Two-agent single-machine scheduling with release times and deadlines

Yunqiang Yin

State Key Laboratory Breeding Base of
Nuclear Resources and Environment,
East China Institute of Technology,
Nanchang, 330013, China
E-mail: yunqiangyin@gmail.com

Shuenn-Ren Cheng

Graduate Institute of Business Administration,
Cheng Shiu University, Kaohsiung County, Taiwan
No. 840, Chengcing Rd., Niaosong Dist.,
Kaohsiung City 83347, Taiwan
E-mail: tommy@csu.edu.tw

T.C.E. Cheng

Dean's Office,
Faculty of Business,
The Hong Kong Polytechnic University,
11 Yuk Choi Road, Hung Hom, Kowloon, Hong Kong
E-mail: Edwin.Cheng@inet.polyu.edu.hk

Wen-Hung Wu

Department of Business Administration,
Kang-Ning Junior College, Taipei, Taiwan
No. 137, Lane 75, Sec. 3, Kangning Rd.,
Neihu District, Taipei City 114, Taiwan
E-mail: wu410226@knjc.edu.tw

Chin-Chia Wu*

Department of Statistics,
Feng Chia University,
No. 100, Wenhwa Rd.,
Seatwen, Taichung, Taiwan
E-mail: cchwu@fcu.edu.tw
*Corresponding author

Abstract: Multiple-agent scheduling has attracted considerable research attention in recent years. However, studies of multiple-agent scheduling with release times and deadlines are few. In the presence of ready times, sometimes it is beneficial to wait for future job arrivals in constructing a schedule. Inspired by the importance of ready times, we study the single-machine two-agent scheduling problem with releases times and deadlines to minimise the number of tardy jobs of one agent under the restriction that the maximum lateness of the jobs of the other agent cannot exceed a given value Q. Having established that the problem is strongly NP-hard, we provide a branch-and-bound and a simulated annealing algorithm to search for the optimal and approximate solutions, respectively. The results of computational experiments reveal that the SA algorithm can generate near-optimal solutions quickly.

Keywords: scheduling; two agents; simulated annealing; release times.

Reference to this paper should be made as follows: Yin, Y., Cheng, S-R., Cheng, T.C.E., Wu, W-H. and Wu, C-C. (2013) 'Two-agent single-machine scheduling with release times and deadlines', *Int. J. Shipping and Transport Logistics*, Vol. 5, No. 1, pp.75–94.

Biographical notes: Yunqiang Yin received his BS, MS, and PhD from Shandong University of Science and Technology, Kunming University of Science and Technology, and Beijing Normal University, China, in 2003, 2006, and 2009, respectively. He has worked at East China Institute of Technology since 2009. His research covers semi-group theory, ring theory, module theory and their applications, and algebraic hyper-structure theory, fuzzy sets, rough sets and process scheduling. He has published more than 70 papers in these fields and written a book on fuzzy hemi-rings.

Shuenn-Ren Cheng received his BS in Statistics from Tung Hui University, MBA from St. John University in New York, and PhD from Manuel Quezon University. He is a Vice Professor of Cheng Shiu University, Taiwan. His areas of research include applied statistics and finance.

T.C.E. Cheng is Dean of the Faculty of Business and Chair Professor of Management of The Hong Kong Polytechnic University. He obtained his PhD and ScD from the University of Cambridge, England. His research interests are in operations management and scheduling.

Wen-Hung Wu is an Associate Professor at the Kang-Ning College of Medical Care and Management, Taiwan. He received his PhD in Management from Fu-Jen Catholic University, Taiwan. His present research includes management and scheduling.

Chin-Chia Wu is a Professor in the Department of Statistics, Feng Chia University, Taiwan. He received his Doctoral degree from the Graduate Institute of Management, School of Management, National Taiwan University of Science and Technology, Taiwan in 1997. His teaching and research interests include applied statistics and operations research.

1 Introduction

Multiple-agent scheduling has received considerable research attention since Baker and Smith (2003) and Agnetis et al. (2004) introduced the multi-agent concept to scheduling. For example, Cheng et al. (2006) showed that the feasibility model of single-machine multi-agent scheduling is a strongly NP-complete in general. Agnetis et al. (2007) determined the complexity of some single-machine multi-agent scheduling problems and developed solution algorithms for them. Cheng et al. (2008) studied the complexity of two models of single-machine multi-agent scheduling, namely the feasibility model and minimality model. Agnetis et al. (2009) applied a Lagrangian dual to obtain a good bound and solved all the considered problems in strongly polynomial time. Leung et al. (2010) generalised the results for some two-agent problems and solved one open problem involving identical parallel machines. For more results on multi-agent scheduling, the reader may refer to Yuan et al. (2005), Ng et al. (2006), Wan et al. (2010), Cheng et al. (2011a, 2011b), Liu et al. (2010, 2011), Mor and Mosheiov (2010, 2011), Nong et al. (2011), Li and Hsu (2012), and Yin et al. (2012), among others.

In the scheduling literature, studies involving due date-based objective functions, e.g., number of tardy jobs, and ready times are relatively limited. French (1982) points out that in some real-life applications, the penalty incurred by a late job does not depend on how late it is as a job that finishes a minute late might just as well be a century late. For instance, if an aircraft is scheduled to land at a time after which it will have exhausted its fuel, then the results are just as catastrophic whatever the scheduled landing time. In such cases, a reasonable objective would be to minimise the number of tardy jobs. On the other hand, generally each job has a different priority/weight, due date, and ready time. In the presence of ready times, sometimes it is beneficial to wait for future job arrivals in constructing a schedule. Despite multi-agent scheduling has become a popular research topic, study of multiple-agent scheduling with release times is relatively limited, especially involving the objective of minimising the number of tardy jobs. Inspired by these observations, we consider the two-agent single-machine scheduling problem with release times to minimise the number of tardy jobs of one agent with the restriction that the maximum lateness of the jobs of the other agent cannot exceed a given value.

An application of the problem arises in the shipping industry (Lun et al., 2011; Zhang et al., 2011). Ships from different shipping companies call at a port, which needs to determine the order in which it will serve the ships that arrive over time. In this context, the port is the single machine and the arriving ships are the jobs with ready times. Assume that the ships belong to two major shipping firms, which constitute the two agents. From the perspective of the port, it (the machine) wishes to find a schedule to serve (process) the ships of one of the two shipping firms (the jobs of the two agents) such that the number of tardy ships of one shipping firm is minimised, subject to the maximum lateness of the ships of the other shipping firm cannot exceed a given limit.

The rest of this paper is organised as follows: in Section 2, we introduce and formulate the problem under consideration. In Section 3, we show that the problem is strongly NP-hard while in Section 4, we show that two special cases of the problem are polynomially solvable. In Section 5, we present some dominance properties and a lower bound on the optimal solution, and exploit them to develop a branch-and-bound algorithm to solve the problem. In Section 6, we provide five variants of a simulated annealing (SA) algorithm to obtain approximate solutions for the problem. In Section 7, we report the results of extensive computational experiments conducted to assess the performance of all the proposed algorithms. We conclude the paper in the last section.

