Scheduling Interfering Job Sets on Parallel Machines

Hari Balasubramanian[1], John Fowler[2], Ahmet Keha[2], and Michele Pfund[3]
1: Department of Mechanical and Industrial Engineering, University of Massachusetts, Amherst, MA
2: Department of Industrial Engineering, Arizona State University, Tempe, AZ
3: Department of Supply Chain Management, Arizona State University, Tempe, AZ
hbalasubraman@ecs.umass.edu, john.fowler@asu.edu, ahmet.keha@asu.edu, michele.pfund@asu.edu

Abstract

We consider bicriteria scheduling on identical parallel machines in a nontraditional context: jobs belong to two disjoint sets, and each set has a different criterion to be minimized. The jobs are all available at time zero and have to be scheduled (non-preemptively) on m parallel machines. The goal is to generate the set of all non-dominated solutions, so the decision maker can evaluate the tradeoffs and choose the schedule to be implemented. We consider the case where, for one of the two sets, the criterion to be minimized is makespan while for the other the total completion time needs to be minimized. Given that the problem is NP-hard, we propose an iterative SPT-LPT-SPT heuristic and a bicriteria genetic algorithm for the problem. Both approaches are designed to exploit the problem structure and generate a set of non-dominated solutions. In the genetic algorithm we use a special encoding scheme and also a unique strategy - based on the properties of a non-dominated solution - to ensure that all parts of the non-dominated front are explored. The heuristic and the genetic algorithm are compared with a time-indexed integer programming formulation for small and large instances. Results indicate that the both the heuristic and the genetic algorithm provide high solution quality and are computationally efficient. The heuristics proposed also have the potential to be generalized for the problem of interfering job sets involving other bicriteria pairs.

Keywords: interfering job sets, parallel machines, bicriteria scheduling.

1 Introduction

Traditionally, multicriteria scheduling problems have been considered with the objective of minimizing criteria that apply to each of the jobs being scheduled. While motivation for such problems can frequently be found in practice, it is also possible to have situations in which jobs belong to disjoint classes or sets, with a criterion associated with each set. The job sets in such a situation are said to compete or *interfere* with each other for the same resources. Research in the area of interfering job sets is limited. In Hoogeveen [2005]'s review of multicriteria scheduling problems, he mentions the scheduling of interfering job sets as one of the new developments in the area. The work of Peha [1995] is the earliest reference on the topic. He considers the lexicographic optimization of the weighted number of tardy jobs for one set and the total weighted completion time for the other under the assumption of unit processing times, integer release dates and identical parallel machines. Peha [1995]'s research is motivated by real time systems and integrated-services

networks. He provides polynomial time algorithms for the problem which exploit the assumption of unit processing times.

Agnetis et al. [2003] consider the problem of two users competing for common job shop resources. Their motivation comes from the decision by two major companies to propose a joint venture to construct a flexible manufacturing system for their products. Agnetis et al. [2003] further state that discussion with these companies indicated that a decision support system that enables negotiation between the two competing users of the manufacturing system would be useful. Agnetis et al. [2003] also mention other applications where users with different goals compete with each other for the same resources: 1) scheduling multiple flights of different airlines on a common set of runways, 2) scheduling berths and material/people movers (cranes, walkways, etc.) at a port for multiple ships, 3) scheduling clerical workers among different "bosses" in an office, and 4) scheduling a mechanical/electrical workshop for different users. Baker and Smith [2003] present an example in which a prototype shop is shared by both the Research and Development department and the Manufacturing Engineering department. The Research and Development department might have concerns on meeting due-dates while the Manufacturing Engineering department might have concerns about quick response times.

In a different paper Agnetis et al. [2004] present complexity results for generating non-dominated solutions for single machine and shop scheduling problems given that jobs belong to one of two sets. The objectives they consider include $\sum C_j$ and $\sum U_j$. Cheng et al. [2006a] show the NP-hardness of the high multiplicity encoding version of the problem of minimizing $\sum C_j$ on a single machine with a constraint on $\sum U_j$, while Cheng et al. [2006b] show the strong NP-hardness of problem where jobs belong to one of many sets and $\sum w_j U_j$ is to be minimized for each set. Agnetis et al. [2007] tackle the computational complexity of the single machine problem involving interfering job sets and with more generally defined cost functions. Baker and Smith [2003] also consider the single machine version of the problem and show the polynomial solvability of bicriteria problems involving 1) C_{max} 2) $\sum C_j$ 3) $\sum w_j C_j$ 4) L_{max}, except for the $\sum w_j C_j$, L_{max} pair, which they show to be NP-hard. Their bicriteria optimization function is a linear combination of the criteria with weights on each criterion. We note that the Baker and Smith [2003] approach of minimizing a linear combination is an *a priori* approach: information is available beforehand on the weights of the two criteria.

The problem of scheduling interfering job sets has some unique structural properties in the single machine environment. We consider the case where jobs belong to one of two disjoint sets $C1$ and $C2$. Key results from Baker and Smith [2003] that can be proved easily for regular scheduling measures by contradiction arguments are:

Property 1: If makespan is the criteria for one of the sets, then there is an optimal schedule in which all jobs belonging to the makespan set are processed consecutively.

Property 2: If total completion time is the criteria for one of the sets, then there exists an optimal schedule in which jobs in the total completion time set are processed in shortest processing time (SPT) order.

Intuitively, Property 1 tells us that since only the latest finish time of the set of makespan jobs matters, processing the jobs non-consecutively can never increase the objective value of the jobs in the other set (true for regular criteria). Property 1 has an interesting implication on the optimal schedule of jobs. Since all the jobs for which makespan is to be minimized are to be processed consecutively, they can all be accumulated into a single "makespan job". The processing time of the makespan job is the sum of the processing times of the jobs in the makespan set. Minimizing a

weighted linear combination of the the two objectives then reduces to the total weighted completion time problem (there could be a different weight on the makespan job and equal weights on all the $\sum C_j$ jobs), solvable using the well-known weighted shortest processing time (WSPT) rule.

If the set of all non-dominated solutions were to be generated (not considered by Baker and Smith [2003]), the makespan job would be placed at each position in the schedule (position 1, position 2,...,position n_2+1) preceded and/or followed by the $\sum C_j$ jobs ordered by the SPT rule (due to property 2).

