A hybrid genetic heuristic for scheduling parallel batch processing machines with arbitrary job sizes

Ali Husseinzadeh Kashan, Behrooz Karimi*, Masoud Jenabi

Department of Industrial Engineering, Amirkabir University of Technology, 424 Hafez Ave., Tehran 15916-34311, Iran

Available online 17 August 2006

Abstract

This paper investigates the scheduling problem of parallel identical batch processing machines in which each machine can process a group of jobs simultaneously as a batch. Each job is characterized by its size and processing time. The processing time of a batch is given by the longest processing time among all jobs in the batch. Based on developing heuristic approaches, we proposed a hybrid genetic heuristic (HGH) to minimize makespan objective. To verify the performance of our algorithm, comparisons are made through using a simulated annealing (SA) approach addressed in the literature as a comparator algorithm. Computational experiments reveal that affording the knowledge of problem through using heuristic procedures, gives HGH the ability of finding optimal or near optimal solutions in a reasonable time.

© 2006 Elsevier Ltd. All rights reserved.

Keywords: Scheduling; Parallel batch processing machines; Genetic algorithm; Simulated annealing

1. Introduction

This paper addresses the problem of scheduling \( n \) jobs on \( m \) identical parallel batch processing machines. A batch processing machine is capable to accommodate a group of jobs simultaneously. Once processing of a batch is initiated, it cannot be interrupted and other jobs cannot be introduced into the machine until processing is completed. With each job \( i \) we shall associate an arbitrary size \( s_i \) and an arbitrary processing time \( p_i \). Each batch processing machine has a capacity \( B \), so the sum of job sizes in each batch must be less than or equal to \( B \). We assume that no job has a size exceeding the machine capacity and it cannot be split across batches.

This research is motivated by burn-in operation in semiconductor manufacturing [1]. The purpose of burn-in operation is to test the integrated circuits by subjecting them to thermal stress for an extended period. The burn-in oven has a limited capacity and sub-grouping the boards holding the circuits in to batches would be inevitable. So, using all or as much available space in the processor maximizes efficiency. The processing times of burn-in operations are generally longer compared to those of other testing operations, so the burn-in operations constitute a bottleneck in the final testing operation and optimal scheduling of the burn-in operations is of great concern in productivity and on-time delivery management.

In scheduling problems, makespan (\( C_{\text{max}} \)) is equivalent to the completion time of the last job leaving the system. The small \( C_{\text{max}} \) usually implies a high utilization. The utilization for bottleneck station is closely related to throughput

* Corresponding author. Tel.: +98 21 66413034; fax: +98 21 66413025.
E-mail address: b.karimi@aut.ac.ir (B. Karimi).

0305-0548/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2006.07.005
rate of the system. Therefore, reducing the $C_{\text{max}}$ should also lead to a higher throughput rate. This concern led us to consider the makespan performance measure as the scheduling objective.

The structure of the paper is as follows. In Section 2, we review related works on scheduling models of batch processing machines. Section 3 examines the combinatorial complexity of the problem under the considered assumptions. Sections 4 and 5, respectively, deal with the proposed lower bound and a heuristic approach developed for the problem. The proposed hybrid genetic heuristic (HGH) and its implementation is presented in Section 6. Section 7 deals with the effectiveness of the HGH compared to the SA comparator algorithm. The paper will be concluded in Section 8.

2. Previous related works

Problems related to scheduling batch processing machines, in which machine has the ability of processing a number of jobs simultaneously, have been examined extensively by many researchers in recent years. Ikura and Gimple [2] probably are the first who addressed scheduling with batch processors. Given identical job processing times, they proposed an $O(n^2)$ algorithm for the problem of determining whether a schedule exists in which all the due dates can be met, when release times and due dates are agreeable. Lee et al. [1] considered minimizing maximum lateness, $L_{\text{max}}$, and number of tardy jobs, $\sum U_i$, under the same assumptions. Li and Lee [3] showed that these problems are both strongly NP-hard, if release times and due dates are not agreeable.

There are number of researches concerned with scheduling burn-in ovens in semiconductor manufacturing industry. In such these operations, the batch processing time depends to batch content and is given by the processing time of the longest job in the batch. Lee et al. [1], the first researchers who popularized scheduling burn-in operations, gave efficient algorithms for minimizing $L_{\text{max}}$ and $\sum U_i$, assuming agreeable processing times and due dates. Brucker et al. [4] gave several exact algorithms and complexity results for the case of infinite machine capacity. For the limited case, they proved that due date-based scheduling criteria give rise to NP-hard problems and provided an $O(n^{B/(B-1)})$ exact algorithm for minimizing $\sum C_i$. Chandru et al. [5] provided heuristics and a branch-and-bound method for minimizing $\sum C_i$ that can be implemented for solving problems with up to 35 jobs. Adding some new elimination rules, Dupont and Jolai Ghazvini [6] made their method useful for the problems with up to 100 jobs. Uzsoy and Yang [7] studied the problem of minimizing total weighted completion time, $\sum w_i C_i$, and provided some heuristics and a branch-and-bound algorithm. Lee and Uzsoy [8] studied the problem of minimizing $C_{\text{max}}$ in the presence of dynamic job arrivals. Scheduling a single batch processing machine where boards are also considered as a secondary resource constraint has been considered by Kempf et al. [9].

Scheduling batching machines with incompatible job families, i.e. when the processing times of all jobs in the same family are equal and jobs of different families cannot be processed together in the same batch, has been addressed by researchers. Uzsoy [10] developed efficient algorithms for minimizing $L_{\text{max}}$ and $\sum w_i C_i$ on a single batch processing machine. He also addressed the problems of minimizing $C_{\text{max}}$ and $L_{\text{max}}$ in the presence of dynamic job arrivals. Mehta and Uzsoy [11] provided exact and heuristic algorithms for minimizing total tardiness, $\sum T_i$. Dobson and Nambimadom [12] studied the problem of minimizing $\sum w_i C_i$ where each job requires a different amount of machine capacity. Jolai [13] showed that the problem of minimizing number of tardy jobs, $\sum U_i$, is NP-hard. In this paper a dynamic programming algorithm with polynomial time complexity for the fixed number of job families and batch machine capacity is provided.