2 Model formulation

We introduce the scheduling problem considered in this paper as follows: Consider two competing agents, called agents *A* and *B*, respectively. Each of them has a set of non-preemptive jobs to be processed on a single machine. Jobs arrive dynamically and thus have unequal release times. Agent *A* has to execute the job set $J^A = \{J_1^A, J_2^A, \ldots, J_{n_A}^A\}$, whereas agent *B* has to execute the job set $J^B = \{J_1^B, J_2^B, \ldots, J_{n_B}^B\}$ Let $X \in \{A, B\}$. The jobs of agent *X* are called *X*-jobs. The processing time, due date, and release time of job J_j^X in the set J^X are positive integers p_j^X, d_j^X, and r_j^X, respectively, for all $j \in \{1, 2, \ldots, n_X\}$. Let *S* denote a feasible schedule of the $n_A + n_B$ jobs, i.e., a feasible assignment of starting times to the jobs of both agents. The completion time of job J_j^X is denoted as $C_j^X(S)$ and the lateness of job J_j^X is given by $L_j^X(S) = C_j^X(S) - d_j^X$. We write C_j^X and L_j^X for $C_j^X(S)$ and $L_j^X(S)$ respectively, whenever this does not cause confusion. We consider the scheduling problem to minimise the number of tardy jobs of agent *A*, subject to the maximum lateness of the jobs of agent *B* does not exceed a given value *Q*. Using the three-field notation scheme $\alpha \mid \beta \mid \gamma^A: \gamma^B$ introduced by Agnetis et al. (2004), we denote the problem by $1|r_j|\sum U_j^A : L_{\max}^B$, where U_j^A denotes whether or not job J_j^A is tardy with $U_j^A = 1$ if $C_j^A > d_j^A$ and $U_j^A = 0$ otherwise, and $L_{\max}^B = \max\{L_j^B \mid J_j^B \in J^B\}$.

3 NP-hardness of $1|r_j|\sum U_j^A : L_{\max}^B$

In this section, we prove that problem $1|r_j|\sum U_j^A : L_{\max}^B$ is strongly NP-hard by a reduction from 3-PARTITION.

3-PARTITION: Given a set of $3n$ positive integers $\{a_1, a_2, \ldots, a_{3n}\}$ and a positive integer *b* such that $\frac{b}{4} < a_j < \frac{b}{2}$, $j = 1, 2, \ldots, 3n$, $\sum_{i=1}^{3n} a_i = nb$, are there *n* pairwise disjoint three-element subsets S_i such that $\sum_{j \in S_i} a_j = b$ for $i = 1, 2, \ldots, n$?

Theorem 1: Problem $1|r_j|\sum U_j^A : L_{\max}^B$ is strongly NP-hard.

Proof: We reduce 3-PARTITION to problem $1|r_j|\sum U_j^A \le Q_1, L_{\max}^B \le Q_2$. Given an instance of 3-PARTITION, we construct an instance of problem $1|r_j|\sum U_j^A \le Q_1$, $L_{\max}^B \le Q_2$ as follows:

$$p_j^A = 2a_j, \quad j = 1, 2, \ldots, 3n$$

$$d_j^A = \left\lceil \frac{j}{3} \right\rceil (2b+1), \quad j = 1, 2, \ldots, 3n$$

$$r_j^A = 0, \quad j = 1, 2, \ldots, 3n$$

$$p_j^B = 1, \quad j = 1, 2, \ldots, n$$

$$d_j^B = j(2b+1), \quad j = 1, 2, \ldots, n$$

$$r_j^B = j(2b+1) - 1, \quad j = 1, 2, \ldots, n$$

$$Q_1 = Q_2 = 0.$$

It is easy to see that there is a solution to the 3-PARTITION instance if and only if there is a feasible schedule for the constructed instance of the scheduling problem.

4 Two polynomially solvable cases of $1|r_j|\sum U_j^A : L_{\max}^B$

Since problem $1|r_j|\sum U_j^A : L_{\max}^B$ is strongly NP-hard, it is of interest to identify some of its solvable special cases with a view to locating the exact boundary between the 'easy' and 'hard' problems. First, we consider the case where all the *A*-jobs have equal due dates, i.e., $r_j^A = 0$ for all *j* = 1, 2,…,*n*$_A$, denoted as $1|r_j^A = 0 : r_j^B|\sum U_j^A : L_{\max}^B$, and show that this case can be solved in $O(n_A \log n_A + n_B \log n_B)$ time by employing the idea of Agnetis et al. (2004) for solving problem $1|\ |\sum U_j^A : L_{\max}^B$.

For each *B*-job J_j^B, define a deadline D_j^B such that $C_j^B - d_j^B \le Q$ for $C_j^B \le D_j^B$ and $C_j^B - d_j^B > Q$ for $C_j^B > D_j^B$, i.e., $D_j^B - d_j^B = Q$. Re-arrange the *B*-jobs in non-decreasing order of D_j^B. Next, define the latest start time LS_j^B of job J_j^B as the maximum value of the starting time of J_j^B that permits a feasible schedule such that $C_j^B \le D_j^B$ for all $J_j^B \in J^B$. Starting from the last *B*-job $J_{n_B}^B$, set $LS_{n_B} = D_{n_B}^B - p_{n_B}^B$. Now, we consider the following cases.

- *Case 1:* $LS_{n_B} > D_{n_B-1}^B$. Then set $LS_{n_B-1} = D_{n_B-1}^B - p_{n_B-1}^B$.
- *Case 2:* If there is only one job $J_{n_B-1}^B$ such that $LS_{n_B} \le D_{n_B-1}^B$, then set $LS_{n_B-1} = LS_{n_B} - p_{n_B-1}^B$.
- *Case 3:* If there are more than one *B*-job whose deadlines are larger than LS_{n_B} and assume that *k* is the smallest index such that $LS_{n_B} \le D_k^B$, then order the jobs in $\{J_k^B, \ldots, J_{n_B-1}^B\}$ in non-decreasing order of r_j^B and let $LS_j = LS_{j+1} - p_j^B$ for all $j = n_B - 1, \ldots, k$.

Continue backwards in this way until we obtain LS_1. Clearly if all the jobs J_j^B start after time LS_j or $r_j^B > LS_j$, then the generated sequence is feasible.

Now combining the idea developed by Agnetis et al. (2004) to construct a polynomial-time algorithm for solving $1| \; |\sum U_j^A : L_{\max}^B$, we obtain the following result.

Theorem 2: Problem $1|r_j^A = 0, r_j^B|\sum U_j^A : L_{\max}^B$ can be solved in $O(n_A \log n_A + n_B \log n_B)$ time.

Proof: The proof is similar to that of Theorem 6.3 in Agnetis et al. (2004).

We now consider another special case where the processing times of all the jobs are equal, i.e., $p_j^X = p$ for all $j = 1, 2, \ldots, n_x$, where $X \in \{A, B\}$, denoted as $1|r_j, p_j^X = p|\sum U_j^A : L_{\max}^B$, and show that this case can be solved in $O((n_A + n_B)^7)$ time.