This research extends the single machine research of Baker and Smith [2003] by considering the problem of two interfering jobs sets in the identical parallel machine environment. We limit our study to two well-known classical scheduling criteria: makespan and total completion time. Jobs are divided into two disjoint sets: one for which makespan needs to be minimized, and the other for which total completion time needs to be minimized. The problem is NP-hard as the single criterion problem of minimizing makespan on parallel machines is NP-hard. We propose computationally efficient heuristic techniques that can be extended with modifications to other bicriteria pairs. Our goal is to generate the set of non-dominated solutions, so the decision maker can evaluate the tradeoffs in the criteria. This is the *a posteriori* approach in which the decision maker makes his choice only after a set of points is presented. Since makespan is equivalent to the decision problem involving a common deadline (i.e. whether a feasible schedule can be obtained such that all jobs finish before a common deadline) the set of non-dominated solutions can give the decision-maker important information on whether jobs for one set can be finished by a given time and the resulting compromise or effect on the sum of completion times of the other customer's jobs.

We propose two different heuristic techniques in the paper: an iterative SPT-LPT-SPT (S-L-S) heuristic and a bicriteria genetic algorithm. Both approaches are designed specifically to exploit the problem structure and the properties of a non-dominated solution. Over the last two decades, a number of different approaches have been proposed for using evolutionary algorithms for multicriteria problems. We point to the tutorial by Landa Silva and Burke [2004b] for a concise description of the main developments in the area. We note that the genetic algorithm proposed in this research, due to its problem specific nature, is quite different from the GA approaches proposed in the literature for traditional multicriteria parallel machine scheduling problems. We propose a special encoding scheme in this research and also a strategy to explore all portions of the non-dominated front; our ideas are built to capture the structural implications of interference between jobs sets, especially when one of the criteria is makespan.

The remainder of the paper is organized as follows. We define our problem in Section 2. In Section 3 we discuss key aspects of the problem structure and propose the S-L-S heuristic. In Section 4, we describe the bicriteria genetic algorithm. We then compare, in Sections 5 and 6, the performance of the S-L-S heuristic and the genetic algorithm to the true set of non-dominated solutions generated by an integer program. Finally, the paper is summarized, overall conclusions are given and future research directions are discussed in Section 7.

2 Problem definition

We consider the problem of scheduling n jobs on m identical parallel machines. The jobs belong to one of two disjoint sets $C1$ and $C2$ with n_1 and n_2 jobs in them respectively, such that $n_1 + n_2 = n$. For the jobs in the set $C1$, we are interested in minimizing criterion C_{max} and for the jobs in the set $C2$, we are interested in minimizing criterion $\sum C_j$. Each job has a processing time p_j and we

assume that all jobs are available for processing at time zero.

In the $\alpha|\beta|\gamma$ scheduling notation of Graham et al. [1979], we refer to this problem as $P|inter|ND(C_{max}, \sum C_j)$, where *inter* is our notation to indicate that jobs of the different sets interfere with one another, and $ND(C_{max}, \sum C_j)$ is our notation to indicate we are interested in generating the set of non-dominated points.

The goal is to generate a set of *non-dominated* or *Pareto optimal* solutions, so the decision maker can determine the tradeoffs involved in scheduling the two sets of jobs. Let S be the set of feasible solutions for a bicriteria optimization problem with interfering job sets. Let $z_1(x)$ and $z_2(x)$ be the objective values for criteria z_1 (corresponding to set $C1$) and z_2 (corresponding to set $C2$) for a feasible solution $x \in S$ (both z_1 and z_2 need to be minimized).

Definition 2.1. *A solution x^* is Pareto optimal or non-dominated if there exists no other solution $x \in S$ for which $z_1(x) \leq z_1(x^*)$ and $z_2(x) \leq z_2(x^*)$ where at least one of the inequalities is strict.*

The problem of generating the set of non-dominated solutions for C_{max} and $\sum C_j$, given that the jobs sets corresponding to the two criteria interfere with each other, is strongly NP-hard. Indeed solving any bicriteria problem with interfering job sets involving classical scheduling criteria in the parallel machine environment is strongly NP-hard. This is because only $P||\sum C_j$ is polynomially solvable (for an extensive compilation of complexity results for single criteria scheduling problems see Brucker and Knust [2006]) while all other criteria are NP-hard. Our goal in this research is to develop computationally efficient heuristics that are able to generate solutions that are non-dominated or near non-dominated.

3 Some features of the $P|inter|ND(C_{max}, \sum C_j)$ problem

Some of the properties of the single machine problem can be extended to the parallel machine case. It can easily be shown that in a non-dominated solution, jobs in $C1$, for which makespan is to be minimized and that are assigned to the same machine, are always contiguous (i.e. they are processed consecutively). This is because non-contiguous processing of the jobs in $C1$ can only increase the $\sum C_j$ of jobs in $C2$. Once the makespan jobs have been assigned to the machines, the n_1 jobs of set $C1$ are now reduced to a maximum of m "makespan" jobs (there could be fewer) to be scheduled, where each makespan job is an aggregation of jobs from $C1$ assigned to a given machine. It can also be shown that on a given machine jobs in the set $C2$ for which $\sum C_j$ is to be minimized are processed in SPT order. (However, it is not possible to make a stronger statement that the jobs in $C2$ will appear in SPT order on the m machines; a counterexample that disproves this is given at the end of this Section).

A non-dominated solution, from the above discussion, can be visualized as follows: a "block" or subschedule S_1 consisting of k jobs belonging to $C2$ followed by a subschedule S_2 consisting of up to m makespan jobs ($C1$), followed by a subschedule S_3 consisting of jobs belonging to $C2$ that were not scheduled in S_1. The situation is shown in Figure 1, where the jobs in gray represent the jobs in $C2$ while the unshaded jobs represent the jobs in $C1$. Naturally, if the decision maker is inclined towards a schedule with lower makespan, the subschedule S_2 of makespan jobs will occur early in the schedule (S_1 may be a null set), while if the preference is towards a lower $\sum C_j$, it will occur later in the schedule.

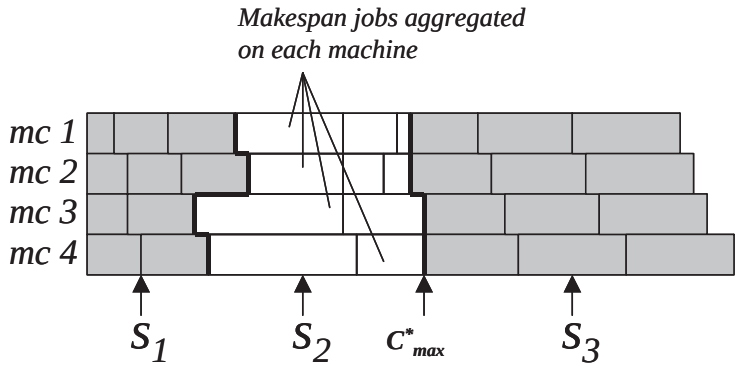

Figure 1: Structure of a non-dominated solution when one of the criteria is makespan

We devise a heuristic to generate a set of near non-dominated points based on the idea that the jobs of the two sets occur in these three alternating blocks. It is well known that the $P||\sum C_j$ problem is optimally solved by scheduling jobs in Shortest Processing Time order (Conway et al. [1967]) and that the Longest Processing Time (LPT) heuristic is an effective heuristic for the $P||C_{max}$ problem Graham [1966]. We use the SPT and LPT heuristics iteratively to generate a set of points using the following algorithm:

Iterative SPT-LPT-SPT (S-L-S)

For every $k = 0$ to $k = n_2$, do:

Step 1. Construct a partial SPT schedule for the k shortest processing time jobs in $C2$. Let $(A_1, A_2, ..., A_m)$ be the finish times on the machines once the k jobs have been scheduled.