Minimizing $\sum C_i$ on a batch processing machine with compatible job families has been considered by Chandru et al. [14], Hochbaum and Landy [15] and Brucker et al. [4]. They have presented dynamic programming algorithms with time complexity order of $O(m^3 B^{m+1})$, $O(3^m m^2)$ and $O(B^2 2^m m^2)$, respectively.

Considering jobs with different sizes, Uzsoy [16] gave complexity results for the $C_{\text{max}}$ and $\sum C_i$ criteria and provided some heuristics and a branch-and-bound algorithm. Jolai and Dupont [17] considered $\sum C_i$ criterion for the same problem. A branch-and-bound procedure for minimizing the $C_{\text{max}}$ was also developed by Dupont and Dhaenens-Flipo [18]. Considering jobs release times, Shuguang et al. [19] presented an approximation algorithm with worst-case ratio $2 + \varepsilon$ for this problem.

Recently solving the batch machine scheduling problems using metaheuristic algorithms have been considered by researchers. Wang and Uzsoy [20] coupled a random key-based representation genetic algorithm (GA) with a dynamic programming algorithm to minimize $L_{\text{max}}$ on a batch processing machine in the presence of job ready times. Melouk et al. [21] developed a simulated annealing (SA) approach for minimizing $C_{\text{max}}$ to schedule a batch processing machine.
with different job sizes. Koh et al. [22] proposed some heuristics and a random key-based representation GA approach for the problems of minimizing $C_{\text{max}}$ and $\Sigma w_i C_i$ on a batch processing machine with incompatible job families and non-identical job sizes.

Scheduling with identical parallel batch processing machines, Lee et al. [1] examined the worst-case error bound of any list-scheduling algorithm for minimizing $C_{\text{max}}$ with unit job sizes and extended the use of LPT rule [23] to the case of parallel batching machines. They also proposed the worst-case error bound of a list-scheduling algorithm for minimizing $L_{\text{max}}$. Balasubramanian et al. [24] developed two different versions of GA, based on the decomposition approaches for minimizing total weighted tardiness $\Sigma w_i T_i$, with incompatible job families. Also in another paper Monch et al. [25] extended their solution method to consider the case of unequal job ready times. Koh et al. [26] proposed a number of heuristics and designed a random key-based GA for the problems of minimizing $C_{\text{max}}$ and $\Sigma w_i C_i$, with incompatible job families and non-identical job sizes. Malve and Uzsoy [27] considered the problem of minimizing $L_{\text{max}}$ with dynamic job arrivals and incompatible job families. They proposed a random key-based GA combined with a family of iterative improvement heuristics. Chang et al. [28] developed a SA approach for minimizing $C_{\text{max}}$ to schedule different size jobs. They compared the results of proposed SA with CPLEX optimization software. We use their proposed SA as a comparator algorithm in our computational experiments, since it is a recently published benchmark algorithm based on our problem assumptions.

3. Combinatorial complexity

Uzsoy [16] proved that makespan minimization on a batch processing machine with non-identical job sizes and identical processing times is equivalent to a bin-packing problem which is NP-hard in the strong sense. Considering each batch as an individual job, optimal scheduling of parallel machines for minimizing makespan is also NP-hard in the ordinary sense even for two machines. Regarding the cited results the NP-hardness of minimizing makespan under the considered assumptions can be inferred.

To find out the intractability of the problem, an upper bound on the search space is attainable through using the following relation:

$$\sum_{k=1}^{n} \left[ \frac{1}{k!} \sum_{b=0}^{k-1} \frac{(-1)^b (k-b)^n c(k, b)}{k!} \right] \left( \sum_{i=1}^{m} \sum_{j=0}^{i-1} \frac{1}{i!} (-1)^{i-j} (i-j)^k c(i, j) \right),$$

where $l = \lceil \sum_{i=1}^{n} s_i / B \rceil$ is the minimum number of required batches to group the jobs ($\lceil x \rceil$ denotes the smallest integer larger than or equal to $x$) and $c(p, q)$ is the number of combinations of $p$ taken $q$ at a time.

The above relation gives the number of all possible schedules by enumerating all the batching plans, considering this point that for each batching plan containing $k$ batches, we should enumerate all the ways of assigning its batches to $m$ machines.

As can be seen, increasing the problem size, results in exponential growth of the number of possible schedules; therefore obtaining the optimal solution(s) becomes impractical for large-scale problems.

4. A lower bound on the optimal makespan

In this section a lower bound on the optimal makespan is proposed. The role of the lower bound is to use it as a reference for evaluating the performance results of our HGH approach and also the performance of SA comparator approach.

For the case of parallel identical processors with unit capacity, formula (2) gives the minimum makespan when the preemption is allowed [29]:

$$C_{\text{max}}^p = \max \left\{ \sum_{i=1}^{n} \frac{p_i}{m}, \max_{i=1, \ldots, n} \{ p_i \} \right\}.$$

$$\sum_{k=1}^{n} \left[ \frac{1}{k!} \sum_{b=0}^{k-1} \frac{(-1)^b (k-b)^n c(k, b)}{k!} \right] \left( \sum_{i=1}^{m} \sum_{j=0}^{i-1} \frac{1}{i!} (-1)^{i-j} (i-j)^k c(i, j) \right),$$
The formula states that either the work will be allocated evenly among the machines, or the length of the longest job will determine the makespan. If we prohibit job preemption, then the above $C_{\text{max}}^P$ is a lower bound on the optimal makespan for the scheduling problem with unit capacity parallel machines. Now we propose the following lower bound on the optimal makespan related to our problem:

**Lower bound**

**BEGIN**

1. Put the jobs satisfying following relation in the set $J$, and remove them from the set of whole jobs and assign each of them to an individual batch,

$$J = \left\{ q \mid B - s_q < \min_{i \in \{1, \ldots, n\}} \{s_i\} \right\}. \quad (3)$$

2. For the reduced problem construct an instance with unit job sizes where each job $g$ is replaced by $s_g$ jobs of unit size and processing time $p_g$. List these jobs in decreasing order of their processing time and successively group the $B$ jobs with longest processing times into the same batch. For the current batching plan with $l = \lceil \sum_{i \in J} s_i / B \rceil$ number of batches, compute the relation:

$$CS = \sum_{k=1}^{l} P_k + \sum_{i \in J} p_i,$$  \quad (4)

where $P_k$ is the processing time of batch $k$.

