วิศวกรรมสาร มข. ปีที่ 32 ฉบับที่ 2 (237 - 250) มีนาคม - เมษายน 2548
KKU Engineering Journal Vol. 32 No. 2 (237 - 250) March – April 2005

A Hybrid of List-Scheduling and Column-Generation for Scheduling of n Job-Groups through m Identical Parallel Machines with Minimum Makespan *

Seekharin Sukto[1] and Peerayuth Charnsethikul [2]

[1] Lecturer, Department of Industrial Engineering, Faculty of Engineering, Khon Kaen University, Khon Kaen, 40002
[2] Associate Professor, Department of Industrial Engineering, Faculty of Engineering, Kasetsart University, Bangkok, 10903

Email: Seesuk@kku.ac.th

Abstract

This paper presents an optimization based heuristic for minimizing makespan on scheduling of n job-groups with q_i as the number of identical jobs and p_i as the processing time, included any setup time required, in job-group i through a set of m identical parallel machines. This heuristic, referred as *LSCCG*, is based on *LS* (List-Scheduling) heuristic and *CG* (Column Generation) which are hybrid between the well-known in scheduling problems such as *LPT* (Longest Processing Time) heuristic or *IP* (Integer Programming) heuristic and column generation procedure in cutting stock problem, respectively. The first model, referred as *LPTCCG* which are hybrid between *LPT* heuristic, is constructed the initial pattern and adjusted to a lower bound by *ALPTP* (Adjusted LPT Pattern); while the second method, *CG*, is modified to *CCG* (Continuous Column Generation) attempting to strengthen makespan bound and collectively generated patterns to minimize a number of machines required. This model is repeatedly looped by *LPT*, *ALPTP* and *CCG* to reach an upper bound of minimum makespan. The average performance of this proposed is considerably better than that of *LPT* heuristic in case of n/m ratio less than 2.5. The Better model, referred as *IPCCG*, is based on *IP* (Integer Programming) heuristic by *LINGO 6.0* software and *CG*. As the same, *IP* heuristic is generated the initial pattern, which is interrupted at 200,000 iterations and adjusted to a lower bound by *AIPP* (Adjusted IP Pattern) and then attempted to strengthen makespan bound with a number of available machines by *CCG*. *IPCCG* heuristic is repeatedly looped by *IP* Heuristic, *AIPP* and *CCG* to reach an upper bound of minimum makespan. The average performance of this proposed, *IPCCG* heuristic is considerably better than that of *LPTCCG* heuristic.

Keywords: parallel machines, multi-processor, identical machines, scheduling, bin packing, cutting stock, minimize makespan, completion time, LP-based heuristic, IP-based heuristic, column generation, longest processing time.

* Original manucript submitted: June 22, 2004 and Final manucript received: August 30, 2004

Introduction

Scheduling problem has developed over some thirty-five years. In 1968 was B.C. (Before Complexity), most certainly, this condition greatly influenced the literature on scheduling as well as the way scheduling subject was perceived. Scheduling involved the optimal allocation or assignment of resources, overtime, to a set of tasks or activities or jobs, where a *"job"* consists of one or more activities and a *"machine"* is a resource that can perform at most one activity at a time. Scheduling problems include important problems in the obvious setting of manufacturing, transportation and logistics as well as in fields such as communications, media management and sports. Moreover, effective scheduling solutions accordingly can produce substantial economic dividends (Parker, 1995).

Parallel machines scheduling problems require the construction of machine processing schedules for a given set of jobs to optimize a given measure of performance. For the example of this problem, each job consists of only one activity include the scheduling of computer systems on one hand and the scheduling of bottle- neck workstations involving multiple machines on the other. The parallel machines scheduling problem is also an important generalization of the single machine problem and can be sub-problem in many complex multiple machine problems. The parallel machines may be identical, as each machine requires the same amount of time to process a given set of jobs. If parallel machines have different ages of technology, these machines may require different amounts of time to process a given set of jobs (Gupta and Ruiz-Torres, 2001).
Coffman et al. (1978) offered an application of bin packing to multiprocessor scheduling problems. Therefore, bin packing problems had a natural dual-like relationship with multiprocessor or parallel machines scheduling problems. Dyckhoff (1990) developed a consistent and systematic approach for a comprehensive typology integrating the various kinds of cutting and packing (C&P) problems. Because of the dominant role played by the patterns and their nature as geometric combinations one may say that C&P problems belong to the field of "geometric combinatorics". In a narrow sense, C&P problems are concerned with object and items defined by one, two or all three spatial dimensions of space. In the case of cutting stock problems the large object are given by solid materials cut up into small items as pieces. Since trim loss optimization is a main objective one also calls of trim loss problems. For the case of packing problems in the narrow sense are characterized by large objects defined as the empty, useful space. Packing small material items or chips into these objects may also be looked at as cutting the empty space of the large objects into parts of empty spaces some of which are occupied by small items, the other being *"trim loss"*. Conversely, cutting stock problems may be looked at as packing the space occupied by small items into the space occupied by large objects. C&P problems can also be considered in an abstract, generalized sense taking place in no-spatial dimensions. Type of C&P problems is shortly denoted by a fourth-field of respective symbols $\alpha/\beta/\gamma/\delta$ where denotes dimensionality, kind of assignment, assortment of large objects and assortment of small items, respectively. In the case of parallel machines scheduling problem for the time of job dimension belongs to type of C&P problems as *1/V/1/M*, a natural dual like relationship between the C&P problems.