Theorem 3: Problem $1|r_j, p_j^X = p|\sum U_j^A : L_{\max}^B$ can be solved in $O((n_A + n_B)^7)$ time.

Proof: For each B-job J_j^B, compute its deadline D_j^B, which can viewed as the modified due date of job J_j^B. Assign to each A-job J_j^A a weight $w_j^A = 1$ and to each B-job J_j^B a weight $w_j^B = n_A + 1$. Now apply Baptiste's (1999) algorithm to solve problem $1|r_j, p_j = p|\sum_{J_j^X \in J^X} w_j^X U_j^X$ for the whole job set $J^A \cup J^B$. Since the weights of the B-jobs are so large, they are all on time in the schedule constructed by the algorithm. The running time of Baptiste's (1999) algorithm is $O(n^7)$, where n is the number of jobs. Thus, problem $1|r_j, p_j = p|\sum_{J_j^X \in J^X} w_j^X U_j^X$ can be solved in $O((n_A + n_B)^7)$ time, as required.

5 Branch-and-bound algorithm

While no efficient algorithm is likely to exist to solve an NP-hard scheduling problem in theory, it is still necessary to solve such a problem or find near-optimal solutions in a fast and effective manner in practice (see Flavia Monaco and Sammarra, 2011). In this section, we provide a branch-and-bound and a SA algorithm to search for the optimal and approximately solutions for problem $1|r_j|\sum U_j^A : L_{\max}^B$. In order to speed up the search process in the branch-and-bound algorithm, we derive some dominance properties of the optimal solution in the following.

5.1 Dominance properties

Assume that schedule S has two adjacent jobs J_i^X and J_j^Y with J_i^X immediately preceding J_j^Y, where $X, Y \in \{A, B\}$. Create from S a new schedule S' by swapping the jobs J_i^X and J_j^Y and leaving the other jobs unchanged in schedule *S*. In addition, assume that the starting time to process J_i^X in *S* is *t*. We have the following results.

Lemma 1: If $J_i^X, J_j^Y \in J^A$, $\max\{r_j^A, t\} \geq \max\{r_i^A, t\}$, $d_j^A \geq \max\{\max\{r_i^A, t\} + p_i^A, r_j^A\} + p_j^A$ and $\max\{\max\{r_j^A, t\} + p_j^A, r_i^A\} + p_i^A > d_i^A \geq \max\{r_i^A, t\} + p_i^A$, then *S* dominates *S*′.

Proof: The completion times of the jobs J_i^A and J_j^A in *S* and *S*′ are, respectively,

$$C_i^A(S) = \max\left\{r_i^A, t\right\} + p_i^A$$
$$C_j^A(S) = \max\left\{\max\left\{r_i^A, t\right\} + p_i^A, r_j^A\right\} + p_j^A$$
$$C_j^A(S') = \max\left\{r_j^A, t\right\} + p_j^A$$

and

$$C_i^A(S') = \max\left\{\max\left\{r_j^A, t\right\} + p_j^A, r_i^A\right\} + p_i^A.$$

It is easy to see that $C_i^A(S') > C_i^A(S)$ and $C_j^A(S) \geq C_j^A(S')$. Moreover, it follows from

$$d_j^A \geq \max\left\{\max\left\{r_i^A, t\right\} + p_i^A, r_j^A\right\} + p_j^A$$

and

$$\max\left\{\max\left\{r_j^A, t\right\} + p_j^A, r_i^A\right\} + p_i^A > d_i^A \geq \max\left\{r_i^A, t\right\} + p_i^A$$

that $U_j^A(S') + U_i^A(S') = 1 > 0 = U_i^A(S) + U_j^A(S)$, and from $\max\{r_j^A, t\} \geq \max\{r_i^A, t\}$ that $C_j^A(S) \leq C_i^A(S')$. The result follows.

Lemma 2: If $J_i^X, J_j^Y \in J^A$, $\max\{r_j^A, t\} \geq \max\{r_i^A, t\}$, $\max\{r_j^A, t\} + p_j^A > d_j^A$ and $\max\{\max\{r_j^A, t\} + p_j^A, r_i^A\}$ then *S* dominates *S*′.

Proof: Analogous to the proof of Lemma 1, if the given conditions hold, we have $U_j^A(S') + U_i^A(S') = 2 > 1 = U_i^A(S) + U_j^A(S)$ and $C_j^A(S) \leq C_i^A(S')$, as required.

We can easily establish the following results in the same way as we proved Lemmas 1 and 2.

Lemma 3: If $J_i^X, J_j^Y \in J^A$, $r_j^A \geq \max\{r_i^A, t\} + p_i^A$, and $d_i^A \geq \max\{r_i^A, t\} + p_i^A$, then *S* dominates *S*′.

Lemma 4: If $J_i^X, J_j^Y \in J^B$, $\max\{r_i^B, t\} + p_i^B - d_i^B \leq Q < \max\{\max\{r_j^B, t\} + p_j^B, r_i^B\} + p_i^B - d_i^B$, and $\max\{\max\{r_i^B, t\} + p_i^B, r_j^B\} + p_j^B - d_j^B \leq Q$, then *S* dominates *S*′.

Lemma 5: If $J_i^X \in J^A, J_j^Y \in J^B$, $\max\{r_j^B, t\} \geq \max\{r_i^A, t\}$, and $\max\{\max\{r_i^A, t\} + p_i^A, r_j^B\} + p_j^B - d_j^B \leq Q$, then *S* dominates *S*′.

Next, we present two lemmas to determine the feasibility of a partial sequence. Let (π, -, -) be a sequence of the jobs, where π is the scheduled part with k jobs. Moreover, let *US* be the unscheduled job set and $C_{[*]}$ the completion time of the last job in π.

Lemma 6: If there is a B-job J_j^B in *US* such that $\max\{C_{[k]}, r_j^B\} + p_j^B - d_j^B > Q$, then any sequence ($\pi$, π') is not a feasible solution, where π' is a sequence of the job set *US*.

Lemma 7: If all the unscheduled jobs belong to J^A and there exists an A-job J^A such that $\max\{C_{[k]}, r_j^A\} + p_j^A \le r_k^A$ for all the jobs $J_k^A \in US \setminus \{J_j^A\}$, then job J_j^A may be assigned to (k + 1)th position.

Now let $A(t)$ be the set of available unscheduled jobs at time t, i.e., $A(t) = \{J_j^X \in US \mid t \ge r_j^X\}$, and $B(t)$ the set of unavailable and unscheduled jobs at time t, i.e., $B(t) = \{J_j^X \in US \mid t < r_j^X\}$.

Lemma 8: If $\sum_{J_j^X \in A(t)} p_j^X + t \le \min_{J_k^Y \in B(t)} \{r_k^Y\}$ or $B(t) = \varnothing$, and $A(t) \subseteq J^A$, then there is an optimal schedule such that the early A-jobs in $A(t)$ are scheduled in the earliest due date (EDD) order, and the tardy A-jobs in $A(t)$ are scheduled in any order after the completion of all the processed jobs.

Proof: The first part can be proved by the insertion argument and the second part can be easily observed.