3. The lower bound on the optimal makespan ($C_{\text{LB}}^{\text{max}}$) can be obtained as follows:

$$C_{\text{LB}}^{\text{max}} = \max \left\{ \lceil CS / m \rceil, \max_{i=1, \ldots, n} \{p_i\} \right\}. \quad (5)$$

**END**

This bound can be further improved by ordering the batches resulted in steps 1 and 2 in decreasing order of their processing time and considering that: $C_{\text{max}}^{\text{Optimal}} \geq P_m + P_{m+1}$, where $P_m$ is the processing time of the $m$th batch in the ordering. Then, it is possible to tighten the bound as follows:

$$C_{\text{LB}}^{\text{max}} = \max \left\{ \lceil CS / m \rceil, \max_{i=1, \ldots, n} \{p_i\}, P_m + P_{m+1} \right\}. \quad (6)$$

Contrary to the unit capacity counterpart, for the case of parallel batch processing machines, for each feasible batching plan, Eq. (2) can be applied to get the related lower bound, by considering each batch as an individual job. So, there may be many lower bounds with respect to existence of many feasible batching planes. Since $CS$ is a lower limit for the summation of batch processing times, to ensure reaching a true lower bound, we allow the jobs to be split among batches. However to get a more effective bound, splitting the jobs that cannot be grouped with any job in a same batch is not allowed.

5. Proposed heuristic

Considering the above lower bound, as the problem size increases, it is expected for $C_{\text{LB}}^{\text{max}}$ to be equal to $\lceil CS / m \rceil$. This implies that in the batching phase, a batching plan with smaller summation of batch processing times would probably result in a smaller makespan in scheduling phase. Hence, it can be inferred that grouping jobs with longer processing times in the same batch would be desirable (this follows directly from the optimal solution to the unit size problem).

For the sake of grouping longer processing time jobs in the same batch, we propose the following heuristic named random batches procedure (RBP), which constructs a feasible batching plan through simultaneously minimizing the
residual batch capacity. The RBP heuristic corresponded to the batching phase is as follows:

**RBP heuristic**

**BEGIN**

1. Assign all jobs to \( L \) \( (L \leqslant \lceil \sum_{i=1}^{n} s_i / B \rceil) \) batches randomly without considering the capacity restriction;
2. **IF** the derived batching plan is feasible **THEN** stop; **ELSE**
3. **REPEAT**
   - Choose a batch with capacity violation and the longest batch processing time. Select the job with the longest processing time in it;
4. **IF** the selected job fits in any existing batch (consider these batches as set \( k \)) **THEN**
   - **IF** among batches in \( k \), there are some batches having the batch processing time longer than the processing time of the selected job (batches \( r \subseteq k \)) **THEN** put the selected job in one of batches in \( r \) with the smallest residual capacity; **ELSE** put the job in a feasible batch in \( k \) with the longest processing time;
   - **END IF**
5. **ELSE** create a new batch and assign it the selected job;
   - **END IF**
**UNTIL** getting a feasible batching plan

**END**

The effect of the randomness included in the derived batching plan by RBP will be further highlighted by increasing the initial number of batches \( (L) \). So starting RBP with small number of batches would be desirable. The above heuristic tries to give a batching plan with smaller summation of batch processing times. To schedule the batches resulted by RBP (scheduling phase), the following step could be added to RBP:

6. **Assign jobs to machines in arbitrary order.**

   Arranging batches in any order may cause the schedule to be ineffective, even though the batching is done effectively. Since the longest processing time (LPT) rule guarantees very good result for minimizing makespan on parallel identical machines with unit capacity [23], we refine the above step as follows:

6. **List the batches in LPT order (considering each batch as a job) and assign them to machines successively, whenever a machine becomes free.**

**6. Proposed HGH and its implementation**

GA are powerful and broadly applicable stochastic search and optimization techniques based on principles of evolution theory. In the past few years, GA have received considerable attention for solving difficult combinatorial optimization problems [30].

We propose a HGH that uses the heuristic approaches to find a superior schedule. The HGH searches the solutions space via generating random batches of jobs using GA operators. For each generated offspring chromosome, the RBP is applied to ensure feasibility, and then for a feasible batching plan the LPT rule is applied to schedule the batches on machines effectively (once a capacity feasible batching plan is achieved, the problem reduces to minimizing \( C_{\text{max}} \) on parallel identical processors while viewing each batch as a job). To improve effectiveness of HGH, two local search heuristics are developed based on the problem characteristics for both batching and scheduling phases. The following steps describe in detail how the HGH is implemented in this research.