The First Known formulation of a cutting Stock problem was given in 1939 by the Russian economist Kantorovich (1960). The most significant advance in solving this problem was the seminal work of Gilmore and Gomory (1961, 1963) and designated as the standard problem of the one-dimensional cutting stock optimization.

Problem	Parallel machines scheduling	Bin packing	Cutting stock
Classification	$P//C_{max}$	*1/V/1/M*	*1/V/1/R*
Large object	Machines	Bins	Materials
Small object	Jobs	Chips	Pieces
Performance	C_{max} (Makespan)	Capacity	Length of Material

Table 1 *The relationship of parallel machines scheduling, bin packing and cutting stock problem*

Dyckhoff (1981), each possible cutting pattern is identified by one variable. Dropping integrality conditions leads to linear programming (LP) problems characterized by an enormous number of potential variables and a small number of constraints. In order to solve these LP Problems, Gilmore and Gomory have developed a special technique similar to the *"column generation procedures"* known from the Dantzig-Wolfe decomposition principle. Considering knapsack problems at each stage of the revised simplex method one obtains improved cutting patterns, furthermore, only a few columns need to be generated. Heuristic procedure is the priori generation of a small number of *"good"* cutting patterns. Thus, the corresponding linear programming can be solved by standard LP algorithms.

A common measure of performance used in scheduling theory related to shop efficiency and utilization is the maximum completion time or *"makespan"*. This is defined as the elapsed time in which a given set of jobs completes processing in a given set of machines. In Practical term, a schedule that minimizes the makespan also minimizes the machine time is operating, which could relate to minimizing support cost and maximizing the use of resources. The minimization of the makespan could also have practical use in the scheduling of batches where a *"batch"* is defined as a set of jobs. Creating a schedule that minimizes the makespan for the set of jobs in a batch will get the next batch start sooner, resulting in higher efficiency and resource utilization. In the parallel machines scheduling problem, minimization of makespan also identifies a bottleneck machine requirement maximum attention of the machine management.

This paper considers the identical parallel machines scheduling problem to minimize makespan and proposes a new heuristic that is based on *LS* (List-Scheduling) heuristic and *CG* (Column Generation) which are applied to solve this makespan case problem. This heuristic, called *LSCCG* (List-Scheduling and Continuous Column Generation), is applied to two models. The first model, referred as *LPTCCG* which are hybrid between *LPT* (Longest Processing Time) heuristic (Graham, 1969), is constructed the initial pattern of scheduling then adjusted to a lower bound by *ALPTP* (Adjusted LPT Pattern) and *CCG* (Continuous Column Generation), attempted to strengthen makespan bound and collectively generated patterns to minimize a number of machines required (Gilmore and Gomory, 1961, 1963). This model is repeatedly looped by *LPT, ALPTP* and *CCG* to reach an upper bound of minimum makespan. The average performance of this proposed is considerably better than that of *LPT* heuristic in case of *n/m* ratio less than 2.5 (Sukto and Charnsethikul, 2002a). The second and better model, referred as *IPCCG*, is based on *IP* (Integer Programming) heuristic by *LINGO 6.0* software and *CCG*. As the same, *IP* heuristic is generated the initial pattern, which is interrupted at 200,000 iterations and adjusted to a lower bound by *AIPP* (Adjusted IP Pattern) then attempted to strengthen makespan bound with a number of available machines by *CCG*. *IPCCG* heuristic is repeatedly looped by *IP* Heuristic, *AIPP* and *CCG* to reach an upper bound of minimum makespan (Sukto and Charnsethikul, 2002b). Therefore, the average performance of this proposed, *IPCCG* heuristic is considerably better than that of *LPTCCG* heuristic.

1. Problem formulation and fundamental algorithms

Specifically, we consider the following scheduling problem; i = {1, 2, 3,..., n} of n simultaneously available types of job is to be processed by m identical parallel machines. Each job requires one operation which can be processed by one of the m machines. The processing time, included any setup time required, and number of jobs in each type i are p_i and q_i, respectively. A job once started is processed to completion without interruption. There is no precedence relationship between the jobs and each machine can process only one job at a time. For each machine j = {1, 2, 3,..., m}, let S_j be the subset of jobs assigned to machine j; $S_j = \{s_{ij}\}$, with completion time $C(S_j)$ (1). For m machines, let C be the amount of machine time (2). The makespan, $C_{max}(S)$ (3) and the lower bound of makespan, C^*_{LB} (4), of this schedule is calculated as follows:

$$C(S_j) = \sum_{i \in Sj} p_i s_{ij} \tag{1}$$

$$C = \sum_{i=1}^{n} p_i q_i \tag{2}$$

$$C_{max}(S) = \underset{1 \le j \le m}{Max}\, C(S_j) = \underset{1 \le j \le m}{Max} \{ \sum_{i \in Sj} p_i s_{ij} \} \tag{3}$$

$$C^*_{LB} = \lceil C/m \rceil = \lceil \{ \sum_{i=1}^{n} p_i q_i / m \rceil ; \text{Integer} \tag{4}$$

Then, the problem considered in this paper is one of finding the subsets $\{S_1, S_2, ..., S_m\}$ of the given n types of job such that the makespan, $C_{max}(S)$ given by equation (3) is minimum. Following the standard third-field notation of scheduling problems, the above identical parallel machines problem to minimize makespan is designated as a $P//C_{max}$ problem where P designates the identical parallel machines and C_{max} denotes the maximum completion time or makespan. The $P//C_{max}$ problem has been shown to be *NP-hard* in strong sense for an arbitrary number of machine m (Garey and Johnson, 1979). In view of the *NP-hard* nature of the above $P//C_{max}$ problem, several polynomial time algorithms have been proposed to find an approximate solution.

