วิศวกรรมสาร มข. ปีที่ 32 ฉบับที่ 2 (217 - 236) มีนาคม - เมษายน 2548
KKU Engineering Journal Vol. 32 No. 2 (217 - 236) March – April 2005

List-Scheduling and Column-Generation for Scheduling of n Job-Groups with Set up Time through m Identical Parallel Machines to Minimize 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, p_i as the processing time and su_i as the 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 the *LPT* (Longest Processing Time) heuristic or in packing problems such as the *BFD* (Best-Fit Decreasing) heuristic and column generation procedure in a cutting stock problem, respectively. The first algorithm, referred to as *LPTCCG* which are hybrid between *LPT* heuristic, is constructed using the initial pattern and adjusted to a lower bound by *ALSP* (Adjusted list-scheduling pattern); while the *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, ALSP* and *CCG* to reach an upper bound of minimum makespan. The average performance of this proposed algorithm is considerably better than that of the *LPT* heuristic. The better algorithm, referred to as *BFDCCG*, is based on *BFD* heuristic and *CG*. As the same principle, the *BFD* heuristic is generated the initial pattern and adjusted to a lower bound by *ALSP* and then attempted to strengthen makespan bound with a number of available machines by *CCG*. *BFDCCG* heuristic is repeatedly looped by *BFD, ALSP* and *CCG* to reach an upper bound of minimum makespan. The average performance of this proposed algorithm. *BFDCCG* heuristic is considerably better than that of the *LPTCCG* heuristic and computational time also. In set up time case, referred to as *LPTsuCCG* and *BFDsuCCG* which are similar to *LSCCG*, the only difference is in the set up time addition after List-scheduling the initial pattern and sub problem formulation in *CCG* procedure. As expected, the average performance and computational time of the *BFDsuCCG* heuristic is considerably better than the *LPTsuCCG* heuristic.

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

* Original manucript submitted: July 14, 2004 and Final manucript received: August 16, 2004

Introduction

Scheduling problems have developed over some thirty-five years. In 1968, most certainly, this condition greatly influenced the literature on scheduling as well as the way the 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 including 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 objects are given by solid materials cut up into small items as pieces. Since trim loss optimization is a main objective one also speaks 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 α denotes dimensionality, β denotes kind of assignment, γ denotes assortment of large objects and δ denotes assortment of small items. In the case of parallel machines the 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*

From 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. The 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 time that machine 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 to start sooner, resulting in higher efficiency and resource utilization. In the parallel machines scheduling problem, minimization of makespan also identifies a bottleneck machine requirement of 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 the *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 to as *LPTCCG* which are a hybrid between the *LPT* (Longest processing time) heuristic (Graham, 1969) and the *BFD* (Best-fit decreasing) heuristic (Johnson et al., 1974), are constructed along the initial pattern of scheduling then adjusted to a lower bound by the *ALSP* (Adjusted list-scheduling pattern) and the *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 *LS*, *ALSP* and *CCG* to reach an upper bound of minimum makespan. The average performance of the *LPTCCG* is considerably better than that of the *LPT* heuristic in the case of the *n/m* ratio being less than 2.5 (Sukto and Charnsethikul, 2002). The second and better model, referred to as *BFDCCG*, is based on the *BFD* (Best-fit decreasing) heuristic. Then provide additional set up time after List-Scheduling, referred to as the *LPTsuCCG* and the *BFDsuCCG*.

1. Problem formulation and fundamental algorithms

Specifically, we consider the following scheduling problem; i = {1, 2, 3,..., n} of n simultaneously available groups of jobs are 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, setup time required, and number of jobs in each group i are p_i, su_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:

$$C(S_j)=\sum_{i\in Sj}(p_is_{ij}+su_i) \tag{1}$$

$$C=\sum_{i=1}^{n}(p_iq_i+su_i) \tag{2}$$

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

$$C^*_{LB}=\lceil C/m\rceil=\lceil\sum_{i=1}^{n}(p_iq_i+su_i)/m\rceil\text{; Integer} \tag{4}$$

Then, the problem considered in this paper is one of finding the subsets $\{S_1, S_2,..., S_m\}$ of the given n groups 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 a 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 follows.

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.

1.2 Algorithm BFD: Best-Fit Decreasing procedure

The basic bin packing problem to be considered can be stated quite simply: given a list L = $(a_1, a_2,\ldots, a_n)$ of real numbers in (0, 1], place the element of L into a minimum number L^* of "bins" so that no bin contains numbers whose sum exceeds 1 (Johnson et al,1974). We can expect the problem of finding a packing which uses exactly L^* bins to require in general a lengthy combinatorial search for its solution. Thus we feel justified in considering the performance of various heuristic algorithms for constructing packing. In particular we shall consider the following four placement algorithms.

1.2.1 First-fit (FF). Let the bins be indexed as $B_1, B_2, \ldots$, with each initially filled to level zero. The number $a_1, a_2,\ldots, a_n$ will be placed in that order. To place a_i, find the least j such that B_j is filled to level $\beta \le 1 - a_i$, and place a_i in B_j. B_j is now filled to level $\beta + a_i$.

1.2.2 Best-fit (BF). Let the bins be indexed as $B_1, B_2, \ldots$, with each initially filled to level zero. The number $a_1, a_2,\ldots, a_n$ will be placed in that order. To place a_i, find the least j such that B_j is filled to level $\beta \le 1 - a_i$ and β is as large as possible, and place a_i in B_j. B_j is now filled to level $\beta + a_i$.

1.2.3 First-fit decreasing (FFD). Arrange L = $(a_1, a_2,\ldots, a_n)$ into a non-increasing order and apply algorithm 2.2.1, first-fit (FF), to the derived list.