**6.1. Coding**

In HGH, a solution for the problem of assigning jobs to batches is an array whose length is equal to the number of jobs to be scheduled. Each gene represents the batch to which the job is assigned. Fig. 1 shows the chromosome representation of HGH for a batch-scheduling problem with five jobs.
6.2. Initial population

As stated before, the performance of RBP depends on the number of initial batches, where starting RBP with large number of initial batches would be unfavorable. However, starting RBP with small number of initial batches may cause premature convergence to a local optimum. To cope with poor quality of initial seeds and to avoid premature convergence, we use a truncated geometric distribution to generate randomly the number of initial batches used for RBP. Using a geometric distribution to simulate the number of initial batches ensures that the probability of starting RBP with a large number of batches would be small compared to the high probability for starting with small number of batches. The following relation gives the random number of initial batches to start with RBP:

\[
L = \left\lceil \frac{\ln(1 - (1 - (1 - p)^{l-1})R)}{\ln(1 - p)} \right\rceil + 1, \quad L \in \{2, 3, \ldots, l\},
\]

where \(L\) is a random number of initial batches for starting RBP, distributed by a truncated geometric distribution, \(R \in [0, 1]\), \(p\) is the probability of success and \(l\) is the minimum number of required batches.

The above approach for generating the initial population in HGH has some advantages, since it uses efficiently the knowledge of the problem to group the jobs with longer processing times in the same batches as long as possible. One can adjust the quality of initial population by choosing a proper value of \(p\). Setting a high value for \(p\), seems to be effective on accelerating the convergence rate through a good quality initial seeds but may cause premature convergence to a local optimum. Low value of \(p\) causes to less-accelerate converging due to the relatively low quality of initial population. So a tuned value of \(p\) would be drastic to exploit advantages of such a robust mechanism of generating initial population. Based on our primary experiments we found 0.1 to be appropriate value for \(p\), for all problem instances.

6.3. Selection

For selecting the chromosomes, the tournament selection rule is followed. In this rule two individuals are chosen randomly, and then the fittest one will be chosen if a random value \(r\) is smaller than a probability value \(k\), otherwise the other one is chosen. Selected individuals are returned to the population and can be chosen again as a parent. Here, we use a value equal to 0.7 for the probability value.

6.4. Crossover

The parameterized uniform crossover of Bean [31] is utilized in which a bias coin is used to determine which of two selected parents should pass its gene. Using parameterized uniform crossover allows us to bias the search more strongly toward components of better solutions and to accelerate convergence. Due to our representation it may happen to lose some of batches in the generated offspring chromosomes. In this case we rename the batches of the offspring chromosome as it is shown in Fig. 2. For any infeasible generated offspring chromosome, the RBP is applied to ensure the feasibility. In this case each generated offspring chromosome indicates an initial batching plan, which is infeasible. Then we ensure the feasibility through using the RBP heuristic.
6.5. Mutation

The swapping mutation is used as mutation operator. It selects two random genes of a selected chromosome and swaps their content. The feasibility of the mutated schedule is kept through using the RBP heuristic. Fig. 3 shows the swap mutation procedure applied for HGH.

6.6. Local search heuristics

To improve the effectiveness of HGH, we developed two local search heuristics, based on the problem characteristics. The first heuristic, which is used for the batching phase, ensures reduction in summation of batch processing times corresponding to a batching plan by pairwise swapping jobs between batches. For a given batching plan in scheduling phase, our second heuristic tries to reduce makespan through suitable pairwise interchange of batches between machines. We name the corresponding local search heuristic to batching and scheduling phases as LSHBP and LSHSP, respectively. Both LSHBP and LSHSP are detailed as follows:

**LSHBP heuristic**

For any selected batching plan do:

**BEGIN**

1. Set \( k = 1 \);
2. Find the \( k \)th batch (name it batch \( a \)) that has only one job with maximum processing time;
3. IF there is not any batch \( a \)
   THEN consider the current batching plan as the result of LSHBP and go to 9;
   ELSE
4. Set \( u = 1 \);
5. Find the \( u \)th batch (name it batch \( b \)) with batch processing time equal or longer than the processing time of batch \( a \);
6. IF there is not any batch \( b \)
   THEN set \( k = k + 1 \) and go to 2;
   ELSE

7. IF there is a job \( j^b \) in \( b \) such that: \( p_{j^a} > p_{j^b}, p_{j^a} = \max_{i|job \ i \ in \ batch \ a} \{p_i\} \)
   THEN
8. **IF** swapping jobs $j^b$ and $j^a$ between batches $a$ and $b$ does not violate the capacity restriction
   **THEN** Swap the jobs $j^b$ and $j^a$ between batches $a$ and $b$, then go to 1;
   **ELSE** set $u = u + 1$ and go to 5;
   **END IF**
   **ELSE** set $u = u + 1$ and go to 5;
   **END IF**
   **END IF**

9. List the batches in LPT order (considering each batch as a job) and then assign them to machines successively, whenever a machine becomes free;
END

For a given batching plan in an iteration of LSHBP, swapping the jobs $j^a$ and $j^b$ causes decreasing the processing time of batch $a$ without any change in processing time of batch $b$. Therefore, the swapping ensures improvement in the summation of batch processing times.

In problems with small size jobs, swapping the jobs may cause less capacity violation comparing to other problems. Therefore LSHBP will have more latitude with this kind of problems and can be significantly effective to direct the search to the optimal solution(s). However, its effect on the other problem categories should not be disregard, especially for mixed size jobs problems (see Section 7).

To decrease the computation time spent by LSHBP, only a small number of generated batching plans based on their fitness ranking, should be selected to be improved. So in each generation of HGH, the LSHBP is applied on the selected chromosomes based on a predefined rate. We found 0.25 to be an appropriate rate value by our primary experiments.