1.1 Algorithm LPT: Longest processing time procedure

The well-known LPT rule (Graham, 1969) is a part of a family of algorithms known as List-scheduling algorithms which iteratively assigns a list of jobs to the least loaded machine. The LPT rule has received extensive attention and has been shown to perform very well for the makespan single criteria problem. The LPT algorithm assigns an available job with maximum processing time to the first machine. The steps of this are as follow.

Input: n, m, p_i for i = 1,2,...,n

Step 1. Let $\sigma = (\sigma(1), \sigma(2)..., \sigma(n))$ be the job ordering such that $p_{\sigma(1)} \ge p_{\sigma(2)} \ge ... \ge p_{\sigma(n)}$. Set $S_1 = ... = S_m = \phi$ and set k = 1. Further, set $C(S_1) = ... = C(S_m) = 0$

Step 2. Select machine m_j so that $C(S_j)$ is as small as possible. Schedule job $\sigma(k)$ on machine m_j. Set $C(S_j) = C(S_j) + p_{\sigma(k)}$ and $S_j = (S_j \sigma(k))$ for j = 1, 2, ..., n.

Step 3. If k < n, then set k = k + 1 and return to step 2. Otherwise, stop. The schedule where jobs in S_j are processed on machine j is the heuristic solution of the $P//C_{max}$ problem makespan $C_{max} = \underset{1 \le j \le m}{Max} \{C(S_j)\}$.

The computational time required for LPT algorithm is O(n log n) and the worst-case performance ratio of algorithm is 4/3 – 1/3m (Graham, 1969).

1.2 Algorithm Column Generation: cutting stock problem procedure

Gilmore and Gomory (1961, 1963) presented a linear programming formulation for the cutting stock problem involving column generation and efficient solution of knapsack problems. We simplicity assume that all pieces available for cutting have length L and that $\underset{1\le i\le n}{Max}\ l_i \le L$. A cutting pattern, represented by a vector $(a_1, a_2,\ldots, a_n)$, consists of a_1 pieces of length l_1, a_2 pieces of length l_2, etc. The number of cutting patterns will, in general, be enormous. Let j index is the various cutting patterns and define

a_{ij} denote the number of pieces of length l_i cut by pattern j
x_j denote the number of times of pattern j is used
d_i denote demand for length l_i

The problem may be expressed as a large scale linear integer program

$$Z = \min \sum_{j=1}^{m} c_j x_j \tag{5}$$

$$\text{Subject to} \sum_{j=1}^{m} a_{ij} x_j \ge d_i \tag{6}$$

$$x_j \ge 0, \text{ integer} \tag{7}$$

Where $a_{ij}x_j$ is the number of pieces of length l_i cut using pattern j, $c_j = 1$, and the objective function minimizes total pieces of material for cutting. With high demands, the number of time different patterns are used will generally be large, and rounding up to integer, after solving the linear program (5) – (7) leads to very good solutions. The large number of variables (cutting patterns), however, makes direct solution via the simplex method infeasible for all but the most trivial problem. Column generation generates the coefficient data a_{ij} when needed, and can be thought of as an extension to decomposition. In decomposition, data for any variable correspond to an extreme point or extreme ray of another linear programming. New data is generated by solving this linear programming sub-problem with an appropriate objective function. In column generation the sub-problem need not be a linear programming; in the cutting stock problem the sub-problem is a knapsack problem.

The relative cost coefficient for a non-basic variable x_j is $\bar{c}_j = 1 - \sum_{i=1}^{n} \pi_i a_{ij}$ where π_i are the simplex multipliers from (4). Since (5) – (7) is a minimization problem, we seek the most negative reduced cost, so the k^{th} sub-problem becomes

$$v^k = \underset{1\le j\le n}{Min} [1 - \sum_{i=1}^{n} \pi_i^k a_{ij}] \tag{8}$$

A cutting pattern $a = (a_1, a_2, \ldots, a_n)$ must satisfy $\sum_{i=1}^{n} l_i a_i \le L$ and a_i are nonnegative integer. The sub-problem reduces to determining the coefficients a_{ij} of a new pattern which minimizes (8) or equivalently

$$Z = \max \sum_{i=1}^{n} \pi_i a_i \tag{9}$$

$$\text{Subject to} \quad \sum_{i=1}^{n} l_i a_i \le L \tag{10}$$

$$a_i \ge 0, \text{ integer} \tag{11}$$

Problem of this form is called knapsack problem: suppose there are n objects, the i^{th} object having weight l_i and value π_i, and it is desired to find the most valuable subset of objects whose total weight does not exceed L. We can view L as being the capacity of a knapsack. If the optimal objective value in (9) is v then if $v > 1$, the corresponding pattern vector is formed and enters the basis. Otherwise, if $v \le 1$ the current solution is optimal. The column generation approach to the cutting stock problem is most efficient when an optimal solution is obtained before too many columns have been added to the restricted master problem, and the sub-problems can be easily solved (Golden, 1976).