Step 2. Given $(A_1, A_2, ..., A_m)$ from step 1, use the modified longest processing time (MLPT) of Lee [1991] to schedule the jobs in $C1$. Update the finish times of the machines to $(A_1^{'}, A_2^{'}, ..., A_m^{'})$.

Step 3. Given $(A_1^{'}, A_2^{'}, ..., A_m^{'})$ from step 2, schedule the remaining $n_2 - k$ jobs in $C2$ in SPT order.

Step 4. Let C_{max}^* be the makespan value of jobs in $C1$ obtained at the end of step 2. For each machine i on which jobs in $C1$ finish before C_{max}^*, determine if there exist jobs in S_3 assigned to machine i (based on the SPT ordering of step 3) that can be scheduled earlier than the makespan jobs on machine i while still not exceeding the determining C_{max}^* value. If such a job or jobs do exist, schedule them before the makespan jobs on machine i.

Remarks

1. Note that when $k = 0$ the algorithm goes directly to step 2 and schedules all the jobs in $C1$ using an LPT-based heuristic and in step 3 it schedules all the jobs in $C2$ in SPT order. The resulting schedule will be close to optimal for C_{max} while $\sum C_j$ becomes a secondary criterion. On the other hand, when $k = n_2$ only steps 1 and 2 are executed: in step 1, the optimal schedule (SPT schedule) for the jobs in $C2$ is obtained followed by an LPT-based ordering of jobs in $C1$. Here, $\sum C_j$ can be viewed as a primary criterion while C_{max} is the secondary criterion.

2. When $1 \leq k \leq n_2 - 1$, the algorithm progressively shifts the jobs in $C1$ from the beginning of the schedule to the end. This process seeks to generate the set of non-dominated points.

3. When $1 \leq k \leq n_2$, the k jobs from $C1$ scheduled prior to the makespan jobs may have caused the machines to finish at different times. Thus the makespan for jobs in $C1$ now needs to be minimized when machines are not all available at the same time. This problem is a generalization of the classic makespan scheduling problem on parallel machines. We use the Modified Longest Processing Time (MLPT) algorithm of Lee [1991] to obtain a schedule of jobs in $C1$. The heuristic assumes that in addition to the jobs in $C1$ m additional jobs, each of length equal to the finish times of the machines, $(A_1, A_2, ..., A_m)$, also need to be scheduled (if any of these finish times are 0, it is assumed to be job with a processing time of 0). The LPT rule is used to assign jobs in such a way that each machine has exactly one of the m additional jobs. After the scheduling is finished, the m additional jobs are moved to the beginning of the sequence on their respective machines. For more details, we refer the reader to Lee [1991].

4. Let C^*_{max} be the makespan value of jobs in $C1$ obtained at the end of Step 2. C^*_{max} is the determining makespan value (see Figure 1) realized on one or more machines. Step 4 in the algorithm is essentially a post-processing procedure. It is used to check if any of the m makespan jobs that finish before C^*_{max} can be delayed to accommodate jobs from S_3. If so, after this post-processing step, the makespan value remains unaffected but the jobs from C_2 can be scheduled earlier and the $\sum C_j$ improved.

5. Given the set of finish times $(A^{'}_1, A^{'}_2, ..., A^{'}_m)$ on the machines at the end of step 2, the remaining unscheduled jobs are scheduled in SPT order in step 3. It is known that for any given set of finish times SPT order is optimal for the $\sum C_j$ of the remaining unscheduled jobs in $C2$ (Sanlaville and Schmidt [1998]). Note that at this stage the problem reduces to minimizing the total completion time when machine availability times are different.

6. A total of $n_2 + 1$ schedules are generated by this procedure. We discard schedules that are dominated.

Finally, we present a counterexample (see Figure 2) that shows that the jobs in S_1 need not follow SPT ordering in a non-dominated solution, thus illustrating a situation where the heuristic can fail. Consider the following 2-machine instance: there are two 2 jobs in $C1$, each with a processing time of 10; and there are 3 jobs in $C2$ with processing times of 2, 2, and 4. Suppose we are at stage $k = 3$ in the algorithm. Thus all jobs from $C2$ are scheduled first followed by jobs in $C1$. Going by the SPT ordering of step 1 and followed then by the LPT ordering of step 2, the schedule on machine 1 (given by the processing times of the jobs) is: 2 4 10 while the schedule on machine 2 will be: 2 10. The $\sum C_j$ of the schedule is $2+2+6 = 10$ while the C_{max} of the schedule is 16. Consider now the schedule Machine 1: 2 2 10 and Machine 2: 4 10, which has the same $\sum C_j$ value of $2 + 4 + 4 = 10$ and a C_{max} value of 14. Clearly the latter schedule which does not follow an SPT schedule dominates the former. However, the latter schedule still achieves the same $\sum C_j$ as the former. We note that the genetic algorithm described in the next Section would likely generate this latter solution.

For a discussion on the characterization of the class of optimal schedules for $P||\sum C_j$ (which includes schedules that do not follow SPT ordering) we point the reader to Conway et al. [1967]. Indeed, such a study could lead to interesting theoretical considerations regarding the structure of a non-dominated solution.

Finally, the following theorem (proof in appendix) illustrates an important property of the $P|inter|ND(C_{max}, \sum C_j)$ problem. This proof pertains to every non-dominated *point* in the criteria space, where a point corresponds to a specific C_{max} (for jobs in $C1$) and $\sum C_j$ (for jobs in $C2$) value in the criteria space. The theorem implies that for every such non-dominated point, there

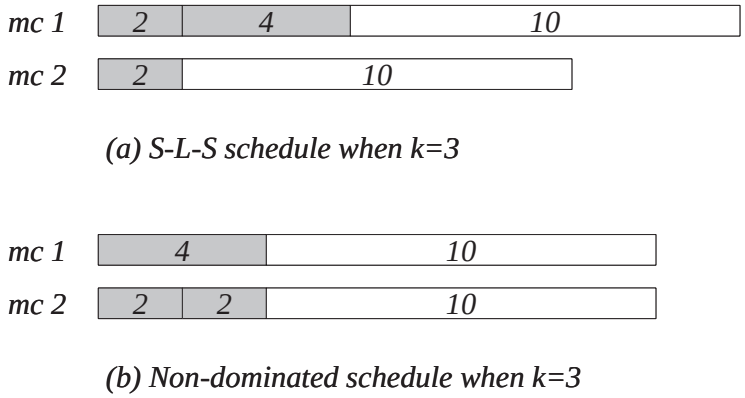

Figure 2: Counterexample disproving optimality of SPT ordering of subschedule S_1

exists a schedule such that the jobs in S_1 are the k shortest jobs, even though they may not be scheduled in SPT order.