**LHSP heuristic**
For each schedule do:
BEGIN
1. Arrange the machines in decreasing order of their finish time $(FT)$, where the finish time is defined by the time in which the processing on the last batch by the machine is completed;
2. **IF** all machines have the same finish time
   **THEN** stop; the optimal makespan corresponding to the current schedule is achieved.
   **ELSE** set $i = 1$; $j = m$; and $k = 0$;
   **END IF**
3. Select the $i$th machine and name it $m_i$. Considering the $j$th machine (name it $m_j$), find the batches $a \in m_i$ and $b \in m_j$ (the $\in$ denotes the assignment of the batch to machine) such that: $0 < P_a - P_b < FT_i - FT_j$ ($P_a$ and $P_b$ are the processing time of batches $a$ and $b$, respectively);
4. **IF** there is not any pair of $a$ and $b$
   **THEN** set $j = j - 1$; and $k = k + 1$;
5. **IF** $k = m - (i + 1)$
   **THEN** set $i = i + 1$; and $j = m$;
6. **IF** $i = m$
   **THEN** stop; No more improvement in $C_{max}$ of the schedule is possible.
   **ELSE** go to 3;
   **END IF
   **ELSE** go to 3,
5. **END IF**
   **ELSE** Interchange batches $a$ and $b$ between $m_i$ and $m_j$ and go to 1;
   **END IF**
END

Effectiveness of proposed LSHBP and LSHSP has been experienced on an extensive set of randomly generated instances. Tables 1 and 2, summarize the performance of LSHBP and LSHSP, respectively. The performance of each heuristic has been estimated based on the average ratio of 100 randomly generated feasible batching plans for each
of the six problem categories (see Section 7) combined with four levels for the number of jobs and two levels for the number of machines. Also each combination was replicated 30 times with different seed data to create different instances. In Table 2 the first column for each category with different number of machines (columns 3 and 7) shows the $C_{\text{max}}$ related to a list-scheduling algorithm, in which the batches are sorted in arbitrary order and are assigned to machines successively, whenever a machine becomes free. The next columns (column 4 and 8) show the improved $C_{\text{max}}$ of the corresponded batches order, by the LSHSP. The results for the LPT ordered batches and the improved cases are reported in columns 5, 9 and 6, 10, respectively. It should be noted that the performance in Table 1 is measured based on the average percentage of improvement (the average difference between the ratio of a randomly generated batching plan and the ratio of the improved batching plan by LSHBP respect to $C_S$) resulted by LSHBP, while the performance in Table 2 is measured based on the ratio respect to $C_{\text{max}}$.

As can be seen, both heuristics are significantly effective and can perform very well as a complement part of any constructive heuristic such as RBP.
In each generation of HGH, the new population is formed by the offspring generated by GA operators. The rest of the population is filled with the best chromosomes of the former generation.

6.7. Parameter tuning

For each GA there are a number of parameters which choosing a proper value for them affects the search behavior and improves the quality of convergence. These parameters include the population size, number of generations, crossover rate, mutation rate and the head probability of the bias coin. For tuning crossover rate, mutation rate and the head probability, different values are considered; 0.5 and 0.7 for crossover rate, 0.05, 0.1 and 0.15 for mutation rate and 0.7 and 0.9 for head probability. Based on the primary experiments on HGH, the appropriate values found to be 0.5, 0.15 and 0.9 for crossover rate, mutation rate and the head probability, respectively. About the population size and stopping criteria, we evaluated the pairs \((n, 50)\) and \((2n, 100)\), where the first member of these ordered pairs denotes the population size (which increases by the problem size) and the second one indicates the number of generations. The results for the pair of \((2n, 100)\) are slightly better in average performance (while there is not any difference in the best and the worst-case performance) at the expense of significant increase in running times. So, it is preferred to keep the pair \((n, 50)\) to provide good results together with saving of the execution time. However the HGH will be stopped if there is no improvement in the best solution obtained in last 15 generations. Worth to mention that to avoid useless computations in both HGH and SA, we set an extra stopping criterion that is getting to the lower bound solution explained in Section 4.

The general framework of the proposed HGH is described in the following procedure:

**Algorithm HGH**

**BEGIN**

% Initial population:

FOR each chromosome of initial population

1. Simulate the number of initial batches using Eq. (7);
2. Apply RBP to reach a feasible batching plan;
3. Assign batches to machines using LPT rule;
4. Apply LSHSP on the current schedule to make further improvement;

END FOR

% Crossover

5. Set crossover_offspring = 1;

REPEAT

6. Crossing over the selected parents using parameterized uniform crossover to get the offspring;
7. For the generated offspring, rename the batches in a numerical order;
8. Ensure feasibility of the generated offspring, using steps 2 and 3 of RBP;
9. Assign batches to machines using LPT rule;
10. Apply LSHSP on the current schedule to make further improvement;
11. Set crossover_offspring = crossover_offspring + 1;
    UNTIL crossover_offspring < (0.5 * n)
% Mutation

12. Set mutation_offspring = 1;

REPEAT

13. Mutate the selected parent using swap mutation to get the offspring;
14. Ensure feasibility of the generated offspring, using steps 2 and 3 of RBP;
15. Assign batches to machines using LPT rule;
16. Apply LSHSP on the current schedule to make further improvement;
17. Set mutation_offspring = mutation_offspring + 1;
    UNTIL mutation_offspring < (0.15 * n)
18. Migrate the top \((0.1 * n)\) chromosomes of the former generation to the current population;
% LSHBP

19. Sort all chromosomes in increasing order of their fitness (delete the repeated chromosomes);
Table 3
Factors and levels

<table>
<thead>
<tr>
<th>Factors</th>
<th>Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>2, 4</td>
</tr>
<tr>
<td></td>
<td>J</td>
</tr>
<tr>
<td></td>
<td>p</td>
</tr>
<tr>
<td></td>
<td>s</td>
</tr>
</tbody>
</table>

Table 4
Upper bound for the search space of test problem instances

<table>
<thead>
<tr>
<th>Job size</th>
<th>n = 10</th>
<th>n = 20</th>
<th>n = 50</th>
<th>n = 100</th>
</tr>
</thead>
<tbody>
<tr>
<td>m = 2</td>
<td>1.89E6</td>
<td>6.12E15</td>
<td>1.72E50</td>
<td>2.84E116</td>
</tr>
<tr>
<td>m = 4</td>
<td>1.32E7</td>
<td>1.14E18</td>
<td>7.11E56</td>
<td>3.21E130</td>
</tr>
<tr>
<td>s1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s2</td>
<td>2.20E6</td>
<td>1.19E16</td>
<td>2.23E52</td>
<td>3.95E124</td>
</tr>
<tr>
<td>s3</td>
<td>1.21E6</td>
<td>1.15E15</td>
<td>5.65E46</td>
<td>3.80E106</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.38E7</td>
<td>1.33E18</td>
<td>3.63E57</td>
<td>1.97E134</td>
</tr>
<tr>
<td></td>
<td>1.11E7</td>
<td>5.53E17</td>
<td>5.88E54</td>
<td>3.93E123</td>
</tr>
</tbody>
</table>