5.2 *A lower bound for* $1|r_j|\sum U_j^A : L_{\max}^B$

The efficiency of a branch-and-bound algorithm largely depends on the effectiveness of lower bounds for curtailing the partial sequences. In this subsection, we propose a lower bound. Let *AS* be a partial sequence in which the order of the first k jobs is determined and *US* be the unscheduled part with n_1 A-jobs and n_2 B-jobs, where $n_1 + n_2 = n_A + n_B - k$. We develop a optimal solution for $1|r_j^A = 0, r_j^B|\sum U_j^A : L_{\max}^B$, which is evidently a lower bound for $1|r_j|\sum U_j^A : L_{\max}^B$ in the following.

Algorithm 1:

Step 1 Compute the latest start time LS_j^B of job J_j^B in *US* and enumerate the A-jobs in *US* such that $d_{(1)}^A \le d_{(2)}^A \le \cdots \le d_{(n_1)}^A$.

Step 2 Foe each B-job J_j^B in *US*, let it start processing at time LS_j^B.

Step 3 Define block i as the ith set of contiguously processed B-jobs. Let there be $n_\beta \le n_2$ blocks and let the starting time and finishing time of each block i be $s(i)$ and $f(i)$, respectively.

Step 4 For each job J_j^A in *US*, if d_j^A falls outside any reserved interval, subtract from d_j^A the total length of all the blocks preceding d_i^A, i.e.,

$\Delta_j^A = d_j^A - \sum_{f(i) \le d_j^A} (f(i) - s(i)).$ If d_j^A falls within block *i*, do the same, but instead of d_j^A use the left extreme of block *k*, i.e., let

$\Delta_j^A = s(k) - \sum_{f(i) \le d_j^A} (f(i) - s(i))$

Step 5 Solve problem $1|\Delta_j^A|\sum U_j^A$ using Moore's algorithm as follows:

- Step 5.1 Order the *A*-jobs in *US* in non-decreasing order of Δ_j^A (i.e., in the EDD order).
- Step 5.2 If no job in the sequence is late, stop. The schedule is optimal.
- Step 5.3 Find the first late job in the schedule and denote it by J_u^A.
- Step 5.4 Find a job J_v^A with $p_v^A = \max_{1 \le i \le u} p_i^A$. Remove job J_v^A from the schedule and process it after the completion of all the processed jobs. Go to Step 5.2.

Let *m* be the optimal solution value of problem $1|\Delta_j^A|\sum U_j^A$. Then a lower bound for the partial sequence *PS* is $LB = \sum_{i=1}^{k} U_{[j]}^X I_X + m,$ where $X \in \{A, B\}$ and $I_X = 1$ if $X = A$, and 0 otherwise.

5.3 Branch-and-bound algorithm

We adopt the depth-first search and assign jobs in a forward manner starting with the first position. In the searching tree, we choose a branch and systematically work down the tree until we either eliminate it by virtue of the dominance properties or the lower bound, or we reach its final node, in which case the resulting sequence either replaces the initial incumbent solution or is eliminated. The branch-and-bound algorithm runs as follows:

Step 1 *Initialisation:* Use a SA heuristic (to be discussed below) to obtain an initial incumbent solution.

Step 2 *Branching:* Apply the depth-first search in the branching procedure until all the nodes are explored or eliminated.

Step 3 *Eliminating:* Apply Lemmas 1–6, to eliminate the dominated partial sequences. Use Lemma 7 to determine the job in the (*k* + 1)th position. For the non-dominated nodes, use Lemma 8 to determine the order of the unscheduled jobs.

Step 4 *Bounding:* Calculate a lower bound on the number of tardy jobs for each unfathomed partial sequence or the number of tardy jobs of the completed sequence for agent *A*. If the lower bound for an unfathomed partial sequence is larger than the initial incumbent solution, eliminate that node and all the nodes beyond it in the branch. If the value of the completed sequence is less than the incumbent solution, adopt the completed sequence as the new incumbent solution. Otherwise, eliminate it.

Step 5 Stopping rule: Repeat Steps 2 to 4 until no more nodes to explore.

5.4 SA algorithm

SA is a well-known meta-heuristic method widely applied to solve combinatorial optimisation problems (Kirkpatrick et al., 1983; Ekren and Ekren, 2010; Lin et al., 2011; Song et al., 2012; Stahlbock and Voβ, 2010; Li and Pang, 2011). Adopting hill climbing moves governed by a control parameter, SA has the advantage of avoiding getting trapped in a local optimum.

We present an SA algorithm to treat problem $1|r_j|\sum U_j^A : L_{\max}^B$ as follows: the SA algorithm commences with four initial solutions. In order to guarantee that the initial solution is feasible, we first arrange the jobs of agent B in the EDD order, followed by arranging the jobs of agent A in four ways, giving rise to four simple variants of the SA: SA_1, in which the A-jobs are in the smallest processing time (SPT) order; SA_2, in which the A-jobs are in the smallest ready time (SRT) order; SA_3, in which the A-jobs are in the EDD order; and SA_4, in which the A-jobs are in the weighted smallest processing time (WSPT) order. Moreover, in order to improve solution quality, we define a compound SA variant as $SA_5 = \min\{SA_1, SA_2, SA_3, SA_4\}$. For neighbourhood generation, we employ the pairwise interchange (PI) generation method to generate neighbourhood solutions. For the acceptance probability, we adopt the following acceptance probability

$$P(accept) = \exp(-\delta \times \Delta U_T),$$

where δ is a control parameter, which changes in the kth iteration according to the method proposed by Ben-Arieh and Maimon (1992) as follows:

$$\delta = \frac{k}{\beta},$$

where β is an experimental constant and ΔU_T is change in the objective value. After preliminary trials, we set $\beta = 2$. As for the stopping rule, all the SA variants are stopped after $100n$ iterations, where n is the number of jobs because, based on preliminary trials, the schedule is quite stable at this stage.

6 Computational results

We conducted extensive computational simulation tests to assess the performance of the branch-and-bound and SA algorithms. We coded all the algorithms in Fortran and ran them on a PC with a Intel(R) Core(TM)2 Quad CPU at 2.66 GHz and 4 GB RAM under XP Windows. Following Reeves (1995), we generated the processing times from a uniform distribution over the integers between 1 and 100, and the release times from a uniform distribution over the integers $(0, 20n\lambda)$, where n is the number of jobs and λ is a control variable. We generated five different sets of problem instances by giving λ the values $1/n$, 0.25, 0.5, 0.75, and 1.0. Moreover, following Fisher (1971), we generated the job due dates from a uniform distribution over the range of integers $T(1 - \tau - R / 2)$ to

$T(1 - \tau + R / 2)$, where τ is the tardiness factor, R is the due date range, and T is the sum of the job processing times, i.e., $T = \sum_{i=1}^{n} p_i$. The combination (τ, R) took the values (0.25, 0.25), (0.25, 0.5), (0.25, 0.75), (0.5, 0.25), (0.5, 0.5), and (0.5, 0.75). We fixed the proportion of A-jobs at pro = 0.5 in the tests.