Theorem 1. *For every non-dominated point in the criteria space, there exists a schedule that can be divided into three sets: S_1 consisting of the k shortest jobs in $C2$ (i.e. the $\sum C_j$ jobs); S_2 consisting of jobs in $C1$ (i.e. the C_{max} jobs); and S_3 consisting of remaining $n_2 - k$ jobs in $C2$.*

Note that there can exist multiple non-dominated schedules that do not follow this decomposition; our claim is that there exists at least one schedule that follows the properties described in the theorem. We now use the theorem to propose an encoding for the genetic algorithm, described in the next Section.

4 A genetic algorithm for the $P|inter|ND(C_{max}, \sum C_j)$ problem

We now propose a genetic algorithm (GA) to generate the set of non-dominated solutions (for an introduction to genetic algorithms see Goldberg [1989]). The discussion in Section 3 on the key characteristics of the problem of generating the set of non-dominated points forms the basis of the genetic algorithm. Indeed, the design of the GA is very similar to the iterative S-L-S heuristic but allows for a search of schedules other than those provided by the heuristic.

Every solution in the GA consists of 3 subschedules S_1, S_2 and S_3 (see Figure 1, and the discussion in Section 3). The first subschedule S_1 consists of k shortest processing time jobs from from $C2$. The second subschedule S_2 follows S_1 and consists of all n_1 jobs from $C1$. The third subschedule S_3 follows S_1 and S_2, and consists of $n_2 - k$ jobs from $C2$. The value of k can be said to control tradeoff between the C_{max} and $\sum C_j$ values: a high (low) value of k generally leads to solutions with a high (low) value of C_{max} and a low (high) value of $\sum C_j$. In the GA, we force the condition that there are a fixed number of solutions Q with k jobs in $S1$ for every value of k, from 0 to n_2. This strategy attempts to ensure that solutions are searched for in all parts of the non-dominated front and no biases get inadvertently introduced towards particular sections.

The proposed GA uses the same fitness ranking as the non-dominated sorting genetic algorithm (NSGA-II) of Deb et al. [2000]. We also refer to an implementation by Pasupathy et al. [2006] of

the NSGA-II algorithm for a bicriteria flow shop scheduling problem involving makespan and total completion time. The idea is to classify individuals in the population into successive non-dominated fronts, and using the non-domination rank value as an indicator of fitness. The tournament selection method (individuals are compared based on their non-domination ranks; see Brindle [1981] and Goldberg et al. [1990] for details of tournament selection) is used to create parents for the crossover operation.

In NSGA-II, if there is a tie, a secondary measure that selects solutions in sparser regions is used. Selection to the next generation also uses non-domination rank and the secondary criterion. Such a preference for solutions in sparse regions allows for coverage of all parts of the non-dominated front. For a review of measures proposed to achieve diversity along the non-dominated front see Landa Silva and Burke [2004a]. In our approach, we do not use a secondary criterion to break ties. Instead, we exploit the problem structure and control population sizes for certain sets of solutions to explore different parts of the front. Next, we describe the main steps of the GA.

4.1 Encoding and evaluation

Each chromosome consists of $1 + n_1 + k$ elements. Thus the length of the chromosome can vary depending on the number of jobs k in S_1. The first element of the chromosome indicates the k value, i.e. the number of jobs in S_1. It can therefore have any integer value from 0 to n_2. The next n_1 elements represent the makespan jobs and are assigned a value between 1 and m (including 1 and m). The assignments indicate the machines on which these n_1 jobs are to be processed; a partition of the set of jobs in $C1$ into at most m makespan jobs is obtained. The next k elements of the encoding represent machine assignments for the k shortest processing jobs of set $C2$ (these k jobs will make up subschedule S_1). We note here based on Theorem 1 that for every non-dominated point in the criteria space there exists a schedule that can be represented in this manner.

Figure 3 shows the encoding for a 10-job, 3-machine example (4 jobs in $C1$ with processing times 2, 3, 5 and 9; and 6 jobs in $C2$ with processing times 1, 2, 3, 7, 7 and 8). The encoding has 8 elements. The first element indicates that $k = 3$, which implies that S_1 consists of 3 of the shortest processing time jobs from $C2$ (in the example these are the jobs with processing times 1, 2 and 3). Elements 6-8 of the encoding indicate the machine assignments for these 3 jobs. Elements 2-5 (with values 3, 2, 1 and 3 respectively) indicate that jobs 1 and 4 of $C1$ are to be processed on machine 3, while jobs 2 and 3 are to be processed on machines 2 and 1, respectively.

Note that the encoding does not consider the $n_2 - k$ jobs in $C2$ that are in subschedule S_3. This is because it is known that it is optimal for the $n_2 - k$ jobs in $C2$ to follow SPT ordering on the m machines. We demonstrate next how a chromosome is translated into a complete schedule of jobs in the two sets so it can be evaluated based on its C_{max} and $\sum C_j$ values. We use the example given above to illustrate the construction of the subschedules.

4.1.1 Creating S_1

From the last k elements of the encoding we obtain the machine assignments for the k shortest processing time jobs in $C2$. On each machine jobs are then scheduled in SPT order (as we noted earlier in Section 3, jobs in $C2$ in a non-dominated solution are processed in SPT order on any given machine; this result is an extension of the property of single machine sequences given in Baker and Smith [2003]). Figure 4 (a) shows how subschedule S_1 is created from the last 3 elements of the encoding given in Figure 3.

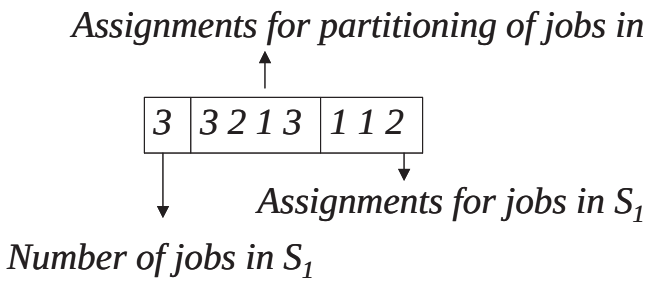

Figure 3: Encoding scheme for each chromosome in the genetic algorithm

In the counterexample at the end of Section 3, we showed that SPT ordering on the m machines for the jobs in S_1 may not be optimal. The GA's method of assigning jobs in S_1 to machines ensures that different machine assignments are attempted and solutions better in the non-dominated sense (if such solutions exist) than the one produced by the iterative S-L-S heuristic are explored.