2. The *LSCCG* heuristic

The proposed *LSCCG* (List-scheduling and Continuous Column Generation) heuristic is *LP*-based heuristic which can be combined by the well-known List-scheduling, identical parallel machines scheduling algorithms, and column generation which is most efficient in cutting stock problem. Continuous column generation is the modification of column generation procedure to increase bound of makespan and collectively generated column. This proposed heuristic loops between List-Scheduling with adjustment and continuous column generation. The initial pattern for column generation is constructed by List-scheduling and adjusted to the lower bound (4). Column generation generates column data or scheduling pattern until sum of product of simplex multipliers by generated column (9) equal to or less than one, $v \le 1$. Then, the patterns will be selected if the number of patterns can be rounded down to be integer and equal to or more than one, $x_j \ge 1$. Further, *LSCCG* repeatedly loops in a part of remaining jobs. Finally, the lower bound of problem will be increased while *LSCCG* pass the stop criteria and then returned to *LSCCG*, repeatedly, until going into stop criteria. This relationship is evidence as following diagram:

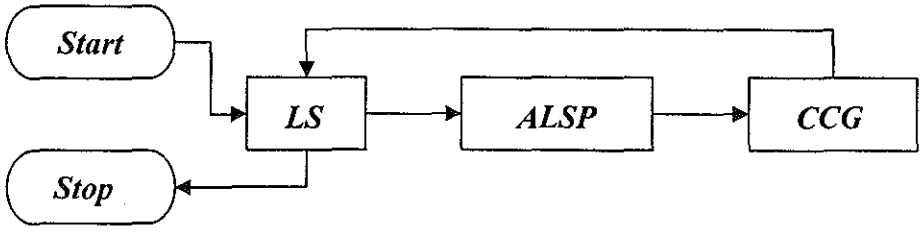

Figure 1 *LSCCG heuristic flow diagram*

2.1 Sub-algorithm LS heuristic: List-scheduling

2.1.1 Sub-algorithm LPT heuristic: Longest Processing Time

Input: n, m, p_i, q_i for i = 1, 2, 3,..., n

Step1. Let $\sigma = (\sigma(1), \sigma(2)..., \sigma(Q))$ be the job ordering, $Q = \sum_{i=1}^{n} q_i$, such that $p_{\sigma(1)} \ge p_{\sigma(2)} \ge ... \ge p_{\sigma(Q)}$. Set $S_1 = ... = S_m = \phi$ and set k = 1. Further, set $C(S_1) = ... = C(S_m) = 0$.

Step2. Select machine m_j so that $C(S_j)$ is as small as possible. Schedule job $\sigma(k)$ on machine m_j. Set $C(S_j) = C(S_j) + p_{\sigma(k)}$ and $S_j = (S_j\sigma(k))$ for j = 1, 2, ..., n.

Step3. If k < Q, then set k = k + 1 and return to step 2. Otherwise, stop. The schedule where jobs

in S_j are processed on machine j is the heuristic solution of the $P//C_{max}$ problem makespan $C_{max} = \underset{1 \le j \le n}{Max} \{C(S_j)\}$.

2.1.2 Sub-algorithm BFD heuristic: Best-fit Decreasing
Input: n, m, p_i, q_i for i = 1, 2, 3,..., n

Step1. Let $\sigma = (\sigma(1), \sigma(2)..., \sigma(Q))$ be the job ordering, $Q = \sum_{i=1}^{n} q_i$, such that $p_{\sigma(1)} \ge p_{\sigma(2)} \ge ... \ge p_{\sigma(Q)}$. Set $C_{LB} = \lceil \sum_{i=1}^{n} p_i/m \rceil$, $S_1 = ... = S_m = \phi$ and set k and j = 1. Further, set $C(S_1) = ... = C(S_m) = 0$.

Step2. Select machine m_j so that $C(S_j)$ is smaller than C_{LB} and go to step 3. Otherwise, set j = j + 1 until j = m.

Step3. Schedule job $\sigma(k)$ on machine m_j. If $p_{\sigma(k)} = 0$ go to step 4. Otherwise, if $C(S_j) + p_{\sigma(k)} > C_{LB}$ then $C(S_j) = C(S_j)$ and go to step 4. Otherwise, Set $C(S_j) = C(S_j) + p_{\sigma(k)}$, $S_j = (S_j\sigma(k))$ and set $p_{\sigma(k)} = 0$.

Step4. If k < Q, then set k = k + 1 and return to step 2. Otherwise, stop. The schedule where jobs in S_j are processed on machine j is the heuristic solution.

2.1.3 Sub-algorithm IP heuristic: Integer Programming
Let j index the various parallel machines and define C_{max} denote an upper bound of minimum makespan or maximum completion time of scheduling, s_{ij} denote the number of jobs of group i schedule on machine j, q_i denote the number of identical jobs and p_i denote the processing time, included any setup time required, in job-group i through a set of m identical parallel machines.

$$Z = \min C_{max} \tag{12}$$

Subject to $$\sum_{j=1}^{m} s_{ij} \ge q_i \quad ; \forall i = 1, 2, 3, ..., n \tag{13}$$

$$\sum_{i=1}^{n} p_i s_{ij} - C_{max} \le 0 \quad ; \forall j = 1, 2, 3, ..., m \tag{14}$$

$$C_{max}, s_{ij} \ge 0 \text{ and integer} \tag{15}$$

This heuristic attempt to strengthen makespan bound with a number of constrain machines referred as *IP* (Integer Programming) heuristic by *LINGO 6.0* software to generate the initial pattern, which is interrupted at 200,000 iterations.

2.2 Sub-algorithm ALSP: Adjusted List-scheduling Pattern

Step 1. Calculate a lower bound of C_{max}. Let $C = \sum_{i=1}^{n} p_i q_i$, then $C^*_{LB} = \lceil C/m \rceil$; integer.

Step 2. Calculate completion time of each machine schedule. $C(S_j) = \sum_{i=1}^{n} p_i s_{ij}$

Step 3. Check each machine schedule to lower bound. Set j = 1. If $C(S_j) > C^*_{LB}$, then go to step 4. Otherwise, j = j + 1 and do this step repeatedly until j = m then stop.