20. Set LSHBP_offspring = 1;

REPEAT
21. Apply LSHBP on the LSHBP_offspring chromosome of the sorted population to get the improved offspring;
22. Assign batches to machines using LPT rule;
23. Apply LSHP on the current schedule to make further improvement;
24. Set LSHBP_offspring = LSHBP_offspring + 1;
   UNTIL LSHBP_offspring < (0.25 * n)
   % Selection
25. From the current population, select n chromosomes using tournament selection technique (n = population_size);
   UNTIL stopping criteria is true.

7. Experimentations and results

For testing effectiveness of proposed HGH, test problem instances are randomly generated in a manner similar to Chang et al. [28] (in addition we have also considered instances with 100 jobs). For covering various types of problems, four factors are identified: number of jobs, number of machines, variation in job processing times, and variation in job sizes. In general six categories of problems are generated by combining three levels of job sizes (small, large and combination of small and large (mixed)) and two levels of processing times. Processing times and job sizes are generated from discrete uniform distribution. The factors, their levels and ranges, are shown in Table 3. The machine capacity is assumed to be 10 in all instances. Table 4 shows the search space related to the test problem instances as a function of problem size and average number of required batch.

Each category of problems is represented with a run code. For example a problem category with 10 jobs, four batch processing machines, processing times generated from the discrete uniform [1,20] and job sizes generated from the discrete uniform [4,8] is shown by J1m2p2s3.

All the algorithms HGH, SA, and the lower bound, are coded in MATLAB 6.5.1 and all test problem instances are solved 15 times by HGH and SA on a Pentium III, 800 MGz computer with 128 MB RAM.

Table 5 represents the results for 10 jobs instances. Columns 2, 3 and 4, report the best, average and the worst performance against the lower bound among the 15 runs of HGH, respectively. Column 5 reports the average time taken by HGH. These values are reported in columns 6, 7, 8 and 9 for SA, respectively. Tables 6–8 can be interpreted in a same manner. For each algorithm, the performance is measured by 
\[
\left( \frac{C_{\text{max}}^{\text{SA}}(\text{HGH}) - C_{\text{LB}}^{\text{max}}}{C_{\text{LB}}^{\text{max}}} \right) \times 100.
\]

Column 10 compares HGH with SA in terms of solutions quality. The content of this column implies how much improvement HGH obtains over SA.
### Table 5
Results for instances with 10 jobs (comparing to the lower bound)

<table>
<thead>
<tr>
<th>Run code</th>
<th>HGH</th>
<th></th>
<th></th>
<th></th>
<th>SA</th>
<th></th>
<th></th>
<th></th>
<th>Avg. diff (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>J1m1p1s1</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.23</td>
<td>0.00</td>
<td>5.16</td>
<td>11.11</td>
<td>1.82</td>
<td>5.16</td>
</tr>
<tr>
<td>J1m1p1s2</td>
<td>0.00</td>
<td>2.75</td>
<td>8.33</td>
<td>0.47</td>
<td>0.00</td>
<td>13.91</td>
<td>33.33</td>
<td>1.19</td>
<td>11.16</td>
</tr>
<tr>
<td>J1m1p1s3</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.23</td>
<td>0.00</td>
<td>2.13</td>
<td>9.09</td>
<td>0.71</td>
<td>2.13</td>
</tr>
<tr>
<td>J1m1p2s1</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.14</td>
<td>0.00</td>
<td>8.14</td>
<td>25.92</td>
<td>1.62</td>
<td>8.14</td>
</tr>
<tr>
<td>J1m1p2s2</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.17</td>
<td>0.00</td>
<td>11.00</td>
<td>35.00</td>
<td>0.61</td>
<td>11.00</td>
</tr>
<tr>
<td>J1m1p2s3</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.13</td>
<td>0.00</td>
<td>6.27</td>
<td>15.15</td>
<td>1.49</td>
<td>6.27</td>
</tr>
<tr>
<td>J1m2p1s1</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.10</td>
<td>0.00</td>
<td>6.70</td>
<td>20.00</td>
<td>0.76</td>
<td>6.70</td>
</tr>
<tr>
<td>J1m2p1s2</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.07</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.01</td>
<td>0.00</td>
</tr>
<tr>
<td>J1m2p1s3</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.12</td>
<td>0.00</td>
<td>8.37</td>
<td>14.28</td>
<td>2.03</td>
<td>8.57</td>
</tr>
<tr>
<td>J1m2p2s1</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.18</td>
<td>0.00</td>
<td>7.52</td>
<td>13.04</td>
<td>1.82</td>
<td>7.52</td>
</tr>
<tr>
<td>J1m2p2s2</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.06</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.01</td>
<td>0.00</td>
</tr>
<tr>
<td>J1m2p2s3</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.14</td>
<td>0.00</td>
<td>5.00</td>
<td>8.33</td>
<td>2.25</td>
<td>5.00</td>
</tr>
</tbody>
</table>