1.2.4 Best-fit decreasing (BFD). Arrange L = $(a_1, a_2,\ldots, a_n)$ into a non-increasing order and apply algorithm 1.2.2, best-fit (BF), to the derived list.

Neither the first-fit nor the best-fit algorithm will ever use more than (17/10) L* + 2 bins. Furthermore, if L is in decreasing order, then neither algorithm will use more than (11/9) L* + 4 bins and both can be implemented using O(n log n) operations. Simulation results on four placement algorithms indicate that the FFD and the BFD are almost always better than the FF and the BF for a random L, with the BF occasionally slightly better than the FF and the BFD better than the FFD also.

1.3 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 simply 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 be the various cutting patterns and

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

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

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

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

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

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 different time patterns which are used will generally be large, and rounding up to the next 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 $\overline{c_j} = 1 - \sum_{i=1}^{n} \pi_i a_{ij}$ where π_i are the simplex multipliers from (6). Since (5) – (7) is a minimization problem, we seek the most negative reduced cost, so the kth sub-problem becomes

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

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

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

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

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

A problem of this form is called the 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 an *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 a cutting stock problem. Continuous column generation is the modification of a column generation procedure to increase the 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 a scheduling pattern until the sum of the product of simplex multipliers by generated column (9) are equal to or less than one, $v \leq 1$. Then, the patterns will be selected if the number of patterns can be rounded down to be an integer and equal to or more than one, $x_j \geq 1$. Further, the *LSCCG* repeatedly loops in a part of remaining jobs. Finally, the lower bound of the problem will be increased while the *LSCCG* passes the stop criteria and is then returned to the *LSCCG*, repeatedly, until going into the stop criteria. This relationship is evidence as in the following diagram:

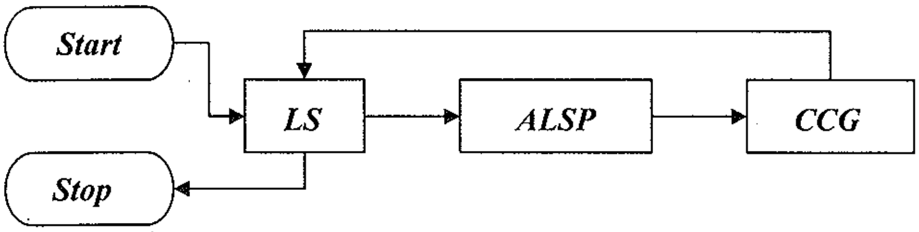

Figure 1 *LSCCG heuristic flow diagram*

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 $p_{\sigma(1)} \geq p_{\sigma(2)} \geq \ldots \geq p_{\sigma(Q)}$. Set $S_1 = \ldots = S_m = \phi$ and set k = 1. Furthermore, set $C(S_1) = \ldots = C(S_m) = 0$.

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 $C_{max} = \underset{1 \leq i \leq n}{Max}\{C(S_j)\}$.

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 $p_{\sigma(1)} \geq p_{\sigma(2)} \geq \ldots \geq p_{\sigma(Q)}$. Set $C_{LB} = \lceil \sum_{i=1}^{n} p_i q_i / m \rceil$, $S_1 = \ldots = S_m = \phi$ and set k and j = 1. Furthermore, set $C(S_1) = \ldots = C(S_m) = 0$.

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.

2.2 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 and let $C_{LB} = C^*_{LB}$.

Step 2. Calculate completion time of each machine schedule and upper bound. $C(S_j) = \sum_{i=1}^{n} p_i s_{ij}$, then $C_{UB} = \max\{ C(S_j)\}$.

Step 3. Check each machine schedule to lower bound. If $C_{UB} > C_{LB}$, then $P = C_{LB} / C_{UB}$ and go to step 4. Otherwise, stop.

Step 4. Adjust each schedule to lower bound. Let $q_i = \lfloor q_i P \rfloor$ and return to list scheduling (3.1). If $C_{max} > C_{LB}$, then $P = P - 0.05$ and do this step until $C_{max} \le C_{LB}$.

2.3 Sub-algorithm CCG: Continuous Column Generation

2.3.1 Continuous column generation without set up time

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

$$\text{Master problem} \quad Z = \min \sum_{j=1}^{m} x_j \tag{12}$$

$$\text{Subject to} \quad \sum_{j=1}^{m} s_{ij} x_j \ge q_i \tag{13}$$

$$x_j \ge 0 \tag{14}$$

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

$$\text{Sub-problem} \quad Z = \max \sum_{i=1}^{n} \pi_i s_i \tag{15}$$

$$\text{Subject to} \quad \sum_{i=1}^{n} p_i s_i \le C_{LB} \tag{16}$$

$$s_i \ge 0 \text{ and integer} \tag{17}$$

2.3.2 Continuous column generation with set up time

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

$$\text{Master problem} \quad Z = \min \sum_{j=1}^{m} x_j \tag{18}$$

$$\text{Subject to} \quad \sum_{j=1}^{m} s_{ij} x_j \ge q_i \tag{19}$$

$$x_j \ge 0 \tag{20}$$

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

$$\text{Sub-problem} \quad Z = \max \sum_{i=1}^{n} \pi_i s_i \tag{21}$$

$$\text{Subject to} \quad \sum_{i=1}^{n} (p_i s_i + su_i\, y_i) \le C_{LB} \tag{22}$$

$$s_i \le M\, y_i \tag{23}$$

$$s_i \ge 0 \text{ and integer, } y_i \in \{0,1\} \tag{24}$$

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 a sub-algorithm LS heuristic (2.1).