For the branch-and-bound algorithm, we recorded the average and the maximum numbers of nodes and CPU times (in seconds). For the five SA variants, we define IR_1, IR_2, and IR_3 as follows:

$$IR_1 = \text{the number of } \{U_T(SA_i) - U_T^*(BB) \le 1\},$$

$$IR_2 = \text{the number of } \{1 < U_T(SA_i) - U_T^*(BB) \le 3\},$$

and

$$IR_3 = \text{the number of } \{U_T(SA_i) - U_T^*(BB) \ge 4\}, \quad i = 1, 2, \ldots, 5$$

where $U_T(SA_i)$ is the number of tardy jobs obtained by SA_i and $U_T^*(BB)$ is the number of tardy jobs of the optimal schedule produced by the branch-and-bound algorithm. We did not record the computational times of the SA variants because they are all fast in solving the problems within a second.

We divided the experiments into two parts. In the first part, we fixed the number of jobs n = 14 and 18. In total, we tested 30 experimental cases in the first part and randomly generated 100 replications for each case. So we tested a total of 3,000 problem instances. For the branch-and-bound algorithm, we terminated its execution and proceeded to run the next set of data if the number of nodes exceeded 10^8. We recorded the instances with number of nodes fewer than 10^8 as solvable instances (SI). Tables 1 and 2 summarise the performance of the branch-and-bound and SA algorithms.

For the performance of the branch-and-bound algorithm, Tables 1 and 2 show that the number of nodes generated by it for instances with smaller values of λ are larger than those for instances with larger values of λ when other parameters are fixed. This trend becomes more significant when λ approaches 1. When λ and τ are fixed, the instances with a bigger value of R are easier to solve than those with a smaller value of R. On the other hand, when λ and R are fixed, the instances with a smaller value of τ are more difficult to solve than those with a large value of τ.

As for the performance of the four simple SA variants, it can be seen that the ratios of IR_1, IR_2, and IR_3 to the total number of solvable instances are 61%, 34%, and 5% for SA_1; 34%, 48%, and 18% for SA_2; 33%, 45%, and 22% for SA_3; and 33%, 45%, and 22% for SA_4, respectively. These results indicate that SA1 performs better than its counterparts. Moreover, as shown in Table 2, 434 out of the 3,000 evaluations of solvable instances, the performance of SA_1 is also slightly better than that of the other three simple SA variants. The performance of all the SA variants is not affected by λ, τ, or R. Since all the SA algorithms are fast in solving the problem within a second and the objective function belongs to the nominal scale, which easily causes a bigger gap between an optimal solution and a near-optimal solution, we take $SA_5 = \min\{SA_i, i = 1, 2,\ldots,4\}$ as a compound SA variant. Tables 1 and 2 show that the ratio of the sum of IR_1 and IR_2 to the total number of solvable instances for SA_5 increases up to 96% and 75%, respectively.

Table 1 Performance of the branch-and-bound and SA algorithms with $n = 14$

λ	τ	R	*Branch and bound algorithm*					*SA*$_1$			*SA*$_2$			*SA*$_3$			*SA*$_4$			*SA*$_5$		
			Nodes		*Cpu time*		*IS*															
			Mean	*Max*	*Mean*	*Max*		IR_1	IR_2	IR_3	IR_1	IR_2	IR_3	IR_1	IR_2	IR_3	IR_1	IR_2	IR_3	IR_1	IR_2	IR_3
1/*n*	0.25	0.25	15,746,602	45,874,719	108.91	306.27	100	100	0	0	78	22	0	78	22	0	78	22	0	100	0	0
		0.5	13,292,110	57,725,500	93.57	389.33	100	76	24	0	54	30	16	54	30	16	54	30	16	83	17	0
		0.75	1,336,809	16,859,039	9.99	122.59	100	39	56	5	47	27	26	47	27	26	47	27	26	56	39	5
	0.5	0.25	4,988,140	15,717,823	34.11	98.52	100	96	4	0	48	52	0	48	52	0	48	52	0	96	4	0
		0.5	2,262,214	17,904,084	17.44	131.20	100	65	35	0	7	87	6	7	87	6	7	87	6	65	35	0
		0.75	1,482,723	13,938,365	11.74	112.86	100	35	63	2	4	47	49	4	47	49	4	47	49	36	62	2
0.25	0.25	0.25	17,074,764	52,384,935	117.84	311.69	100	98	2	0	83	17	0	83	17	0	83	17	0	100	0	0
		0.5	12,252,925	52,890,251	85.42	288.36	100	76	24	0	51	34	15	51	34	15	51	34	15	80	20	0
		0.75	1,516,406	16,811,131	11.27	113.56	100	52	43	5	45	37	18	45	37	18	45	37	18	61	34	5
	0.5	0.25	5,564,838	24,551,845	36.66	120.34	100	96	4	0	58	42	0	58	42	0	58	42	0	96	4	0
		0.5	2,152,230	13,036,759	16.01	83.75	100	69	31	0	8	83	9	8	83	9	8	83	9	69	31	0
		0.75	1,174,618	10,239,591	9.50	83.81	100	41	56	3	3	56	41	3	56	41	3	56	41	41	56	3
0.5	0.25	0.25	18,001,016	41,303,007	121.96	266.98	100	100	0	0	85	15	0	85	15	0	85	15	0	100	0	0
		0.5	13,083,452	41,181,168	92.22	277.50	100	76	24	0	50	40	10	50	40	10	50	40	10	80	20	0
		0.75	915,674	12,003,737	6.74	81.61	100	53	43	4	47	29	24	47	29	24	47	29	24	65	31	4

Table 1 Performance of the branch-and-bound and SA algorithms with n = 14 (continued)

λ	τ	R	*Branch and bound algorithm*					SA_1			SA_2			SA_3			SA_4			SA_5		
			Nodes		*Cpu time*		*IS*															
			Mean	*Max*	*Mean*	*Max*		IR_1	IR_2	IR_3	IR_1	IR_2	IR_3	IR_1	IR_2	IR_3	IR_1	IR_2	IR_3	IR_1	IR_2	IR_3
0.5	0.5	0.25	5,384,730	21,588,643	36.09	119.23	100	95	5	0	44	56	0	44	56	0	44	56	0	95	5	0
		0.5	1,963,960	14,607,111	15.08	83.22	100	60	40	0	8	85	7	8	85	7	8	85	7	60	40	0
		0.75	1,260,472	16,311,963	10.12	113.98	100	35	61	4	3	41	56	3	41	56	3	41	56	35	61	4
0.75	0.25	0.25	17,207,349	39,576,063	118.66	264.59	100	100	0	0	77	23	0	77	23	0	77	23	0	100	0	0
		0.5	10,720,454	37,225,546	76.31	248.28	100	73	26	1	56	27	17	56	27	17	56	27	17	77	22	1
		0.75	2,254,792	30,152,471	16.27	206.20	100	41	55	4	37	37	26	37	37	26	37	37	26	53	43	4
	0.5	0.25	5,271,560	21,357,799	36.34	116.75	100	98	2	0	42	58	0	42	58	0	42	58	0	98	2	0
		0.5	1,926,284	10,386,961	15.31	76.39	100	59	40	1	4	89	7	4	89	7	4	89	7	59	40	1
		0.75	1,129,341	10,944,781	9.22	85.58	100	38	58	4	6	45	49	6	45	49	6	45	49	38	58	4
1	0.25	0.25	6,081,433	47,348,216	54.40	386.11	100	38	52	10	11	63	26	6	31	63	6	31	63	39	51	10
		0.5	3,976,949	36,023,947	34.47	279.16	100	11	52	37	5	42	53	5	8	87	5	8	87	13	56	31
		0.75	1,075,974	7,530,739	9.36	72.45	100	6	54	40	4	42	54	2	11	87	2	11	87	10	57	33
	0.5	0.25	1,206,936	5,813,241	8.69	49.81	100	53	46	1	34	65	1	31	68	1	31	68	1	53	46	1
		0.5	526,316	5,401,508	4.25	51.14	100	25	70	5	13	76	11	6	79	15	6	79	15	26	69	5
		0.75	534,144	3,632,078	4.31	29.89	100	22	66	12	10	64	26	3	59	38	3	59	38	22	66	12