Step 4. Adjust each schedule to lower bound. Set i = n. If $s_{ij} > 0$, then $s_{ij} = s_{ij} - 1$ and return to Step 3. Otherwise, set i = n – 1 and do this step until i = 0 and then return to step 3.

2.3 Sub-algorithm CCG: Continuous Column Generation

Step 1. Formulate master problem as a cutting stock problem (1.2).

$$\text{Master problem} \qquad Z = \min \sum_{j=1}^{m} x_j \tag{16}$$

$$\text{Subject to} \qquad \sum_{j=1}^{m} s_{ij} x_j \geq q_i \tag{17}$$

$$x_j \geq 0 \tag{18}$$

Step 2. Formulate sub-problem and generate patterns as a knapsack problem in a cutting stock problem (1.2) and generate patterns until $v \leq 1$.

$$\text{Sub-problem} \quad Z = \max \sum_{i=1}^{n} \pi_i s_i \tag{19}$$

$$\text{Subject to} \qquad \sum_{i=1}^{n} p_i s_i \leq C_{LB} \tag{20}$$

$$s_i \geq 0 \text{ and integer} \tag{21}$$

2.4 Algorithm LSCCG: List-scheduling and continuous column generation

Let $C_{LB} = C^*_{LB}$ as an initial lower bound.

Step 1. Generate the initial pattern of $S_j = \{s_{ij}\}$ by sub-algorithm LS heuristic (2.1).

Step 2. Adjust the initial pattern by sub-algorithm ALSP (2.2).

Step 3. Formulate problem and generate patterns as a cutting stock problem by sub-algorithm CCG (2.3).

Step 4. Round down set of x_j to be integer and select schedule patterns. If the number of selected patterns $(\sum_{j=1}^{m} x_j) = 0$, then $C_{LB} = C_{LB} + 1$ and return to step 2. Otherwise, go to step 5.

Step 5. Calculate remaining jobs and machines

Step 6. Return to step 1 to generate patterns of remaining jobs on remaining machines. If $C(S_j) \leq C_{LB}$, then stop. Otherwise, return to step 2.

3. Empirical performance evaluation of algorithms

The performance of heuristic algorithms in finding minimum makespan schedules was empirically evaluated by solving in different experimental framework used. The proposed *LSCCG* experiment investigates four variables: the number of machines *(m)*; the number of job types *(n)*; the minimum and maximum values of a uniform distribution used to generate processing time *(p_{gen})*; and the minimum and maximum values of a uniform distribution used to generate the number of jobs in each type or item *(q_{gen})*. The *m* and *n* variables are the set of 10, 15, 20, 25 and 30, while p_{gen} and q_{gen} are generated by *U[50,100]* and *U[10,50]*, respectively. Experiment combination gives a total of 25 groups of the number of machines and the number of job types, 10 instances in each group. Thus, this gives a total of 250 instances.

LPTCCG and *IPCCG* heuristics were implemented in Microsoft Excel with Visual Basic Applications and callable library in *LINDO 6.1* software to solve linear programming in *CCG*. For *IPCCG* heuristic was solved *IP* heuristic by *LINGO 6* software. These implementations ran on a computer notebook, AMD K6-2 3D processor 475 MHz and 128 Mb of main memory. The

performance of minimizing makespan of each instance was normalized by an initial lower bound dividing, C^*_{LB} in equation *(22)*. Therefore, the normalized performance of minimizing makespan from heuristic h , denoted by P_h is calculated as follows:

$$P_h = C^*_{LB}/C_{max(h)} \quad (22)$$

The result is presented in table 2. The experimental parameters are followed by the average P_{LPT}, P_{LPTCCG}, P_{IP2HT}, P_{IP2M} and P_{IPCCG} for 10 instances. As expected, average performance of *IPCCG* heuristic is better than other heuristics as shown in figure 2; *IP2HT* (2×10^5 iterations) and *IP2M* (2×10^6 iterations) heuristics. However, *IPCCG* heuristic outperformed the *LPTCCG* heuristic but there are only few cases. In all the experimental, the ratio *n/m* had a significant effect on the relative performance of *LSCCG* heuristic as shown in figure 3. The performances of *LPTCCG* and *IPCCG* heuristics with *LPT*-based would decrease when *n/m* ratio increased. Especially, *LPTCCG* heuristic would empirically outperform the *LPT* heuristic while *n/m* ratio was more than 2.5 and *IPCCG* heuristic would decrease when *n/m* ratio increased as same as *LPTCCG* heuristic but it would not outperform from *IP* heuristic, *IP2HT*, although *n/m* ratio was 3 as illustrated in figure 4.

In empirical relation to the *CPU* times of *LPTCCG* and *IPCCG* heuristics required is illustrated in figure 5 and 6, respectively. From these figures, number of machines had a significant effect on the computer run times of *LSCCG* heuristic more than number of job types. Therefore, the computer run times increment of *IPCCG* heuristic had a certain trend more than *LPTCCG* heuristic.