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

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

Step 4. Round down the 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 the remaining jobs and machines

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

3. Empirical performance evaluation of algorithms

3.1 LSCCG without set up time

The performance of heuristic algorithms in finding minimum makespan schedules was empirically evaluated by solving in a different experimental framework used. The proposed LSCCG experiment investigates four variables: the number of machines (m); the number of job groups (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 group or item (q_{gen}). The m and n variables are the set of 10, 20, 30, 40 and 50, while p_{gen} is generated by U[10,100] and q_{gen} is generated by a schedule bound control at 500, 1,000, 5,000 and 10,000. The experimental combination gives a total of 25 groups of the number of machines and the number of job groups, 10 instances in each group. Thus, this gives a total of 250 instances in each schedule bound so there are 1,000 instances totally.

LPTCCG and BFDCCG heuristics were implemented with C++ programming and callable library from LINGO 6 software to solve the linear programming in the CCG procedure. These implementations ran on a desktop computer, Intel Pentium 4 processor 2.0 GHz and 128 Mb of main memory. The performance of minimizing the makespan of each instance was normalized by an initial lower bound dividing, C^*_{LB} in equation (25). Therefore, the normalized performance of minimizing makespan from heuristic h , denoted by P_h is calculated as follows:

$$P_h = C_{\max(h)}/C^*_{LB} \tag{25}$$

The results of the heuristic performances are presented in Table 2 at 500 and 1,000 lower bound and in Table 3 at 5,000 and 10,000 lower bound approximately. The experimental parameters are followed by the average P_{LPT}, P_{LPTCCG} and P_{BFDCCG} for 10 instances. As expected, average performance of *LPTCCG* and *BFDCCG* are exactly better than the average performance of *LPT* because of initial pattern improvement. The average performance of the *BFDCCG* heuristic is quite better than other heuristics as shown in Figures 2 and 3, respectively. However, the *BFDCCG* heuristic outperformed the *LPTCCG* heuristic but there are only a few cases. In all of the experiment, the ratio of *n/m* and size of lower bound had a significant effect on the relative performance of the *LSCCG* heuristic.

In empirical relation to the *CPU* times of the *LPTCCG* and the *BFDCCG* heuristics required is illustrated in Table 4 and Figure 4. From these figures, the number of job groups, machines and size of lower bound had a significant effect on the computer run times of the *LSCCG*. Therefore, the computer run times increment of the *BFDCCG* heuristic had a certain trend less than the *LPTCCG* heuristic.

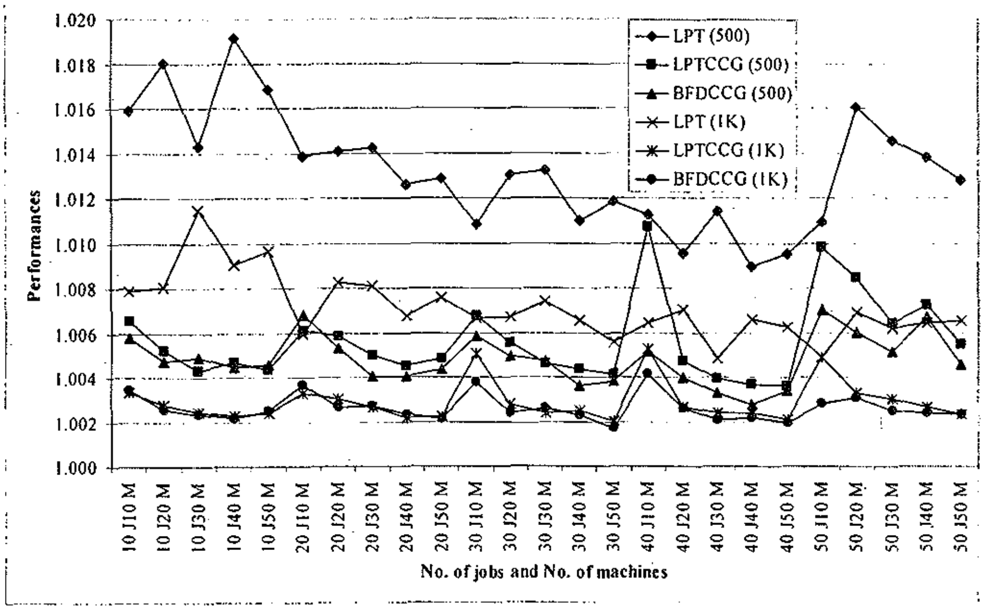

Figure 2 *Performances of heuristics at 500 and 1,000 lower bound.*

No. of Jobs and No. of Machines	Performances					
	500 LB			1,000 LB		
	LPT	*LPTCCG*	*BFDCCG*	*LPT*	*LPTCCG*	*BFDCCG*
10 J10 M	1.01588	1.0065	1.0058	1.0079	1.0034	1.0034
10 J20 M	1.01801	1.0052	1.0047	1.0080	1.0028	1.0025
10 J30 M	1.01432	1.0043	1.0049	1.0114	1.0025	1.0024
10 J40 M	1.01915	1.0047	1.0045	1.0091	1.0023	1.0022
10 J50 M	1.01687	1.0044	1.0046	1.0096	1.0024	1.0025
20 J10 M	1.01387	1.0061	1.0068	1.0060	1.0033	1.0037
20 J20 M	1.01407	1.0059	1.0053	1.0083	1.0031	1.0027
20 J30 M	1.01425	1.0050	1.0041	1.0081	1.0026	1.0027
20 J40 M	1.0126	1.0045	1.0040	1.0068	1.0022	1.0023
20 J50 M	1.01287	1.0048	1.0043	1.0076	1.0022	1.0021
30 J10 M	1.01084	1.0068	1.0059	1.0067	1.0051	1.0038
30 J20 M	1.01301	1.0056	1.0050	1.0067	1.0028	1.0024
30 J30 M	1.01325	1.0046	1.0048	1.0074	1.0024	1.0026
30 J40 M	1.01099	1.0044	1.0036	1.0066	1.0025	1.0023
30 J50 M	1.01184	1.0041	1.0038	1.0056	1.0020	1.0017
40 J10 M	1.01126	1.0107	1.0052	1.0064	1.0053	1.0042
40 J20 M	1.00953	1.0048	1.0040	1.0070	1.0027	1.0026
40 J30 M	1.01142	1.0039	1.0033	1.0049	1.0024	1.0021
40 J40 M	1.00896	1.0037	1.0028	1.0066	1.0024	1.0022
40 J50 M	1.00957	1.0037	1.0034	1.0063	1.0021	1.0020
50 J10 M	1.01092	1.0098	1.0070	1.0049	1.0049	1.0028
50 J20 M	1.01607	1.0085	1.0060	1.0070	1.0033	1.0031
50 J30 M	1.01456	1.0065	1.0052	1.0063	1.0030	1.0025
50 J40 M	1.0138	1.0073	1.0067	1.0065	1.0027	1.0024
50 J50 M	1.01278	1.0055	1.0046	1.0066	1.0023	1.0023
Average	1.01294	1.0057	1.0048	1.0071	1.0029	1.0026

