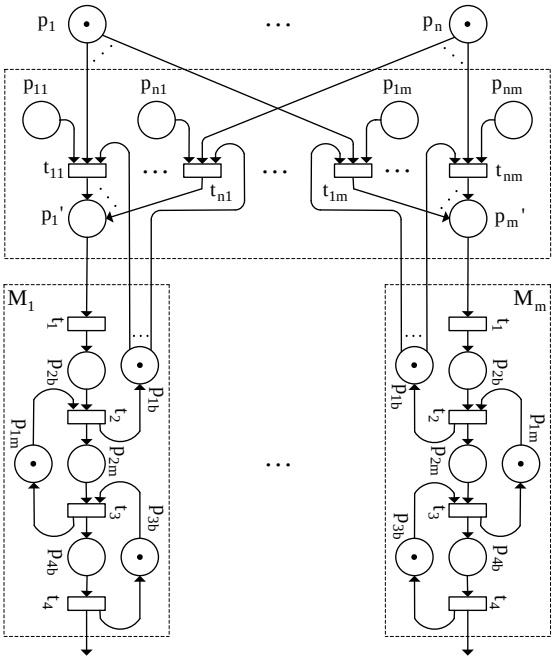

Fig. 5: Petri net-based application of the desired sequencing.

In this Petri net model we can assign the processing time of each job by each machine to corresponding transition from the set of transitions labeled: $t_{ji} : \begin{cases} j = 1,2,...,n \\ i = 1,2,...,m \end{cases}$. In such a case regarding the processing times related to t_{ji} other transitions will be un-timed.

7 Computational Results

In order to evaluate performance of the heuristic approaches developed in this paper, a computational experiment was conducted. The results are provided in Table 1. In this computational experiment, the minimum make-spans of the schedules obtained by the B&B were compared with the lower bound cost (LBC) and upper bound cost (UBC) by using $(\frac{\text{UBC} - \text{LBC}}{\text{MC}}) \times 100\%$. For each testing problem, instances were generated and for each instance, the matrix of machining times was randomly generated. In this table it can be seen that maximum value of

$(\frac{UBC - LBC}{MC}) \times 100\%$ is less than 70. Furthermore it can be seen that in each case the UBC and LBC have acceptable values near to the obtained minimum make-span so that the difference of these two values in worst case is less than 70% of MC.

Table 1: Computational results

n	m	LBC	UBC	MC	$(\frac{UBC-LBC}{MC}) \times 100\%$
8	2	142	171	162	17.90
8	4	65.5	103	87	43.10
8	6	36.5	60	52	45.19
8	8	21	45	35	68.57
12	2	235.5	294	248	23.59
12	4	82.75	114	98	31.89
12	6	44.17	77	55	59.69
12	8	38.5	54	52	29.81
16	2	339	370	350	8.85
16	4	94.75	122	105	25.95
16	6	58	92	74	45.94
16	8	36.75	52	49	31.12
20	2	331.5	426	334	28.29
20	4	119.75	153	134	24.81
20	6	78.5	115	94	38.83
20	8	40	69	48	60.42

8 Conclusions

In this paper, multiprocessor scheduling was discussed were n jobs must be scheduled in an m-machine job-shop environment. We assumed no sequence- dependent setup times on machines. For solving the scheduling problem we developed a new Branch and Bound system which constructs its search tree step by step and doesn't need large size of memory. A lower bound limit for Branch and Bound method was introduced. Furthermore for initializing the root node in the search tree a heuristic upper bound was proposed which reduced the branch-and-bound computations. For applying the resultant optimum schedule on the manufacturing system a supervisor Petri net was constructed. Evaluating the proposed methods a computational experiment was done.

References:

[1] S. Fujita, M. Masukawa and S. Tagashira, A Fast Branch-and-Bound Algorithm with an Improved Lower Bound for Solving the Multiprocessor Scheduling Problem, *Proc. of 9th international conference on parallel and distributed systems (ICPADS02)*, 2002, pp. 611-616.

[2] J. Jonsson and K. G. Shin, A Parameterized Branch-and-Bound Strategy for Scheduling Precedence-Constrained Tasks on a Multiprocessor System, *Proc. of International Conference on Parallel Processing*, August 11-15, 1997, pp. 158-165.

[3] X. Wang and J. Xie, Branch and bound algorithm for flexible flow shop with limited machine availability, *Asian Information-Science-Life*, Vol.1, No.3, 2003.

[4] R. Z. Rios Mercado, and J. F. Bard, A Branch and Bound Algorithm for Permutation Flow Shops with Sequence Dependent Setup Times, *IIE Transactions*, 1999(31), pp. 721-731.

[5] D. He, A. Babayan, A. Kusiak, Scheduling manufacturing systems in an agile environment, *Robotic and Computer Integrated Manufacturing*, Vol. 17, No. 1-2, 2001, pp. 87-67.

[6] P. Y. Gan, K.S. Lee and Y. F. Zhang, A Branch and Bound algorithm based process planning system for plastic injection mould bases, *the international Journal of Advanced Manufacturing System*, Vol. 18, 2001, pp. 624-632.

[7] S. Olaffson, and L. Shi, A method for scheduling in parallel manufacturing systems with flexible resources, *IIE Transactions*, 2000(32), pp. 135-146.

[8] S. Khanmohammadi, Single array Branch and Bound method, *Iranian Journal of Engineering*, Vol. 3, Nos. 1&2, 1990, pp. 71-72.

[9] A. Jalilvand and S. Khanmohammadi, Modeling of Flexible Manufacturing Systems by Timed Petri Net, *International Conference on Computational Intelligence (ICCI-2004)*, December 17-19, 2004, Istanbul, Turkey, pp. 141-144.

[10] Tado Murata, Petri nets: properties, analysis and application, *Proc. of IEEE*, Vol. 77, No. 4, April 1989, pp. 541-580.

[11] A. Jalilvand and S. Khanmohammadi, Using Petri Nets and Branch and Bound Algorithm for Modeling and Scheduling of a Flexible Manufacturing System, *WSEAS Transaction on Systems*, Issue 7, Vol.3, September 2004, pp. 2580-2585.

[12] A. Jalilvand and S. Khanmohammadi, Task scheduling in Manufacturing Systems Based on an Efficient Branch and Bound Algorithm, *Proc. of IEEE Conference on Robotics, Automation and Mechatronics (RAM2004)*, December 1-3, 2004, Singapore, pp. 271-276.