4.1.2 Creating S_2

Let $(A_1, A_2, ..., A_m)$ be the vector of finish times on the machines once subschedule S_1 has been created. The subschedule S_2, which "follows" S_1, consists of all the jobs in the set $C1$, i.e. the jobs for which makespan is to be minimized. The n_1 elements of the encoding starting from the 2nd element assigns jobs in $C1$ to machines, thus partitioning the jobs into a set of at most m makespan jobs. Each makespan job is nothing but an aggregation of the processing times of jobs that have been assigned the same machine number.

Figure 4 (b) shows an example of how S_2 is constructed given that the finish times from S_1 are (3,3,0); the 4 jobs in $C1$ are partitioned into 3 makespan jobs with aggregate sizes of 5, 3 and 11. Here again, the GA differs from the S-L-S heuristic which schedules the jobs in a fixed MLPT sequence. The GA explores many more partition possibilities than the S-L-S heuristic. Indeed, as we shall see in the experimental results Section, the GA is able, with its random assignments of jobs, in $C1$ to tradeoff the makespan value for improvements in the total completion time and generate schedules not considered by the S-L-S heuristic.

4.1.3 Creating S_3

As stated before, the encoding does not explicitly consider the remaining unscheduled $n_2 - k$ jobs in $C2$. If $(A_1^{'}, A_2^{'}, ..., A_m^{'})$ represents the finish times on the machines after subschedules S_1 and S_2 have been created, then the $n_2 - k$ jobs are scheduled in SPT order (i.e. shortest processing time job is scheduled on the earliest available machine until all jobs are scheduled). The problem of creating an optimal S_3 is equivalent to the problem of minimizing $\sum C_j$ given that all machines are not available simultaneously. By expressing the $\sum C_j$ in terms of the positions of the jobs, as illustrated in Conway et al. [1967], it can be shown that the SPT ordering minimizes the $\sum C_j$ of the $n_2 - k$ jobs in $C2$.

Figure 4 (c) shows an example where the 3 remaining jobs in $C2$ (with processing times 7, 7 and 8) are scheduled on the earliest available machines. The complete schedule is now available

for an evaluation of its makespan and completion time. In the iterative S-L-S heuristic, too, the remaining $n_2 - k$ jobs are scheduled using SPT ordering. However, the quality of the schedule is determined heavily by the finish times $(A_1^{'}, A_2^{'}, ..., A_m^{'})$ obtained after the construction of S_1 and S_2.

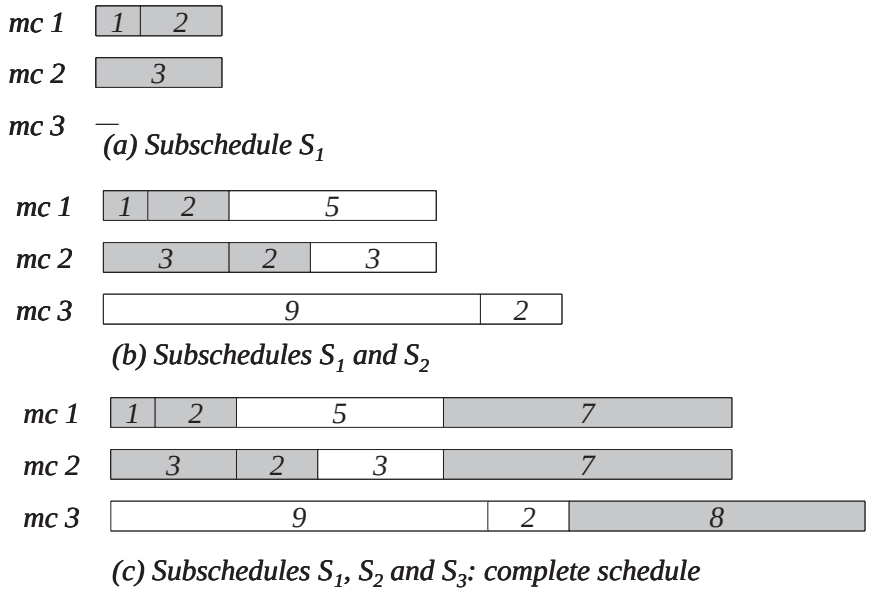

Figure 4: Translation of an encoding into a schedule (values represent processing times)

4.2 Population specifics

We maintain a fixed number of solutions Q for each value of k. Since k can take on values from 0 to n_2, there are $Q(n_2 + 1)$ chromosomes in the population in any given generation. A disadvantage of this approach is that the size of the population grows as n_2 increases, but it is also likely, given the structure of the problem, that the number of non-dominated solutions increases with n_2.

The initial population is randomly generated. To speed up the convergence of the algorithm, we also introduce into the initial population the $n_2 + 1$ solutions generated by the iterative S-L-S heuristic (one solution for each value of k).

4.3 Ranking of chromosomes

Once all the chromosomes are evaluated as described in Section 4.1, they are ranked by the following procedure. All solutions that are non-dominated in a given generation of the genetic algorithm are assigned a rank of 1; the next set of non-dominated solutions (i.e. those dominated by solutions whose rank is 1, but non-dominated amongst the rest) are assigned a rank of 2; the procedure is continued until all chromosomes are ranked. Thus, successive non-dominated fronts are maintained in a population in any given generation. The rank of a chromosome is a measure of its fitness; the

lower its rank the better chance it has of getting selected in the next generation and also to pass on its characteristics through offspring to future generations.

4.4 Crossover

The well-known single point crossover is used to create offspring. However, since chromosomes can vary in the length of their encoding, two types of crossover operations are used. The crossovers are shown in Figure 5. In both types of crossover, each parent is chosen using tournament selection in which a pair of solutions are picked randomly from the population and their ranks compared; the solution with the lower rank wins the tournament. If the ranks are equal then one of the two solutions is randomly chosen with equal probability. In the first type, the crossover point (randomly chosen) occurs only in the first $1 + n_1$ elements of the encoding of two parent chromosomes. Such a crossover is allowed even when the parents have different number of jobs in S_1 (i.e. different values in the first element of their encoding). This type of crossover exchanges different partitions of jobs in $C1$ between two parents to create two offspring. In the second type of crossover, the crossover point (randomly chosen) is always within the last k elements of the encoding. Moreover, the crossover is carried out only between parents that have the same k values (i.e. schedules that have the same number of jobs in S_1). This type of crossover exchanges different machine assignments of the k shortest processing time jobs in $C2$ between two parents to create two offspring. We perform crossovers until there are Q offspring for each value of k. Thus the set of candidate solutions is doubled after crossover.