Table 2 *Performances of LPT, LPTCCG and BFDCCG at 500 and 1,000 LB.*

No. of Jobs and No. of Machines	Performances					
	5,000 LB			10,000 LB		
	LPT	*LPTCCG*	*BFDCCG*	*LPT*	*LPTCCG*	*BFDCCG*
10 J10 M	1.00182	1.00073	1.00073	1.00108	1.00038	1.00035
10 J20 M	1.00209	1.00062	1.00058	1.00089	1.00027	1.00027
10 J30 M	1.00150	1.00044	1.00042	1.00083	1.00027	1.00024
10 J40 M	1.00192	1.00051	1.00049	1.00093	1.00026	1.00026
10 J50 M	1.00188	1.00050	1.00039	1.00111	1.00024	1.00022
20 J10 M	1.00126	1.00059	1.00059	1.00072	1.00034	1.00033
20 J20 M	1.00160	1.00062	1.00056	1.00067	1.00032	1.00028
20 J30 M	1.00145	1.00054	1.00048	1.00059	1.00025	1.00021
20 J40 M	1.00200	1.00056	1.00054	1.00089	1.00029	1.00028
20 J50 M	1.00164	1.00050	1.00043	1.00089	1.00028	1.00023
30 J10 M	1.00119	1.00109	1.00088	1.00066	1.00055	1.00034
30 J20 M	1.00135	1.00057	1.00055	1.00051	1.00024	1.00030
30 J30 M	1.00114	1.00054	1.00049	1.00064	1.00027	1.00023
30 J40 M	1.00113	1.00046	1.00042	1.00060	1.00024	1.00025
30 J50 M	1.00127	1.00046	1.00043	1.00060	1.00023	1.00022
40 J10 M	1.00132	1.00127	1.00095	1.00058	1.00056	1.00039
40 J20 M	1.00141	1.00064	1.00068	1.00079	1.00028	1.00032
40 J30 M	1.00109	1.00057	1.00050	1.00072	1.00026	1.00025
40 J40 M	1.00114	1.00069	1.00044	1.00053	1.00027	1.00027
40 J50 M	1.00102	1.00047	1.00042	1.00070	1.00027	1.00024
50 J10 M	1.00106	1.00106	1.00085	1.00068	1.00068	1.00053
50 J20 M	1.00126	1.00069	1.00058	1.00069	1.00026	1.00026
50 J30 M	1.00100	1.00052	1.00043	1.00065	1.00028	1.00027
50 J40 M	1.00130	1.00049	1.00045	1.00048	1.00027	1.00023
50 J50 M	1.00105	1.00054	1.00044	1.00057	1.00026	1.00026
Average	1.00139	1.00063	1.00055	1.00072	1.00031	1.00028