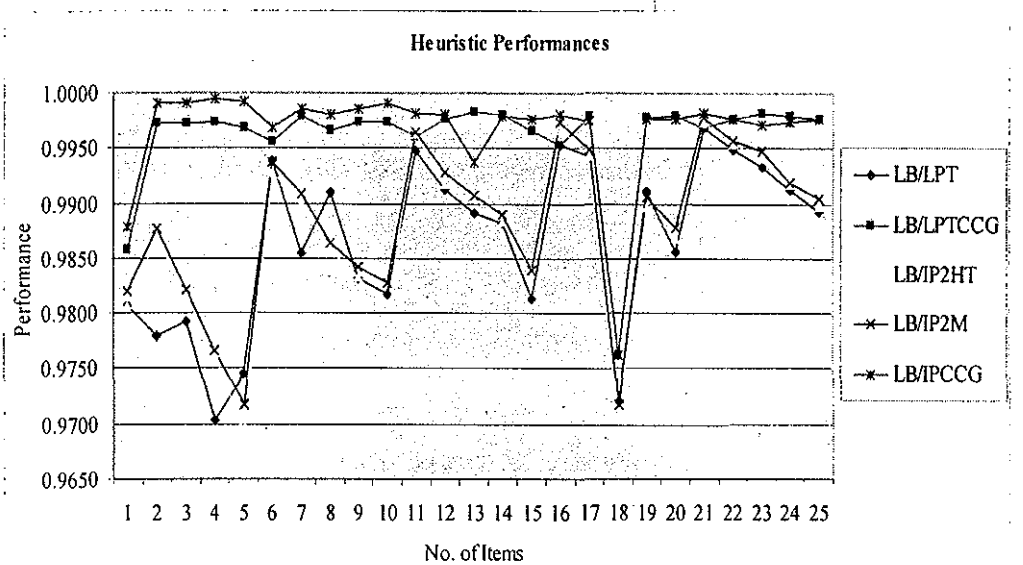

Figure 2 *Comparison of performance of heuristics*

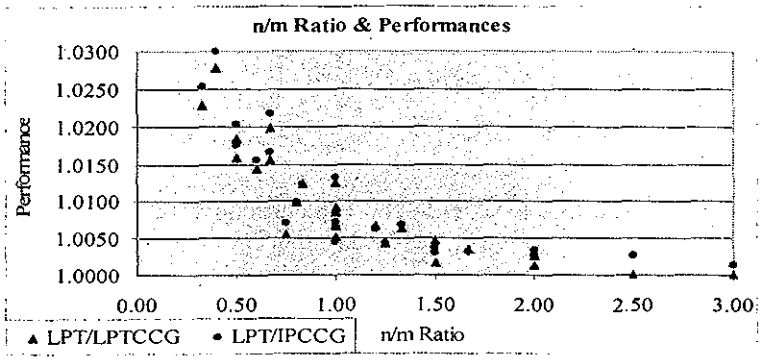

Figure 3 *Impact of n/m ratio to LPT-based performance of heuristics*

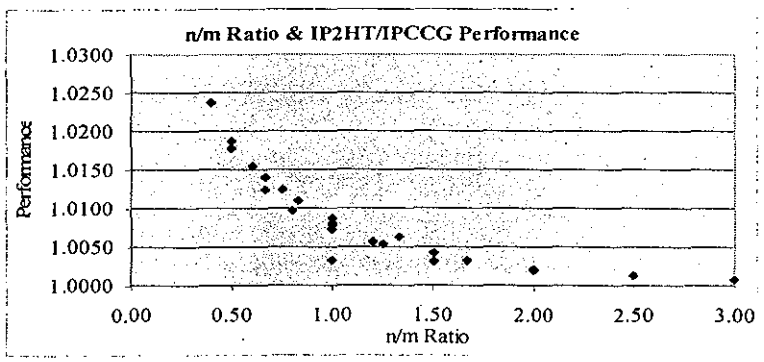

Figure 4 *Impact of n/m ratio to IP2HT-based performance of IPCCG heuristic*

	No. of			Ph					Run time (seconds)	
Items	n	m	n/m	LPT	LPTCCG	IP2HT	IP2M	IPCCG	LPTCCG	IPCCG
1	10	10	1.0000	0.9807	0.9857	0.9807	0.9819	0.9878	1664.60	396.60
2	10	15	0.6667	0.9779	0.9972	0.9869	0.9877	0.9991	925.00	359.50
3	10	20	0.5000	0.9792	0.9972	0.9816	0.9821	0.9990	997.20	303.80
4	10	25	0.4000	0.9704	0.9974	0.9763	0.9767	0.9995	925.00	217.90
5	10	30	0.3333	0.9746	0.9968	0.9690	0.9718	0.9992	989.60	262.30
6	15	10	1.5000	0.9939	0.9956	0.9928	0.9938	0.9969	4535.90	1090.00
7	15	15	1.0000	0.9855	0.9979	0.9900	0.9909	0.9986	2374.80	766.20
8	15	20	0.7500	0.9910	0.9966	0.9858	0.9864	0.9980	3830.30	943.00
9	15	25	0.6000	0.9832	0.9973	0.9833	0.9842	0.9985	2374.80	568.90
10	15	30	0.5000	0.9818	0.9974	0.9806	0.9827	0.9990	1711.00	575.70
11	20	10	2.0000	0.9948	0.9961	0.9963	0.9965	0.9982	7106.20	1302.10
12	20	15	1.3333	0.9913	0.9976	0.9918	0.9928	0.9980	2987.60	1981.80
13	20	20	1.0000	0.9892	0.9983	0.9905	0.9907	0.9937	3691.20	1708.30
14	20	25	0.8000	0.9882	0.9980	0.9884	0.9890	0.9979	2987.60	1602.00
15	20	30	0.6667	0.9814	0.9967	0.9839	0.9839	0.9976	5260.40	1398.10
16	25	10	2.5000	0.9953	0.9953	0.9968	0.9974	0.9980	4368.50	1436.70
17	25	15	1.6667	0.9944	0.9979	0.9943	0.9949	0.9976	3188.30	3162.70
18	25	20	1.2500	0.9720	0.9762	0.9711	0.9718	0.9763	7621.50	2418.90
19	25	25	1.0000	0.9912	0.9978	0.9897	0.9906	0.9976	3188.30	2476.00
20	25	30	0.8333	0.9856	0.9979	0.9869	0.9879	0.9977	3047.50	1795.50
21	30	10	3.0000	0.9969	0.9969	0.9974	0.9977	0.9982	2859.40	2062.10
22	30	15	2.0000	0.9950	0.9976	0.9955	0.9958	0.9976	5241.00	5445.70
23	30	20	1.5000	0.9934	0.9981	0.9941	0.9948	0.9971	5807.20	4891.70
24	30	25	1.2000	0.9913	0.9979	0.9918	0.9919	0.9974	5241.00	3594.20
25	30	30	1.0000	0.9893	0.9976	0.9899	0.9905	0.9976	5371.90	3188.30
Average.*				0.9897	0.9958	0.9903	0.9910	0.9965	3531.83	1757.92