### Table 6
Results for instances with 20 jobs (comparing to the lower bound)

<table>
<thead>
<tr>
<th>Run code</th>
<th>HGH</th>
<th></th>
<th></th>
<th></th>
<th>SA</th>
<th></th>
<th></th>
<th></th>
<th>Avg. diff (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>J2m1p1s1</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.64</td>
<td>0.00</td>
<td>6.85</td>
<td>17.64</td>
<td>4.67</td>
<td>6.85</td>
</tr>
<tr>
<td>J2m1p1s2</td>
<td>0.00</td>
<td>0.65</td>
<td>5.00</td>
<td>1.21</td>
<td>5.00</td>
<td>11.35</td>
<td>30.00</td>
<td>3.42</td>
<td>10.7</td>
</tr>
<tr>
<td>J2m1p1s3</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.54</td>
<td>2.22</td>
<td>6.22</td>
<td>8.88</td>
<td>7.76</td>
<td>6.22</td>
</tr>
<tr>
<td>J2m1p2s1</td>
<td>1.56</td>
<td>1.96</td>
<td>4.68</td>
<td>3.62</td>
<td>6.25</td>
<td>10.82</td>
<td>15.62</td>
<td>6.70</td>
<td>8.86</td>
</tr>
<tr>
<td>J2m1p2s2</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.58</td>
<td>2.63</td>
<td>11.39</td>
<td>15.78</td>
<td>2.36</td>
<td>11.39</td>
</tr>
<tr>
<td>J2m1p2s3</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.98</td>
<td>1.38</td>
<td>4.62</td>
<td>8.33</td>
<td>6.81</td>
<td>4.62</td>
</tr>
<tr>
<td>J2m2p1s1</td>
<td>6.25</td>
<td>6.25</td>
<td>6.25</td>
<td>3.95</td>
<td>12.50</td>
<td>16.68</td>
<td>31.25</td>
<td>6.59</td>
<td>10.43</td>
</tr>
<tr>
<td>J2m2p1s2</td>
<td>0.00</td>
<td>0.46</td>
<td>7.69</td>
<td>0.72</td>
<td>0.00</td>
<td>11.76</td>
<td>23.07</td>
<td>2.23</td>
<td>11.3</td>
</tr>
<tr>
<td>J2m2p1s3</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.57</td>
<td>4.34</td>
<td>4.34</td>
<td>4.34</td>
<td>8.47</td>
<td>4.34</td>
</tr>
<tr>
<td>J2m2p2s1</td>
<td>2.85</td>
<td>2.85</td>
<td>2.85</td>
<td>4.81</td>
<td>5.71</td>
<td>9.51</td>
<td>17.14</td>
<td>7.07</td>
<td>6.66</td>
</tr>
<tr>
<td>J2m2p2s2</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.38</td>
<td>2.10</td>
<td>12.50</td>
<td>26.08</td>
<td>2.90</td>
<td>9</td>
</tr>
<tr>
<td>J2m2p2s3</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.97</td>
<td>2.32</td>
<td>4.18</td>
<td>9.30</td>
<td>9.05</td>
<td>4.18</td>
</tr>
</tbody>
</table>

### Table 7
Results for instances with 50 jobs (comparing to the lower bound)

<table>
<thead>
<tr>
<th>Run code</th>
<th>HGH</th>
<th></th>
<th></th>
<th></th>
<th>SA</th>
<th></th>
<th></th>
<th></th>
<th>Avg. diff (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>J3m1p1s1</td>
<td>1.36</td>
<td>2.27</td>
<td>4.10</td>
<td>27.84</td>
<td>5.47</td>
<td>8.58</td>
<td>10.95</td>
<td>27.50</td>
<td>6.31</td>
</tr>
<tr>
<td>J3m1p1s2</td>
<td>0.00</td>
<td>0.29</td>
<td>2.27</td>
<td>5.14</td>
<td>2.27</td>
<td>6.36</td>
<td>11.36</td>
<td>9.24</td>
<td>6.07</td>
</tr>
<tr>
<td>J3m1p1s3</td>
<td>2.02</td>
<td>2.15</td>
<td>4.04</td>
<td>21.55</td>
<td>2.02</td>
<td>5.15</td>
<td>10.10</td>
<td>42.23</td>
<td>3.00</td>
</tr>
<tr>
<td>J3m1p2s1</td>
<td>4.00</td>
<td>4.84</td>
<td>6.00</td>
<td>33.49</td>
<td>6.66</td>
<td>9.73</td>
<td>11.33</td>
<td>30.95</td>
<td>4.89</td>
</tr>
<tr>
<td>J3m1p2s2</td>
<td>0.00</td>
<td>0.79</td>
<td>2.40</td>
<td>14.55</td>
<td>4.81</td>
<td>8.83</td>
<td>14.45</td>
<td>10.56</td>
<td>8.04</td>
</tr>
<tr>
<td>J3m1p2s3</td>
<td>0.52</td>
<td>0.59</td>
<td>1.05</td>
<td>24.64</td>
<td>1.05</td>
<td>3.01</td>
<td>4.76</td>
<td>44.91</td>
<td>2.42</td>
</tr>
<tr>
<td>J3m2p1s1</td>
<td>2.38</td>
<td>3.00</td>
<td>4.76</td>
<td>35.50</td>
<td>7.14</td>
<td>10.16</td>
<td>11.90</td>
<td>34.08</td>
<td>7.16</td>
</tr>
<tr>
<td>J3m2p1s2</td>
<td>0.00</td>
<td>2.09</td>
<td>4.54</td>
<td>8.96</td>
<td>9.09</td>
<td>10.90</td>
<td>22.72</td>
<td>10.46</td>
<td>8.81</td>
</tr>
<tr>
<td>J3m2p1s3</td>
<td>2.08</td>
<td>2.35</td>
<td>4.16</td>
<td>32.35</td>
<td>4.16</td>
<td>5.41</td>
<td>8.98</td>
<td>42.25</td>
<td>2.63</td>
</tr>
<tr>
<td>J3m2p2s1</td>
<td>2.50</td>
<td>3.50</td>
<td>3.75</td>
<td>33.39</td>
<td>7.50</td>
<td>11.41</td>
<td>13.75</td>
<td>30.80</td>
<td>7.91</td>
</tr>
<tr>
<td>J3m2p2s2</td>
<td>0.00</td>
<td>1.00</td>
<td>2.50</td>
<td>10.26</td>
<td>5.00</td>
<td>8.50</td>
<td>12.50</td>
<td>11.08</td>
<td>7.50</td>
</tr>
<tr>
<td>J3m2p2s3</td>
<td>2.24</td>
<td>2.98</td>
<td>4.49</td>
<td>33.06</td>
<td>3.37</td>
<td>5.61</td>
<td>8.98</td>
<td>42.25</td>
<td>2.63</td>
</tr>
</tbody>
</table>
Table 8
Results for instances with 100 jobs (comparing to the lower bound)