Table 3. *Performances of LPT, LPTCCG and BFDCCG at 5,000 and 10,000 LB.*

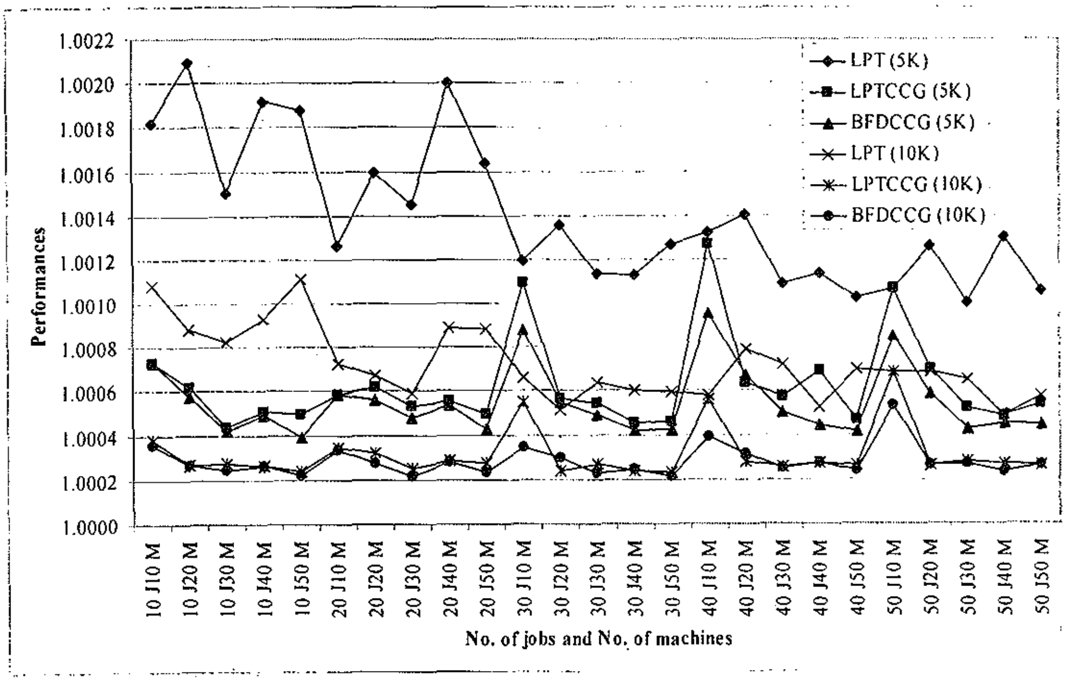

Figure 3 *Performances of heuristics at 5,000 and 10,000 lower bound.*

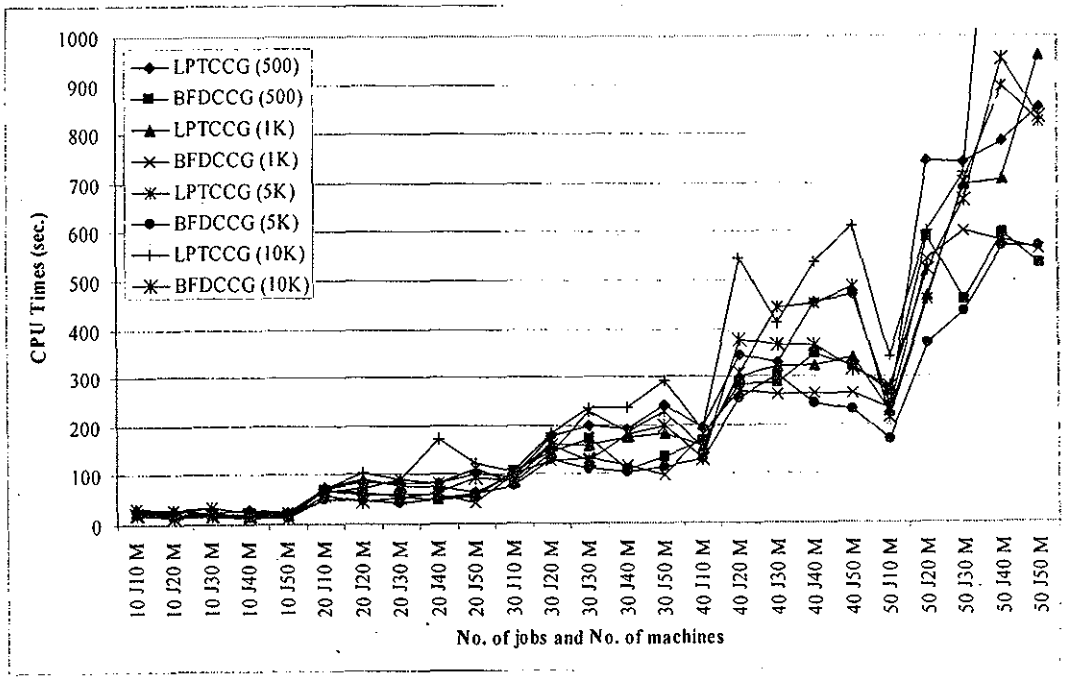

Figure 4 *Computational times of heuristics at various lower bound.*

No. of Jobs and No. of Machines	*CPU* Times (sec.)							
	500 LB		1,000 LB		5,000 LB		10,000 LB	
	LPT CCG	*BFD CCG*	*LPT CCG*	*BFD CCG*	*LPT CCG*	*BFD CCG*	*LPT CCG*	*BFD CCG*
10 J10 M	28.50	21.90	27.50	19.30	31.40	22.70	32.00	21.50
10 J20 M	27.80	18.00	22.50	15.40	28.90	20.90	22.80	12.30
10 J30 M	24.20	17.90	22.30	15.80	34.70	19.80	34.90	19.30
10 J40 M	28.00	16.60	21.40	13.10	24.00	20.20	25.20	18.70
10 J50 M	23.80	19.00	18.80	14.40	24.90	13.90	21.70	13.60
20 J10 M	72.50	70.30	73.50	68.40	67.20	51.50	70.80	58.30
20 J20 M	84.30	62.80	89.10	58.20	72.30	48.80	102.80	46.30
20 J30 M	81.40	55.60	76.80	61.00	91.00	43.80	89.10	52.20
20 J40 M	81.60	47.80	74.30	57.60	82.00	49.90	173.70	63.70
20 J50 M	100.10	55.80	64.50	42.90	110.90	61.60	123.40	92.20
30 J10 M	95.90	106.60	99.40	100.40	83.60	78.40	105.20	87.30
30 J20 M	177.70	145.40	161.90	126.90	143.60	131.60	183.50	159.20
30 J30 M	200.20	172.40	159.00	133.40	228.40	111.10	237.60	128.00
30 J40 M	190.40	108.50	174.50	117.00	187.00	104.50	236.50	181.40
30 J50 M	242.90	136.20	182.40	99.30	228.30	115.60	293.00	199.60
40 J10 M	194.20	168.20	155.90	181.90	150.70	130.30	189.70	129.10
40 J20 M	345.40	285.20	300.80	271.60	309.50	255.50	545.50	376.50
40 J30 M	330.90	286.50	322.30	264.10	443.10	302.30	412.30	366.80
40 J40 M	455.40	348.90	325.00	265.20	451.10	244.70	537.50	368.10
40 J50 M	469.70	320.70	339.30	265.90	488.00	232.20	610.60	315.70
50 J10 M	242.70	268.10	229.10	233.60	216.10	170.20	343.30	280.30
50 J20 M	745.50	588.70	469.20	541.30	459.30	370.60	601.90	521.50
50 J30 M	741.20	460.20	697.10	601.20	702.80	436.70	720.00	666.00
50 J40 M	784.00	597.00	706.20	582.80	895.70	568.20	1549.90	951.40
50 J50 M	854.90	533.40	959.20	563.60	824.78	569.89	1528.80	834.90
Average	264.93	196.47	230.88	188.57	255.17	167.00	351.67	238.56