Table 2 Performance of the branch-and-bound and SA algorithms with $n = 18$

λ	τ	R	*Branch and bound algorithm*					*SA_1*			*SA_2*			*SA_3*			*SA_4*			*SA_5*		
			Nodes		*Cpu time*		*IS*															
			Mean	*Max*	*Mean*	*Max*		*IR_1*	*IR_2*	*IR_3*	*IR_1*	*IR_2*	*IR_3*	*IR_1*	*IR_2*	*IR_3*	*IR_1*	*IR_2*	*IR_3*	*IR_1*	*IR_2*	*IR_3*
$1/n$	0.25	0.25	7,039,601	7,039,601	102.98	102.98	1	1	0	0	0	0	1	0	0	1	0	0	1	1	0	0
		0.5	3	3	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0
		0.75	20,667,630	94,976,990	309.34	1,579.55	39	17	15	7	19	11	9	19	11	9	19	11	9	24	10	5
	0.5	0.25	14,982,561	14,982,561	197.45	197.45	1	1	0	0	0	1	0	0	1	0	0	1	0	1	0	0
		0.5	38,846,860	92,673,794	619.21	1,508.11	12	4	8	0	0	4	8	0	4	8	0	4	8	4	8	0
		0.75	46,428,101	97,826,472	704.34	1,592.36	18	1	10	7	0	1	17	0	1	17	0	1	17	1	10	7
0.25	0.25	0.25					0															
		0.5					0															
		0.75	15,418,026	95,455,783	204.45	1,386.19	31	12	19	0	23	4	4	23	4	4	23	4	4	23	8	0
	0.5	0.25					0															
		0.5	43,370,622	90,565,890	684.08	1,389.31	15	3	8	4	0	3	12	0	3	12	0	3	12	3	8	4
		0.75	34,211,723	96,213,192	515.82	1,711.31	30	6	18	6	0	4	26	0	4	26	0	4	26	6	18	6
0.5	0.25	0.25					0															
		0.5					0															
		0.75	6,037,137	59,416,677	90.48	871.38	36	13	18	5	22	8	6	22	8	6	22	8	6	24	10	2

Table 2 Performance of the branch-and-bound and SA algorithms with n = 18 (continued)

λ	τ	R	*Branch and bound algorithm*					SA_1			SA_2			SA_3			SA_4			SA_5		
			Nodes		*Cpu time*		*IS*															
			Mean	*Max*	*Mean*	*Max*		IR_1	IR_2	IR_3	IR_1	IR_2	IR_3	IR_1	IR_2	IR_3	IR_1	IR_2	IR_3	IR_1	IR_2	IR_3
0.5	0.5	0.25					0															
		0.5	54,366,557	93,378,119	785.91	1,356.44	10	5	5	0	0	9	1	0	9	1	0	9	1	5	5	0
		0.75	36,380,945	77,495,137	553.77	1,175.06	30	3	23	4	0	2	28	0	2	28	0	2	28	3	23	4
0.75	0.25	0.25	43,256,243	80,372,165	627.63	1,148.56	2	2	0	0	0	0	2	0	0	2	0	0	2	2	0	0
		0.5					0															
		0.75	19,547,742	88,548,454	290.13	1,327.63	32	9	22	1	23	7	2	23	7	2	23	7	2	24	7	1
	0.5	0.25					0															
		0.5	54,049,721	99,459,069	820.19	1,573.25	16	4	12	0	2	2	12	2	2	12	2	2	12	4	12	0
		0.75	31,298,389	95,935,981	472.82	1,481.81	32	6	16	10	0	6	26	0	6	26	0	6	26	6	16	10
1	0.25	0.25					0															
		0.5	79,258,768	97,787,518	1,345.41	1,587.38	4	1	1	2	0	1	3	0	0	4	0	0	4	1	1	2
		0.75	40,557,468	97,502,795	641.86	1,565.88	31	0	7	24	0	2	29	0	2	29	0	2	29	0	8	23
	0.5	0.25	35,423,404	93,319,381	526.00	1,426.50	8	2	6	0	2	4	2	2	4	2	2	4	2	2	6	0
		0.5	40,293,282	94,357,546	610.07	1,415.38	36	0	20	16	1	12	23	0	11	25	0	11	25	1	19	16
		0.75	43,912,141	93,934,985	665.57	1,627.94	49	2	17	30	1	8	40	1	2	46	1	2	46	2	18	29

In the second part, we tested the proposed SA variants with the number of jobs fixed at n = 50 and 100 to further assess their performance in handling large job-sized problem instances. As a result, we examined 30 experimental cases. For each case, we randomly generated 100 replications. So we tested a total of 3,000 randomly generated problem instances. For the four simple SA variants, we define IR_1, IR_2, and IR_3 as follows:

$$IR_1 = \text{the number of } \{U_T(SA_i) - U_T^{**}(SA) \le 1\},$$

$$IR_2 = \text{the number of } \{1 < U_T(SA_i) - U_T^{**}(SA) \le 3\}$$

and

$$IR_3 = \text{the number of } \{U_T(SA_i) - U_T^{**}(SA) \ge 4\}, \quad i = 1, 2, \dots, 4,$$

where $U_T(SA_i)$ is the objective value generated by SA_i and $U_T^{**}(SA) = \min\{U_T(SA_i),$ i = 1, 2,…,4} is the smallest objective value obtained among SA_1, SA_2, SA_3, and SA_4. Tables 3 and 4 report the results.

As shown in Table 3, the ratio of IR_1 to the total number of solvable cases for SA_1 is 90% or higher, whereas those of SA_2, SA_3, and SA_4 at 30%, 26%, and 26%, respectively. These results show that SA_1 outperforms its counterparts. Table 4 shows that the SA variants have similar performance. Moreover, the performance of the all proposed SA variants are not affected by λ, τ, or R. In addition, there is no dominance relationship among them. Thus, we recommend that the compound SA_5 be used since it has both accuracy and stability in solving the problem.