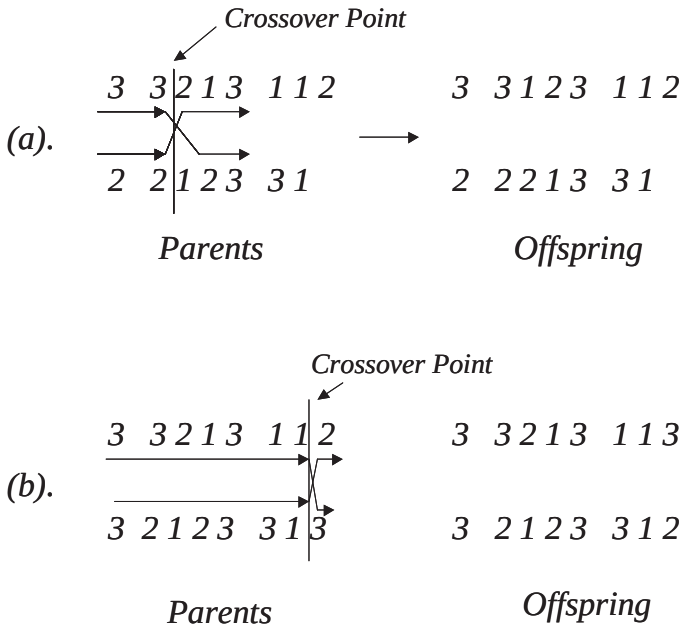

Figure 5: Visual illustration of the two types of crossover

4.5 Mutation

We randomly change the value of an element of a chromosome's encoding in the mutation operation. A random number r is generated for every chromosome in the offspring generated from the crossover step. If r is less than a prefixed probability p_m, then a randomly chosen element of the chromosome's encoding is changed to a different value. However, we exclude the first element of the chromosome from undergoing mutation in order to maintain a fixed number of chromosomes for every value of k.

4.6 Selection of chromosomes into the next generation

After crossover, there are $2Q(n_2+1)$ chromosomes in the population, or $2Q$ chromosomes for every value of k. These chromosomes form the candidate set from which $Q(n_2+1)$ chromosomes need to be chosen. The candidate set is evaluated for its makespan and total completion time values and each chromosome is assigned a rank equal to the non-dominated front it belongs to. For every k, $Q/2$ of the best ranked solutions are chosen for the next generation (if the number of rank 1 solutions is higher than $Q/2$, then all of them are picked). The remaining $Q/2$ solutions (or how many ever remain after the picking of the best) are chosen randomly from the rest of the candidate set of solutions (each solution has an same probability of being picked) with the condition that they have the same value of k. In addition an archive of non-dominated solutions is maintained and updated at the end of every generation.

Thus the overall procedure of the GA involves maintaining fixed quantities of local populations that are geared to cover all areas of the front of non-dominated solutions, while simultaneously ensuring that the local populations are evaluated, ranked and compared "globally".

4.7 Summary of the GA

We now provide a summary of the main steps of the GA:

1) Insert n_2+1 schedules (one for each k) from the iterative S-L-S heuristic into the initial population of the genetic algorithm. Create the rest of the initial population randomly: that is, for each of the assignments that need to be done for the encoding, each job has is randomly assigned to one of the machines. This random assignment is based on a discrete uniform distribution: each machine has an equal likelihood of being chosen. At the end of this step there are Q chromosomes for every k, $k = 0, ..., n_2+1$. Set the number of generations $i = 0$.

2) Evaluate the population to determine makespan and total completion time values. Divide the population into successive non-dominated fronts.

3) Determine parent pairs using the tournament selection method and perform crossover operations to produce Q offspring for every k. Each of the two types of crossover contribute $Q/2$ offspring.

4) Based on the probability of mutation p_m, perform mutations on the offspring produced from the crossover.

5) The offspring and the population of parents form the candidate set for the next generation. They are evaluated and ranked based on the non-dominated front they belong to (similar to step 2).

6) For the next generation, choose $Q/2$ of the best ranked chromosomes for each k and the rest as described in Section 4.6. Maintain an archive of non-dominated solutions generated so far. Set

$i = i + 1$. If $i = N_{gen}$, the pre-specified number of generations, then stop, else go to step 2.

4.8 Setting GA parameters

To set the parameters of the GA, we conducted separate designed experiments for small and large instances. For small instances we determined the following settings: $Q = 3$ (high and low values used were 2 and 7, 3 was chosen after a search was performed over this range), and $p_m = 0.2$ (high and low values were 0.05 and 0.5), and $N_{gen} = 40$. For large instances we determined the following settings: $Q = 4$ (high and low values used were 2 and 10), and $p_m = 0.2$ (high and low values were 0.05 and 0.5), and $N_{gen} = 200$. The probability of crossover was set to 1 for both small and large instances.

4.9 A note on other GA approaches

In addition to the special encoding proposed in this research, we also attempted a genetic algorithm using the traditional, widely used parallel machine encoding found in Loukil et al. [2005]. The encoding partitions the set of jobs and determines also the position of the jobs on the machines. The encoding thus gives a complete schedule of jobs unlike the special encoding proposed in this research in which much of the schedule is not explicitly modeled and is constructed based on aspects of the problem structure. It was determined, based on preliminary computational experiments, that the genetic algorithm that uses this traditional, generic encoding performs quite poorly when compared with the GA proposed. This is because 1) the traditional encoding bypasses some of the key properties of a non-dominated solution and 2) the crossover operation does not retain some of the desirable characteristics of parents that enable good new solutions to be produced. This suggests that the problem of interfering job sets may not always lend itself to generic solution techniques that have used in the literature for traditional parallel machine bicriteria problems. Problem specific methods that exploit structural aspects - such as the GA proposed in this research - are likely to be more effective.

5 Experimental results for small-sized instances

We test the S-L-S heuristic and the bicriteria genetic algorithm proposed in the previous section on small-sized instances. For comparison, we use an integer program (IP) to generate the entire set of non-dominated solutions. The set of solutions provided by the IP serves as a reference set.

If L is a prefixed value of makespan that cannot be exceeded, then the following is a time-indexed formulation for the parallel machine problem with interfering job sets involving C_{max} and $\sum C_j$ (the time-indexed formulation for scheduling problems was originally proposed by Sousa and Wolsey [1992]). In the formulation the decision variable x_{it} is 1 if job i starts at time point t, and 0 otherwise. L denotes the parameter that decides the makespan value. T is the maximum possible start time given by $\sum_{i=1}^{n} p_i$.