Table 2 *The performances of LPT, LPTCCG, IP2HT, IP2M and IPCCG heuristic*

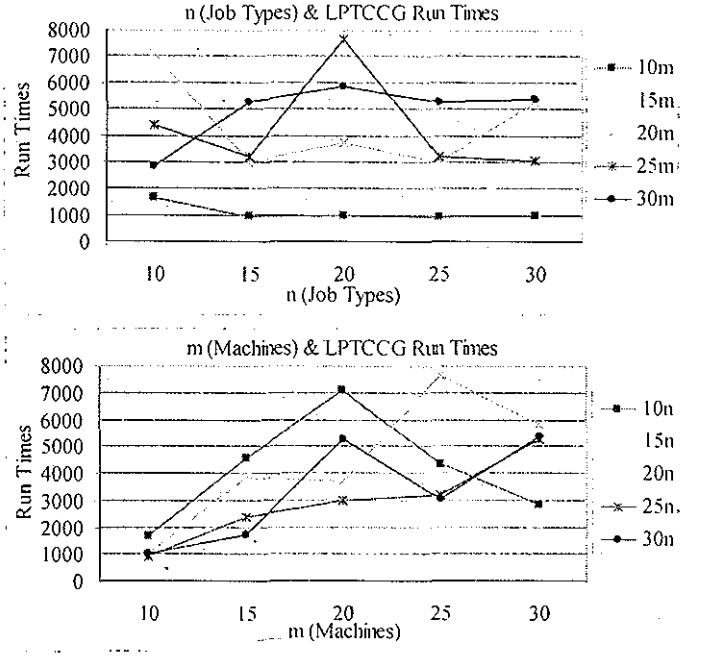

Figure 5 *Relationship of number of job types, number of machines and computer run times by LPTCCG*

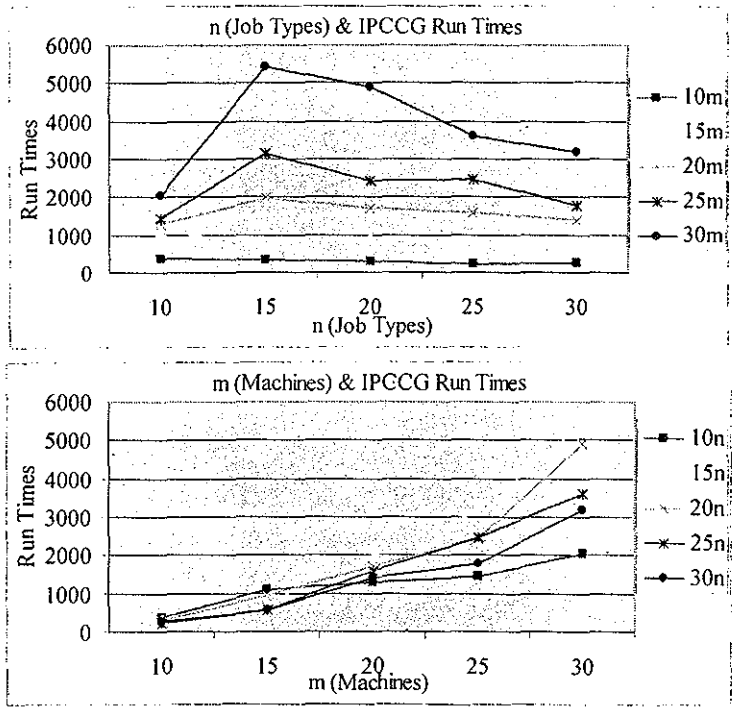

Figure 6 *Relationship of number of job types, number of machines and computer run times by IPCCG*

Conclusions

This paper presents a new heuristic for the identical parallel machines makespan problem. The proposed heuristic, called *LSCCG* is improved pattern solution by Continuous Column Generation procedure from solution of *LPT* heuristic, called *LPTCCG*, and from solution of *IP* heuristic, called *IPCCG*. It commonly used to solve the identical parallel machines makespan problem. The experimental demonstrated how the proposed heuristic outperforms the initial pattern under *n/m* ratio; the number of job types *(n)* and the number of machines *(m)*. These experiments showed that the proposed heuristic is quite robust and provides a better method to solve the makespan problem than *LPT* and *IP* heuristic. Furthermore, *n* and *m* had a significant effect on performance and *CPU* time of this proposed heuristic but *m* had more significant effect than *n*. The proposed *IPCCG* heuristic provides a better method to solve the makespan problem than *LPT, LPTCCG, IP2HT* (2×10^5 iterations) and *IP2M* (2×10^6 iterations) heuristics.