Table 3 Performance of the simple SA variants with n = 50

λ	τ	R	SA_1			SA_2			SA_3			SA_4		
			IR_1	IR_2	IR_3	IR_1	IR_2	IR_3	IR_1	IR_2	IR_3	IR_1	IR_2	IR_3
1/n	0.25	0.25	98	2	0	22	25	53	22	25	53	22	25	53
		0.5	57	20	23	56	10	34	56	10	34	56	10	34
		0.75	72	12	16	53	10	37	53	10	37	53	10	37
	0.5	0.25	100	0	0	8	29	63	8	29	63	8	29	63
		0.5	100	0	0	0	11	89	0	11	89	0	11	89
		0.75	100	0	0	0	6	94	0	6	94	0	6	94
0.25	0.25	0.25	98	2	0	30	33	37	30	33	37	30	33	37
		0.5	61	17	22	52	9	39	52	9	39	52	9	39
		0.75	68	20	12	58	13	29	58	13	29	58	13	29
	0.5	0.25	100	0	0	7	27	66	7	27	66	7	27	66
		0.5	100	0	0	3	13	84	3	13	84	3	13	84
		0.75	100	0	0	0	3	97	0	3	97	0	3	97
0.5	0.25	0.25	93	7	0	35	27	38	35	27	38	35	27	38
		0.5	54	12	34	60	5	35	60	5	35	60	5	35
		0.75	76	12	12	47	25	28	47	25	28	47	25	28
	0.5	0.25	100	0	0	7	27	66	7	27	66	7	27	66
		0.5	100	0	0	1	14	85	1	14	85	1	14	85
		0.75	100	0	0	0	2	98	0	2	98	0	2	98

Table 3 Performance of the simple SA variants with n = 50 (continued)

λ	τ	R	SA_1			SA_2			SA_3			SA_4		
			IR_1	IR_2	IR_3	IR_1	IR_2	IR_3	IR_1	IR_2	IR_3	IR_1	IR_2	IR_3
0.75	0.25	0.25	97	2	1	29	32	39	29	32	39	29	32	39
		0.5	59	16	25	56	11	33	56	11	33	56	11	33
		0.75	64	17	19	54	11	35	54	11	35	54	11	35
	0.5	0.25	100	0	0	11	22	67	11	22	67	11	22	67
		0.5	100	0	0	0	12	88	0	12	88	0	12	88
		0.75	100	0	0	0	7	93	0	7	93	0	7	93
1	0.25	0.25	100	0	0	30	38	32	12	33	55	12	33	55
		0.5	100	0	0	33	42	25	7	19	74	7	19	74
		0.75	99	1	0	26	47	27	0	7	93	0	7	93
	0.5	0.25	99	1	0	83	15	2	79	18	3	79	18	3
		0.5	100	0	0	76	23	1	64	32	4	64	32	4
		0.75	99	1	0	63	33	4	29	48	23	29	48	23

Table 4 Performance of the simple SA variants with n = 100

λ	τ	R	SA_1			SA_2			SA_3			SA_4		
			IR_1	IR_2	IR_3	IR_1	IR_2	IR_3	IR_1	IR_2	IR_3	IR_1	IR_2	IR_3
$1/n$	0.25	0.25	99	0	1	12	9	79	12	9	79	12	9	79
		0.5	64	3	33	46	3	51	46	3	51	46	3	51
		0.75	75	11	14	43	11	46	43	11	46	43	11	46
	0.5	0.25	100	0	0	0	4	96	0	4	96	0	4	96
		0.5	100	0	0	0	0	100	0	0	100	0	0	100
		0.75	100	0	0	0	0	100	0	0	100	0	0	100
0.25	0.25	0.25	98	2	0	5	12	83	5	12	83	5	12	83
		0.5	66	4	30	42	7	51	42	7	51	42	7	51
		0.75	80	7	13	37	8	55	37	8	55	37	8	55
	0.5	0.25	100	0	0	0	5	95	0	5	95	0	5	95
		0.5	100	0	0	0	1	99	0	1	99	0	1	99
		0.75	100	0	0	0	0	100	0	0	100	0	0	100
0.5	0.25	0.25	100	0	0	3	7	90	3	7	90	3	7	90
		0.5	58	6	36	51	1	48	51	1	48	51	1	48
		0.75	74	6	20	37	8	55	37	8	55	37	8	55
	0.5	0.25	100	0	0	0	7	93	0	7	93	0	7	93
		0.5	100	0	0	0	0	100	0	0	100	0	0	100
		0.75	100	0	0	0	0	100	0	0	100	0	0	100

Table 4 Performance of the simple SA variants with n = 100 (continued)

λ	τ	R	SA_1			SA_2			SA_3			SA_4		
			IR_1	IR_2	IR_3	IR_1	IR_2	IR_3	IR_1	IR_2	IR_3	IR_1	IR_2	IR_3
0.75	0.25	0.25	99	1	0	10	15	75	10	15	75	10	15	75
		0.5	56	3	41	50	0	50	50	0	50	50	0	50
		0.75	72	6	22	41	14	45	41	14	45	41	14	45
	0.5	0.25	100	0	0	3	3	94	3	3	94	3	3	94
		0.5	100	0	0	0	0	100	0	0	100	0	0	100
		0.75	100	0	0	0	0	100	0	0	100	0	0	100
1	0.25	0.25	100	0	0	22	23	55	10	20	70	10	20	70
		0.5	100	0	0	14	26	60	1	4	95	1	4	95
		0.75	99	1	0	18	26	56	0	0	100	0	0	100
	0.5	0.25	100	0	0	94	6	0	92	8	0	92	8	0
		0.5	99	1	0	77	22	1	66	25	9	66	25	9
		0.75	94	6	0	43	30	27	5	29	66	5	29	66

7 Conclusions

In this paper, we consider a two-agent single-machine scheduling problem with different job release times. The objective is to find an optimal schedule that minimises the number of the tardy jobs of one agent with the restriction that the maximum lateness of the jobs of the other agent cannot exceed a given value. We first establish that the problem is strongly NP-hard and then show that two special cases of the problem are solvable in polynomial time. Following that, we present some dominance properties and a lower bound on the optimal solution, and exploit them to develop a branch-and-bound algorithm to solve the problem. In addition we provide five variants of a SA algorithm to obtain approximate solutions for the problem. The computational results show that with the help of the proposed initial SA-derived solutions, the branch-and-bound algorithm can solve instances with up to 18 jobs. Moreover, the results also show that the compound variant of the SA performs well in terms of both accuracy and stability.

Acknowledgements

The authors would like to thank to editor and the referees for their interest in our work and their valuable comments for improving the paper. This paper was supported in part by the Natural Science Foundation for Young Scholars of Jiangxi, China (2010GQS0003); in part by the Science Foundation of Education Committee for Young Scholars of Jiangxi, China (GJJ11143); in part by the NSC under grant number NSC 99-2221-E-035-057-MY3.

References

Agnetis, A., Mirchandani, P.B., Pacciarelli, D. and Pacifici, A. (2004) 'Scheduling problems with two competing agents', *Operations Research*, Vol. 52, No. 2, pp.229–242.