Time indexed formulation for $P|inter|\epsilon(\sum C_j|C_{max})$

$$\min \sum_{i \in C2} \sum_{t=0}^{T} x_{it}(t + p_i)$$

Such that:

$$\sum_{t=0}^{L-p_i} x_{it} = 1, \qquad i \in C1$$

$$\sum_{t=0}^{T} x_{it} = 1, \qquad i \in C2$$

$$\sum_{i \in C1 \cup C2} \sum_{s=max(t-p_i+1,0)}^{t} x_{is} \leq m \qquad \forall t \in T$$

$$x_{it} \in \{0,1\} \qquad \forall i \in C1 \cup C2, \forall t \in T$$

In the above formulation, the objective function calculates the completion time of only the jobs in $C2$. The first two sets of constraints ensures that each job is scheduled exactly once; the first set also ensures that no job in $C1$ completes after L. The third set of constraints ensures that no more than m jobs are scheduled in any processing time window. The last set of constraints are integrality constraints on the decision variables.

The optimal solution to the formulation minimizes the $\sum C_j$ for jobs in $C2$ but simultaneously ensures that the jobs in $C1$ do not exceed the pre-fixed makespan value of C_{max} (L). In the terminology of T'Kindt and Billaut [2002] this is the ϵ constraint approach in bicriteria optimization written as $\epsilon(\sum C_j|C_{max})$ in the γ part of the $\alpha|\beta|\gamma$ notation. To generate the set of non-dominated points, we first set $L = T$ and minimize $\sum C_j$. Suppose the resulting makespan value is L_1. We next set $L = L_1 - 1$ and reoptimize. We continue in this fashion until all the non-dominated points have been generated. This is, however, a computationally intensive procedure and is feasible only for small-sized problem instances.

We consider 20-job, 2-machine problem instances (with 10 jobs in each set), and 30-job, 5-machine problems (with 15 jobs in each set). The processing times for each instance are generated using a discrete uniform distribution from 1 to 10. Ten instances are created per category. In addition to finding all non-dominated solutions, we also determine the set of extreme points. The extreme points of an efficient frontier are a subset of the set of non-dominated solutions and form the lower hull of the non-dominated solutions plotted in the objective space. In multicriteria optimization, it is often a goal to generate the set of extreme points, as generating the entire set of non-dominated solutions is often an intractable problem. It is also worthwhile to note that the optimal solution to a composite linear objective function $\alpha z_1 + (1 - \alpha)z_2$ (where z_1 and z_2 are two criteria to be minimized and $0 \leq \alpha \leq 1$) is an extreme point. In our experimental results, we demonstrate that while the set of extreme points guarantees Pareto optimality, it is not necessarily a "good" approximation of the set of non-dominated points, and is bettered by our heuristic approaches. Carlyle et al. [2003] state that a "good approximation typically consists of a set of *diverse* solutions that are *uniformly* distributed along the efficient frontier, and which are also *close* to the efficient frontier." Our judgements of the solution sets under study in this section, whether subjective or quantitative, are based on how close the sets are to realizing these three properties. Our analysis is helped by the availability of a reference set: the set of all non-dominated solutions given by the ϵ-constraint IP.

5.1 Results for two instances

We first report complete results (see Tables 1 and 2) for two instances: one a 20-job 2-machine instance, and the other a 30-job 5-machine instance. We also analyze the results graphically in the objective space (see Figures 6 and 7).

	IP		**S-L-S**		**GA**		
No	**Makespan**	**TCT**	**Makespan**	**TCT**	**Makespan**	**TCT**	**k**
1	**20**	**334**	**20**	**334**	**20**	**334**	0
2	21	333			21	333	0
3	22	314	22	314	22	314	2
4	**24**	**295**	**24**	**295**	**24**	**295**	3
5	25	294			25	294	3
6	26	293			26	293	3
7	**27**	**275**	**27**	**275**	**27**	**275**	4
8	31	256	**31**	**256**	31	256	4
9	32	255			32	255	5
10	33	254			33	254	5
11	**34**	**236**	**34**	**236**	**34**	**236**	6
12	38	217	38	217	38	217	7
13	39	216			39	216	7
14	40	215			40	215	7
15	41	214			41	214	7
16	**42**	**197**	**42**	**197**	**42**	**197**	8
17	47	179	47	179	47	179	9
18	48	178			48	178	9
19	49	177			49	177	9
20	50	176			50	176	9
21	51	175			51	175	9
22	**52**	**158**	**52**	**158**	**52**	**158**	10

Table 1: Comparison of criteria values generated for a 20-job, 2-machine instance. The values in bold indicate the extreme points. For every pair of criteria values, the corresponding k value for the GA solution (i.e. the number of jobs in S_1) is also listed

Table 1 shows that the GA generates the set of all non-dominated solutions exactly. The S-L-S heuristic does generate one schedule per each k. However, during the post-processing step (4), a schedule with k=1, 2, or 3 may eventually end up as a $k = 4$ schedule. This can be observed with the $k = 4$ S-L-S solutions in Table 1. After the three initial steps of the algorithm it is found that more jobs from $C2$ can be added to S_1 without increasing the makespan value. Thus, in the final set of solutions produced by S-L-S, multiple solutions appear for the same k. But it still is true that for each k, the S-L-S schedule produces only one schedule.

Overall, the iterative S-L-S heuristic generates non-dominated solutions but delivers only a subset of the solutions, while the GA is able to generate a greater number of non-dominated solutions for certain k values. Thus the GA can especially useful if the decision-maker chooses to

	IP		**S-L-S**		**GA**		
No	**Makespan**	**TCT**	**Makespan**	**TCT**	**Makespan**	**TCT**	**k**
1	**15**	**343**	**15**	**343**	**15**	**343**	0
2	**16**	**284**	16	298	**16**	**284**	4
3	17	268	17	268	17	268	5
4	**18**	**241**	18	254	**18**	**241**	7
5	**19**	**226**	19	240	**19**	**226**	8
6	20	223	20	225	20	223	8
7	21	207	21	210	21	207	8
8	22	193	22	193	22	193	10
9	**23**	**181**	**23**	**181**	**23**	**181**	11
10	24	180			24	180	11
11	25	168	25	168	25	168	12
12	26	166			26	166	12
13	27	152	27	152	27	152	13
14	28	149			28	149	13
15	29	134	29	136	29	134	14
16	30	130			30	131	14
17	**31**	**118**	**31**	**118**	**31**	**118**	15

Table 2: Comparison of criteria values generated for a 30-job, 5-machine instance. The values in bold indicate the extreme points. For every pair of criteria values, the corresponding k value for the GA solution (the number of jobs in S_1) is also listed