Further Research

We emphasize that this *LP*-based heuristic is combined with column generation and *LS* heuristic. Gupta and Ruiz-Torres (2001) presented a new heuristic for minimize makespan on parallel machines, called *LISTFIT* heuristic that is based on bin packing and list scheduling. Its worst-case performance bound is no worse than that of the commonly used *MULTIFIT* heuristic, Coffman et al (1978). Therefore, we should firstly combine *LISTFIT* heuristic and *CCG* procedure

to be *LFCCG* heuristic, *LISTFIT* heuristic and Continuous Column Generation, instead of *LPTCCG* or *IPCCG* heuristic and benchmark their performances in the further study. Especially, performances of *IP* heuristic by *LINGO 6* should be interrupted at suitable number of iterations as shown in figure 7 and 8; experiment combination gives a total of 25 groups of the number of machines and the number of job types, 10 instances in each group. Thus, this gives a total of 250 instances. Secondly, extension of this proposed heuristic generally restricted to setup time of each batch of job types and due-date of each order of jobs that must be added in column generation sub-problem modification scheme. Thirdly, ratio *n/m* which is *LSCCG* heuristic outperformed must be found out accurately. Finally, the algorithm should be implemented in *C* coding on *DOS* and *CPLEX*, popular optimization software, to solve *IP* heuristic and *CCG* in a huge size of problems.

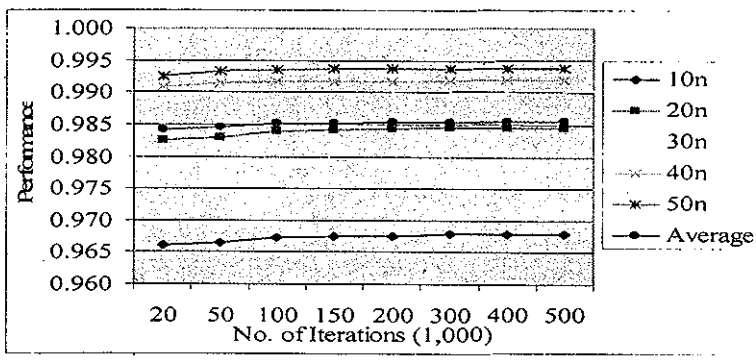

Figure 7 *Performances of IP heuristic by LINGO 6 at each number of iterations interruptions and number of job types*

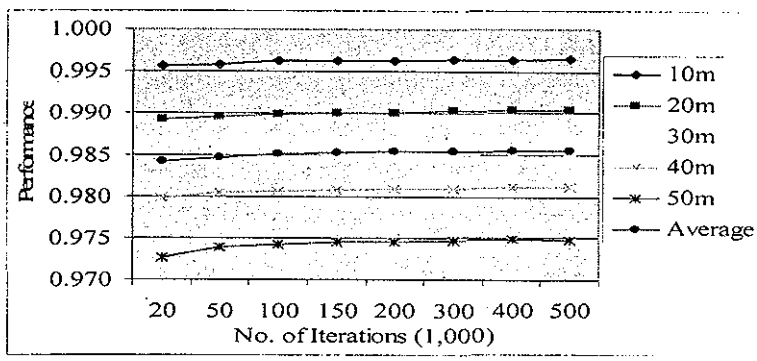

Figure 8 *Performances of IP heuristic by LINGO 6 at each number of iterations interruptions and number of machines*

References

Coffman, E.G., Garey, M.R. and Johnson, D.S. 1978. An Application of Bin-packing to Multiprocessor Scheduling. **SIAM Journal of Computing**. 7: 1-17.

Dyckhoff, H. 1981. A New Linear Programming Approach to the Cutting Stock Problem. **Operations Research, 29**. 1092-1104.

Dyckhoff, H. 1990. A Typology of Cutting and Packing Problems. **Euro J. of Operational Research**. 44: 145-159.

Garey, M.R. and Johnson, J.S. 1979. **Computers and Intractability : A Guide to the Theory of NP-completeness**. San Francisco: Freeman .

Gilmore, P.C. and Gomory, R.E. 1961. A Linear Programming Approach to the Cutting Stock Problem. **Operations Research**. 9: 849-859.

Gilmore, P.C. and Gomory, R.E. 1963. A Linear Programming Approach to the Cutting Stock Problem-Part II. **Operations Research**. 11: 863-888.

Golden, B.L. 1976. Approaches to the Cutting Stock Problem. AIIE Transections, 8, 2665-274.

Graham, R.L. 1969. Bounds on Multiprocessor Timing Anomalies. **SIAM J. of Applied Mathematics**. 17: 416-429.

Gupta, J.N.D. and Ruiz-Torres, J. 2001. A LISTFIT Heuristic for Minimizing Makespan on Identical Parallel Machines. **Production Planning & Control**. 12: 28-36.

Kantorovich, L.V. 1960. Mathematical Method of Organizing and Planning Production. **Management Science**. 6: 366-422.

Parker, R.G. 1995. **Deterministic Scheduling Theory**. London:Chapman & Hall.

Sukto, S. and Charnsethikul, P. 2002a. **A column generation approach for Scheduling of N job-groups through M parallel machines with minimum makespan**. Proceeding of Symposium in Production and Quality Engineering for Competitive Business Environment.; June; Kasetsart University. Bangkok. Thailand.

Sukto, S. and Charnsethikul, P. 2002b. **Scheduling of N job-groups through M parallel machines with minimum makespan : a column generation approach part II**. Proceeding of the 2002 IE Network National Conference, KMITNB.; October; Kanchanaburi. Thailand.