Table 4 *Computational times of LPTCCG and BFDCCG at various LB.*

3.2 LSCCG with set up time

The performance of heuristic algorithms in finding minimum makespan schedules with set up time constraints were empirically evaluated by solving in the different experimental frameworks used. The proposed LSsuCCG experiment investigates five variables: the number of machines (m); the number of job groups (n); the minimum and maximum values of a uniform distribution used to generate processing times (p_{gen}); and the minimum and maximum values of a uniform distribution used to generate the number of jobs in each group or item (q_{gen}). The m and n variables are the set of 10, 20 and 30, while p_{gen} is generated by U[10,100] and q_{gen} is generated by a schedule bound control at 500, 1,000, 5,000 and 10,000 lower bound approximately. Two intervals of set up time (su_{gen}) are generated by U[1,50] and U[50,150]. The experiment combination gives a total of 9 groups of the number of machines and the number of job groups, with 10 instances in each group. Thus, this gives a total of 90 instances in each schedule bound and set up time so there are 720 instances totally.

The LPTsuCCG and the BFDsuCCG heuristics were implemented with a C++ programming and callable library from the LINGO 6 software to solve linear programming in a CCG procedure. These implementations ran on a desktop computer, Intel Pentium 4 processor with 2.0 GHz and 128 Mb of main memory. The performance of minimizing the makespan of each instance was normalized by an initial lower bound dividing as the same previous algorithms. In the case of a lower bound of approximately 500, the result is presented in Table 5 and Figure 5.

No. of Jobs and No. of Machines	Performances			
	su [1,50]		*su [50,150]*	
	LPTsuCCG	*BFDsuCCG*	*LPTsuCCG*	*BFDsuCCG*
10 J10 M	1.0758	1.0638	1.1744	1.1318
10 J20 M	1.0762	1.0660	1.2031	1.1815
10 J30 M	1.0832	1.0790	1.2256	1.1930
20 J10 M	1.0677	1.0075	1.1594	1.0549
20 J20 M	1.0860	1.0539	1.1611	1.1318
20 J30 M	1.0798	1.0543	1.1687	1.1540
30 J10 M		1.0350		1.0619
30 J20 M		1.0336		1.0755
30 J30 M		1.0423		1.1277
Average	1.0781	1.0484	1.1821	1.1236

Table 5 *Performances of LPTsuCCG and BFDsuCCG with su [1,50] and [50,150] at 500 LB.*

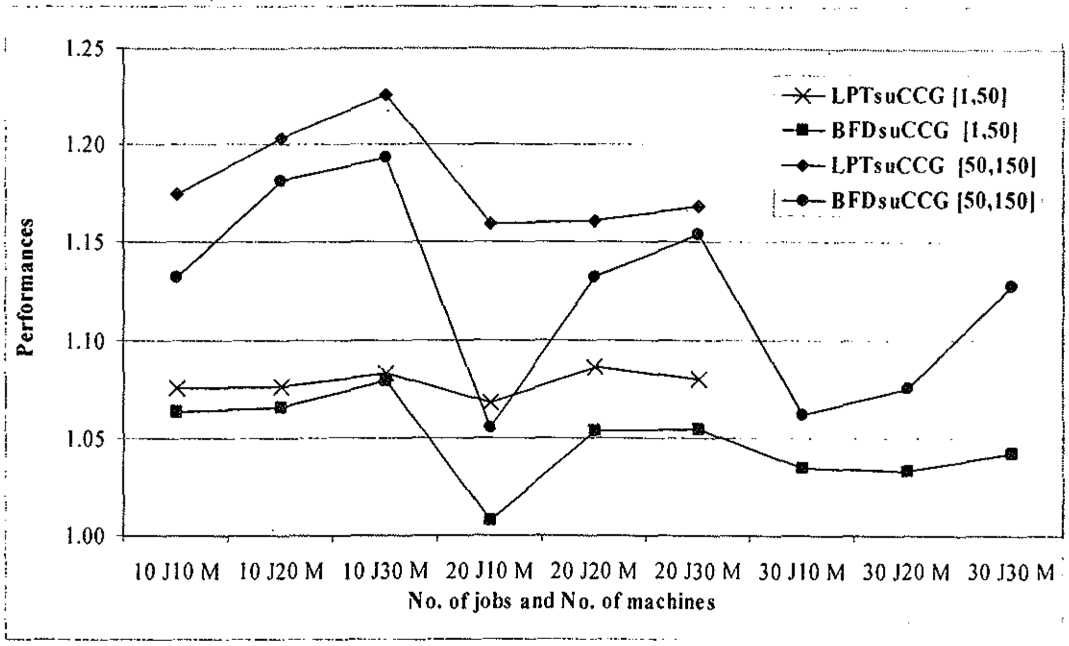

Figure 5 *Performances of LPTsuCCG and BFDsuCCG in each set up time at 500 lower bound.*

These experimental parameters are followed by the average $P_{LPTsuCCG}$ and $P_{BFDsuCCG}$ for 10 instances each with two intervals of set up time. As expected, average performance of *BFDsuCCG* heuristic is exactly better than *LPTsuCCG* heuristic as shown in Figure 5. Especially, the *LPTsuCCG* heuristic took computational time more than half an hour in case of 30 job-groups up. In the same of heuristic, the instances with smaller intervals of set up time had better performances than the larger ones because of initial pattern improvement and size of lower to upper bound of makespan. Therefore, next of the experimental was interested in *BFDsuCCG* heuristic, the results of this heuristic performance are presented in Table 6 at 1,000, 5,000 and 10,000 lower bound and difference intervals of set up time. As the result in Figure 4 is illustrated, the instances with smaller intervals of set up time at the same size of lower bound had better performances than the larger ones in each pair and the instances with larger size of lower bound at the same intervals of set up time had better performances because of initial pattern improvement and size of lower to upper bound of makespan in each group of parameter sizes.