<table>
<thead>
<tr>
<th>Run code</th>
<th>HGH</th>
<th>SA</th>
<th>Avg. diff (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Best $C_{\text{max}}$</td>
<td>Avg. $C_{\text{max}}$</td>
<td>Worst $C_{\text{max}}$</td>
</tr>
<tr>
<td>J4m1p1s1</td>
<td>1.35</td>
<td>1.89</td>
<td>2.02</td>
</tr>
<tr>
<td>J4m1p1s2</td>
<td>0.00</td>
<td>0.42</td>
<td>1.06</td>
</tr>
<tr>
<td>J4m1p1s3</td>
<td>1.54</td>
<td>1.64</td>
<td>2.57</td>
</tr>
<tr>
<td>J4m1p2s1</td>
<td>3.61</td>
<td>3.72</td>
<td>3.97</td>
</tr>
<tr>
<td>J4m1p2s2</td>
<td>0.00</td>
<td>0.69</td>
<td>1.22</td>
</tr>
<tr>
<td>J4m1p2s3</td>
<td>2.05</td>
<td>2.17</td>
<td>2.56</td>
</tr>
<tr>
<td>J4m2p1s1</td>
<td>1.56</td>
<td>2.59</td>
<td>3.12</td>
</tr>
<tr>
<td>J4m2p1s2</td>
<td>0.00</td>
<td>0.65</td>
<td>2.50</td>
</tr>
<tr>
<td>J4m2p1s3</td>
<td>3.09</td>
<td>3.50</td>
<td>4.12</td>
</tr>
<tr>
<td>J4m2p2s1</td>
<td>2.89</td>
<td>3.23</td>
<td>4.34</td>
</tr>
<tr>
<td>J4m2p2s2</td>
<td>0.00</td>
<td>0.32</td>
<td>1.25</td>
</tr>
<tr>
<td>J4m2p2s3</td>
<td>1.56</td>
<td>1.91</td>
<td>2.08</td>
</tr>
</tbody>
</table>

Computational analysis shows that in all categories of test problem instances, HGH performs better than the SA. Especially its superiority over SA has been proved in both cases of the average and the worst-case performance. Regarding the best case performance, no solution is reported by SA that is better than its counterpart reported by HGH, and the best solutions reported by HGH are at least as good as those reported by SA.

As can be seen in the resulting tables, for all problems with small jobs (J.m.p.s2), where the search space is larger than the other problems, HGH achieved the optimal makespan. As was mentioned before, for these problems, jobs swapping can be more easily handled by the LSHBP, so converging to the optimal makespan may be best achieved through the hybridizing LSHBP with HGH. For the problems with mixed size jobs (J.m.p.s1) results show the better performance for HGH compared to SA. Similar to the case with small size jobs, in this case the differences in objective values are still meaningfully large. Because of the diversity in the job sizes, effective use of the batch capacity in assigning the jobs to batches is an important factor that is satisfied by RBP for HGH. The least difference in objective values exists for the problems with large size jobs (J.m.p.s3). In this case, as can be seen from Table 4, the search space is significantly smaller compared to other problems. It is expected that for these problems, 40% of individual jobs lie in exactly one batch (the jobs with sizes 7 and 8) and yet the 60% of jobs need to be assigned efficiently to batches to decrease the makespan. This causes reduction in the differences between objective values of HGH and SA.

As a performance criterion, it is desirable to take into consideration the amount of variation between the worst and the best performance of each algorithm. For each problem instance the SA reports wide range of variation (due to big difference between the worst and the best cases) where the results for HGH show its concentration on an exact value that indicates its good quality of convergence.

Comparing the required running time for the algorithms, the tables show different results for different problem sizes. For all problems with 10 and 20 jobs instances the times taken by HGH are almost trivial compared to SA running times. This can be related to the high rate of convergence in HGH because of its effective components. So in these cases, our second stopping criterion that is getting to the lower bound solution is fired. Therefore the HGH’s needed time for getting to the lower bound solution will be small compared to SA. The major part of the run time in HGH belongs to LSHBP, which increases as the problem size increases. However by eliminating LSHBP, still HGH may be able to outperform SA, but at the expense of the quality performance. We use a population size proportional to the problem size for HGH where the SA uses a fixed value for its parameters (such as the number of temperature decreasing times and also the number of neighboring generation at each temperature) for all problem sizes. This may cause longer execution time for HGH comparing to SA in some large size problem instances.

8. Conclusions and future research directions

In this paper we considered the problem of minimizing makespan on $m$ identical parallel batch processing machines with non-identical job sizes. The processing time of a batch is given by the longest processing time (LPT) of the
jobs in the batch. To examine the problem search space, we developed an enumerative formula as a function of both problem size \((n)\) and the least number of required batches \((l)\). We proposed a hybrid genetic heuristic (HGH) based on a batch-based representation. Our computational results show that the HGH outperforms a simulated annealing (SA) approach, taken from the literature as a comparator algorithm. Also its good performance comparing to a developed lower bound is indicated.

Some characteristics such as using a robust mechanism for generating initial population, using an efficient local search heuristic (LSHPBP) to bring longer jobs together as a batch, which has the ability of steering quickly the search toward the optimal solution(s) and using a local search heuristic (LSHSP) to further improve machines load, are the cases that cause HGH to dominate the SA approach.

For future research, the extension of our approach could include due date related performance measures and also considering dynamic job arrivals and incompatible families.

References