Agnetis, A., Pacciarelli, D. and Pacifici, A. (2007) 'Multi-agent single machine scheduling', *Annals of Operations Research*, Vol. 150, No. 1, pp.3–15.

Agnetis, A., Pascale, G. and Pacciarelli, D. (2009) 'A Lagrangian approach to single-machine scheduling problems with two competing agents', *Journal of Scheduling*, Vol. 12, No. 4, pp.401–415.

Baker, K.R. and Smith, J.C. (2003) 'A multiple criterion model for machine scheduling', *Journal of Scheduling*, Vol. 6, No. 1, pp.7–16.

Baptiste, P. (1999) 'Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine with equal processing times', *Journal of Scheduling*, Vol. 2, No. 6, pp.245–252.

Ben-Arieh, D. and Maimon, O. (1992) 'Annealing method for PCB assembly scheduling on two sequential machines', *International Journal of Computer Integrated Manufacturing*, Vol. 5, No. 6, pp.361–367.

Cheng, T.C.E., Ng, C.T. and Yuan, J.J. (2006) 'Multi-agent scheduling on a single machine to minimize total weighted number of tardy jobs', *Theoretical Computer Science*, Vol. 362, No. 1, pp.273–281.

Cheng, T.C.E., Ng, C.T. and Yuan, J.J. (2008) 'Multi-agent scheduling on a single machine with max-form criteria', *European Journal of Operational Research*, Vol. 188, No. 2, pp.603–609.

Cheng, T.C.E., Cheng, S.R., Wu, W.H., Hsu, P.H. and Wu, C.C. (2011a) 'A two-agent single-machine scheduling problem with truncated sum-of-processing-times-based learning considerations', *Computers & Industrial Engineering*, Vol. 60, No. 4, pp.534–541.

Cheng, T.C.E., Wu, W.H., Cheng, S.R. and Wu, C.C. (2011b) 'Two-agent scheduling with position-based deteriorating jobs and learning effects', *Applied Mathematics and Computation*, Vol. 217, No. 21, pp.8804–8824.

Ekren, O. and Ekren, B.Y. (2010) 'Size optimization of a PV/wind hybrid energy conversion system with battery storage using simulated annealing', *Applied Energy*, Vol. 87, No. 2, pp.592–598.

Fisher, M.L. (1971) 'A dual algorithm for the one-machine scheduling problem', *Mathematical Programming*, Vol. 11, No. 1, pp.229–251.

Flavia Monaco, M. and Sammarra, M. (2011) 'Quay crane scheduling with time windows, one-way and spatial constraints', *International Journal of Shipping and Transport Logistics*, Vol. 3, No. 4, pp.454–474.

French, S. (1982) *Sequencing and Scheduling: An Introduction to the Mathematics of the Job Shop*, Ellis Horwood Limited, Chichester, West Sussex and New York.

Kirkpatrick, S., Gelatt, C. and Vecchi, M. (1983) 'Optimization by simulated annealing', *Science*, Vol. 220, No. 4598, pp.671–680.

Leung, J.Y.T., Pinedo, M. and Wan, G. (2010) 'Competitive two-agent scheduling and its applications', *Operations Research*, Vol. 58, No. 2, pp.458–469.

Li, C-L. and Pang, K-W. (2011) 'An integrated model for ship routing and berth allocation', *International Journal of Shipping and Transport Logistics*, Vol. 3, No. 3, pp.245–260.

Li, D. and Hsu, P. (2012) 'Solving a two-agent single-machine scheduling problem considering learning effect', *Computers & Operations Research*, Vol. 39, No. 7, pp.1644–1651.

Lin, C.C., Lee, Y.Y. and Ye, H.C. (2011) 'Mental map preserving graph drawing using simulated annealing', *Information Sciences*, Vol. 181, No. 19, pp.4253–4272.

Liu, P., Tang, L.X. and Zhou, X.Y. (2010) 'Two-agent group scheduling with deteriorating jobs on a single machine', *International Journal of Advanced Manufacturing Technology*, Vol. 47, Nos. 5–8, pp.657–664.

Liu, P., Yi, N. and Zhou, X.Y. (2011) 'Two-agent single-machine scheduling problems under increasing linear deterioration', *Applied Mathematical Modelling*, Vol. 35, No. 5, pp.2290–2296.

Lun, Y.H.V., Lai, K.H., Ng, C.T., Wong, C.W.Y. and Cheng, T.C.E. (2011) 'Research in shipping and transport logistics', *International Journal of Shipping and Transport Logistics*, Vol. 3, No. 1, pp.1–5.

Mor, B. and Mosheiov, G. (2010) 'Scheduling problems with two competing agents to minimize minmax and minsum earliness measures', *European Journal of Operational Research*, Vol. 206, No. 3, pp.540–546.

Mor, B. and Mosheiov, G. (2011) 'Single machine batch scheduling with two competing agents to minimize total flowtime', *European Journal of Operational Research*, Vol. 215, No. 3, pp.524–531.

Ng, C.T., Cheng, T.C.E. and Yuan, J.J. (2006) 'A note on the complexity of the two-agent scheduling on a single machine', *Journal of Combinatorial Optimization*, Vol. 12, No. 4, pp.387–394.

Nong, Q.Q., Cheng, T.C.E. and Ng, C.T. (2011) 'Two-agent scheduling to minimize the total cost', *European Journal of Operational Research*, Vol. 215, No. 4, pp.39–44.

Reeves, C. (1995) 'Heuristics for scheduling a single machine subject to unequal job release times', *European Journal of Operational Research*, Vol. 80, No. 2, pp.397–403.

Song, S-Z., Ren, J-J. and Fan, J-X. (2012) 'Improved simulated annealing algorithm used for job shop scheduling problems', *Advances in Intelligent and Soft Computing*, Vol. 139, No. 3, pp.17–25.

Stahlbock, R. and Voβ, S. (2010) 'Efficiency considerations for sequencing and scheduling of double-rail-mounted gantry cranes at maritime container terminals', *International Journal of Shipping and Transport Logistics*, Vol. 2, No. 1, pp.95–123.

Wan, G., Vakati, R.S., Leung, J.Y.T. and Pinedo, M. (2010) 'Scheduling two agents with controllable processing times', *European Journal of Operational Research*, Vol. 205, No. 3, pp.528–539.

Yin, Y., Cheng, X.R. and Wu, C.C. (2012) 'Scheduling problems with two agents and a linear non-increasing deterioration to minimize earliness penalties', *Information Sciences*, Vol. 189, No. 8, pp.282–292.

Yuan, J.J., Shang, W.P. and Feng, Q. (2005) 'A note on the scheduling with two families of jobs', *Journal of Scheduling*, Vol. 8, No. 6, pp.537–542.

Zhang, F., Ng, C.T., Tang, G., Cheng, T.C.E. and Lun, Y.H.V. (2011) 'Inverse scheduling: applications in shipping', *International Journal of Shipping and Transport Logistics*, Vol. 3, No. 3, pp.312–322.