In empirical relation to the *CPU* times of *BFDsuCCG* heuristic required is illustrated in Table 7 and Figure 7. From these Figures, number of job groups, machines, size of lower bound and had a significant effect on the computer run times of this heuristic. Therefore, the computer run times increment of *BFDsuCCG* heuristic had a certain trend up to number of job groups more than others parameter.

No. of Jobs and No. of Machines	*BFDsuCCG* Performances					
	1,000 LB		5,000 LB		10,000 LB	
	su [1,50]	*su [50,150]*	*su [1,50]*	*su [50,150]*	*su [1,50]*	*su [50,150]*
10 J10 M	1.0099	1.0889	1.0057	1.0202	1.0035	1.0098
10 J20 M	1.0357	1.0991	1.0071	1.0228	1.0039	1.0122
10 J30 M	1.0358	1.1105	1.0065	1.0192	1.0031	1.0121
20 J10 M	1.0216	1.0498	1.0062	1.0177	1.0028	1.0098
20 J20 M	1.0282	1.0821	1.0072	1.0216	1.0034	1.0112
20 J30 M	1.0337	1.0991	1.0068	1.0235	1.0033	1.0119
30 J10 M	1.0217	1.0412	1.0049	1.0127	1.0030	1.0095
30 J20 M	1.0169	1.0613	1.0056	1.0195	1.0027	1.0102
30 J30 M	1.0235	1.0822	1.0070	1.0219	1.0034	1.0113
Average	1.0252	1.0794	1.0063	1.0199	1.0032	1.0109

Table 6 *Performances of BFDsuCCG with su [1,50] and [50,150] at various LB.*

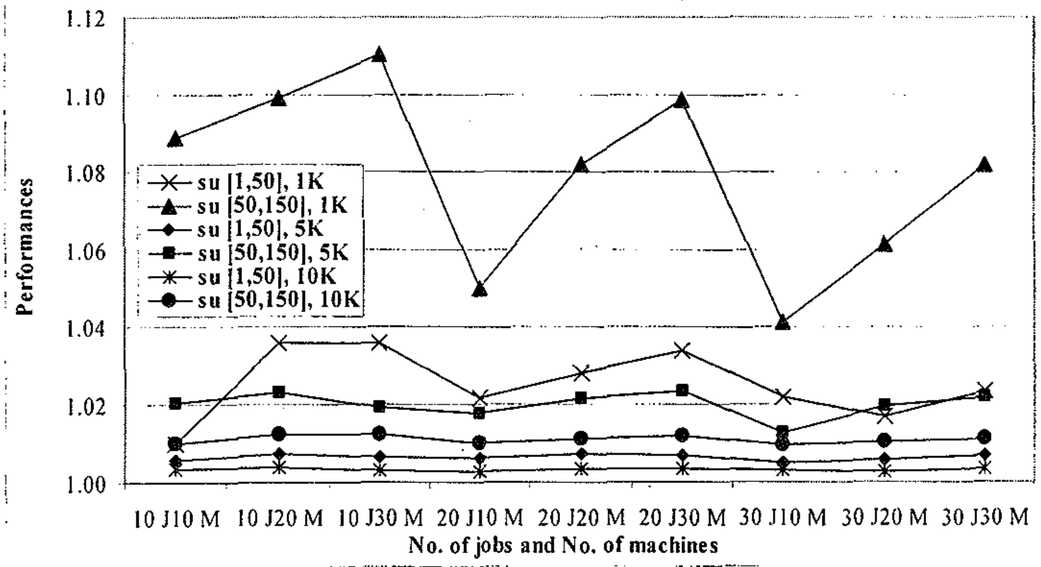

Figure 6 *Performances of BFDsuCCG in each set up time and lower bound.*

No. of Jobs and No. of Machines	BFDsuCCG CPU Times (sec.)							
	500 LB		1,000 LB		5,000 LB		10,000 LB	
	su [1,50]	*su [50,150]*	*su [1,50]*	*su [50,150]*	*su [1,50]*	*su [50,150]*	*su [1,50]*	*su [50,150]*
10 J10 M	240.90	416.10	241.70	559.40	304.20	873.40	279.90	740.30
10 J20 M	249.00	472.70	314.80	640.50	363.40	1000.40	309.60	858.60
10 J30 M	282.00	384.70	315.10	613.30	402.80	948.40	364.20	1030.50
20 J10 M	884.60	858.30	1081.50	1714.50	1516.20	3905.70	1146.00	3959.30
20 J20 M	978.30	1532.50	1238.20	2069.60	1556.30	3711.70	1249.30	4061.00
20 J30 M	1284.80	1911.30	1525.00	2472.60	1712.10	4356.80	1529.90	4991.90
30 J10 M	3506.80	7321.60	4859.00	9300.60	5953.00	16567.00	3740.20	18653.60
30 J20 M	2672.00	3334.00	3204.00	7474.60	4698.40	16256.60	4057.80	17730.40
30 J30 M	3633.80	4495.20	5440.40	8953.60	5352.40	16455.00	4278.60	15936.20
Average	1525.80	2302.93	2024.41	3755.41	2428.76	7119.44	1883.94	7551.31

Table 7 *Computational times of BFDsuCCG with su [1,50] and [50,150] at various LB.*

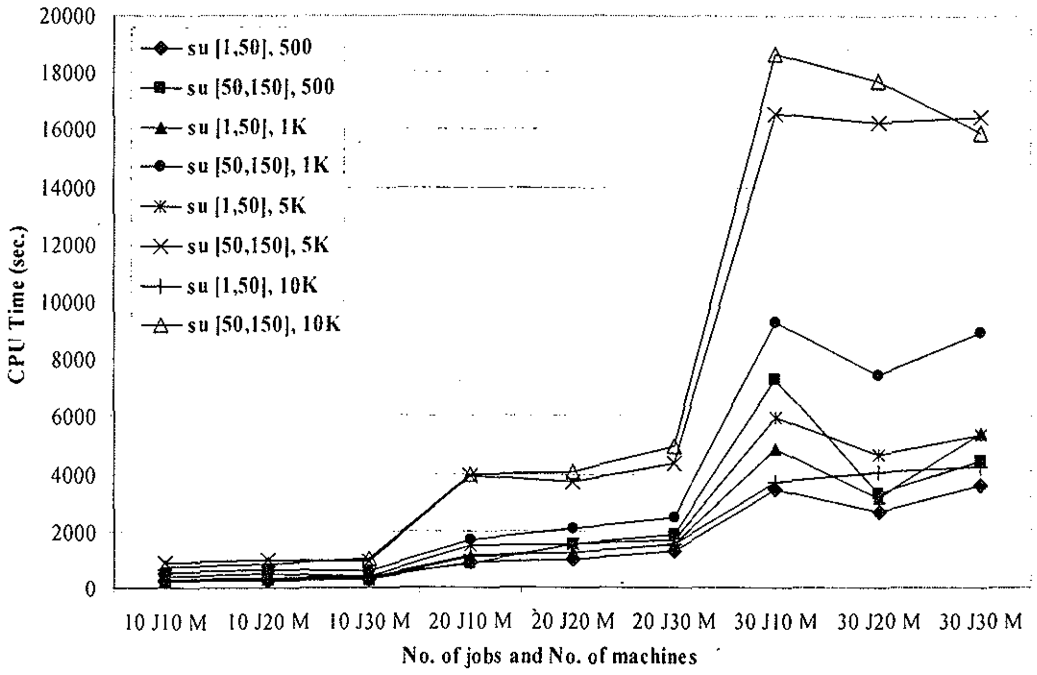

Figure 7 *Computational times of BFDsuCCG in each set up time and lower* bound.

Conclusions

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, p_i as the processing time and su_i as the setup time required, in job-group i through a set of m identical parallel machines. This heuristic, referred to 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 the *LPT* (Longest Processing Time) heuristic or in a packing problem such as the *BFD* (Best-fit decreasing) heuristic and column generation procedure in cutting stock problem, respectively. The algorithms, referred to as the *LPTCCG* and the *BFDCCG* heuristic which are hybrid between *LS* heuristic, is constructed the initial pattern and adjusted to a lower bound by *ALSP* (Adjusted list-scheduling pattern); while the *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 *LS*; *ALSP* and *CCG* to reach an upper bound of minimum makespan. The average performance of the *LPTCCG* heuristic is considerably better than that of the *LPT* heuristic. The results at 500, 1,000, 5,000 and 10,000 lower bound approximately. As expected, average performance of the *LPTCCG* and the *BFDCCG* are exactly better than the average performance of the *LPT* because of initial pattern improvement. The average performance of the *BFDCCG* heuristic is quite better than other. However, the *BFDCCG* heuristic outperformed the *LPTCCG* heuristic but there are only few cases. In all the experimental, the ratio of n/m and size of lower bound had a significant effect on the relative performance of the *LSCCG* heuristic. The *CPU* times of these heuristics required, number of job groups, number of machines and size of lower bound had a significant effect on the computer run times of the *LSCCG*. Therefore, the computer run times increment of the *BFDCCG* heuristic had a certain trend less than the *LPTCCG* heuristic.

The average performance in the set up time case, referred to as the *LPTsuCCG* and the *BFDsuCCG* which are similar to the *LSCCG*, the only difference is in the set up time addition after List-Scheduling the initial pattern and sub problem formulation in the *CCG* procedure. As expected, the average performance and computational time of the *BFDsuCCG* heuristic is considerably better than the *LPTsuCCG* heuristic. These experimental parameters with two intervals of set up time, the *LPTsuCCG* heuristic took computational time more than half an hour in the case of 30 job-groups up. In the same of heuristic, the instances with smaller intervals of set up time had better performances than the larger ones because of initial pattern improvement and size of lower to upper bound of makespan. Considerably, the results of the *BFDsuCCG* heuristic performance in various sizes of lower bound and difference intervals of set up time, the instances with smaller intervals of set up time at the same size of lower bound had better performances than the larger ones in each pair and the instances with larger size of lower bound at the same intervals of set up time had better performances because of initial pattern improvement and size of lower to upper bound of makespan in each group of parameter sizes. In empirical relation to the *CPU* times of the *BFDsuCCG* heuristic required, number of job-groups, number of machines, size of lower bound and had a significant effect on the computer run times of this heuristic. Therefore, the computer run times increment of the *BFDsuCCG* heuristic had a certain trend up to number of job groups more than the other parameters.

References

Coffman, E.G. Garey, M.R. and Johnson, D.S. 1978. An Application of Bin-packing to Multi-processor 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.

Johnson D.S., Demers, A., Ullman, J.D., Gary, M.R. and Graham R.L. 1974. Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms. **SIAM Journal of Computing**. 3,4: 299-325.

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. 2002. **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.