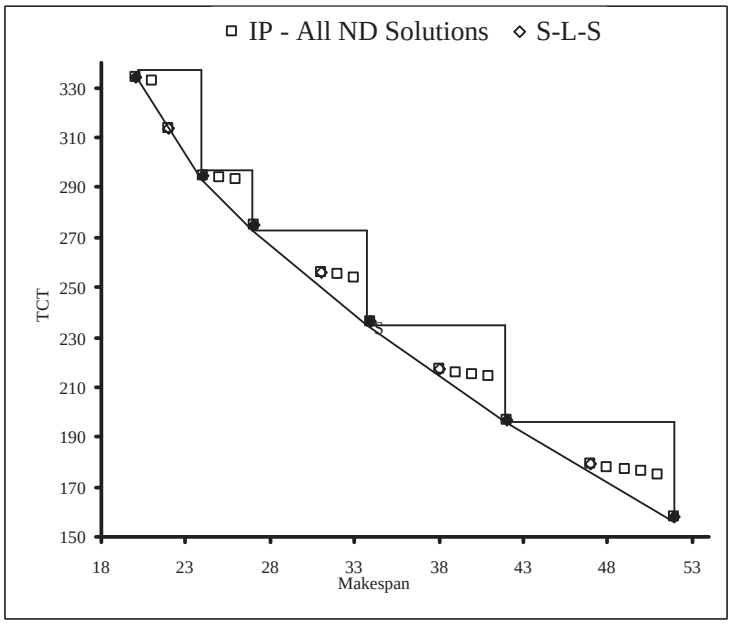

Figure 6: Solutions in objective space: 20-job, 2-machine instance

look for more options in a given region of the objective space. The pairs of numbers in bold represent the extreme points. Clearly the extreme points miss a significant number of non-dominated points; the points generated by the S-L-S heuristic have improved coverage.

Figure 6 plots the ϵ-constraint IP set along with the S-L-S set in the objective space (the GA set is not plotted to allow for clarity; moreover, the GA set is identical to the ϵ-constraint IP set). The extreme points are shaded black. The triangles in the figure between any two adjacent extreme points indicate regions where *non-supported* solutions can be found. Non-supported points are non-dominated points that do not lie on the efficient frontier. From the figure, it is clear that the number of non-supported solutions progressively increases as we move from the top left of the efficient frontier to the bottom right. In these bottom right solutions there are more jobs in S_1, i.e. the k values are high. With a higher number of jobs, there are a greater number of assignments of jobs in S_1 to machines, which leads to schedules with small tradeoffs in $\sum C_j$ and C_{max} for the same k. These small tradeoffs result in a higher number of non-supported schedules. The ability to capture these small tradeoffs, and hence the non-supported solutions, is one key aspect that differentiates the GA from the S-L-S heuristic. This aspect also indicates that Q, the number of solutions per value of k, need not be fixed as it currently is but could be variable.

Table 2 shows that for the 30-job 5-machine instance, the S-L-S heuristic covers the non-dominated solutions better, but falls short a bit in solution quality: some of the the solutions it generates are weakly non-dominated (i.e. there exists no other solution that is better in *both* criteria, but there may exist a solution that is equal in one and better in the other). However, the S-L-S still outdoes the set of extreme points (in bold under the IP column) in terms of coverage. The GA once again performs very well: except for solution 16 (30,131), it matches the ϵ-constraint IP set exactly. As in the 20-job, 2-machine instance, it provides multiple solutions for certain values

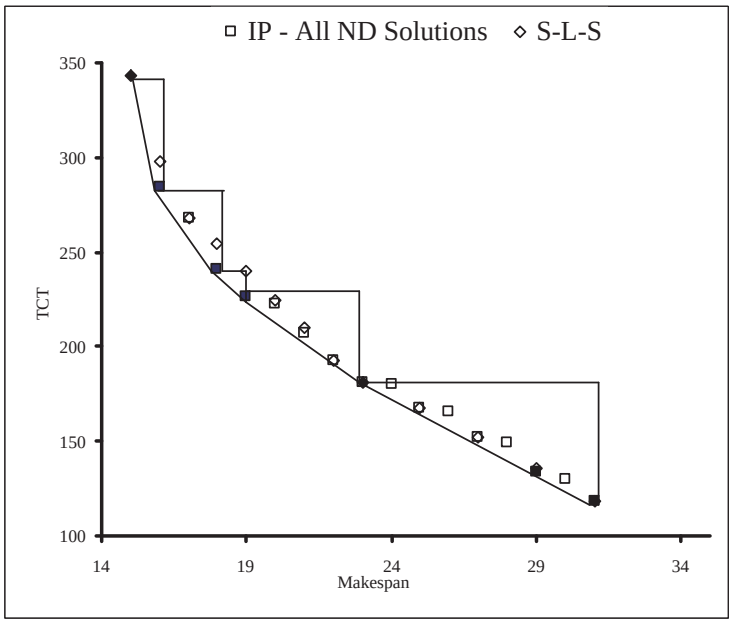

Figure 7: Solutions in objective space: 30-job, 5-machine instance

of k (unlike the S-L-S heuristic), which is useful if the decision were to look for possible options in a particular region of interest.

Figure 7 shows the ϵ-constraint IP set, the set of extreme points (shaded black), and the S-L-S set in the objective space. As before the the number of non-supported solutions progressively increases as we move from the top left of the efficient frontier to the bottom right. The extreme points are fewer in the bottom right of the frontier.

5.2 A quantitative comparison of the solution sets

While the two instances analyzed are fairly representative of the other instances in their respective categories, we now provide an objective quantification of the solution sets. A number of measures exist in literature for comparing solution sets. Carlyle et al. [2003] provide a classification of 20 such measures.

For our comparison purposes, we use the measures of Cyzak and Jaszkiewicz [1998]. These measures are appropriate for our situation as they require a reference set R (which in our case is the IP set) and allow for comparisons in terms of diversity, uniformity and closeness to the reference set. Czyzak and Jaszkiewicz propose two distance measures, *Dist1* and *Dist2*. If M is the solution set whose quality is to be quantified, then the measures are based on calculating $c(x, y)$, the "distance" between a point $x \in M$ and a point $y \in R$:

$c(x, y) = max_{j=1...T}\Big\{0, (1/w_j) \times (f_j(y) - f_j(x))\Big\}$,

where T is the total number of criteria, and $f_j(y)$ and $f_j(x)$ are the objective function values of the jth criterion. Thus, if x and y have identical criteria values, $c(x, y) = 0$, else $c(x, y)$ is equal to the criterion that has the maximum weighted deviation. Here the weight w_j for a given criterion

j is the range of the criterion in the reference set (i.e. the difference between the maximum and minimum value of the criterion).

Next, *Dist1* and *Dist2* are defined as follows:

$Dist1 = (1/|R|) \sum_{y \in R} \left\{ min_{x \in M} \{c(x,y)\} \right\}$ $Dist2 = max_{y \in R} \left\{ min_{x \in M} \{c(x,y)\} \right\}$

The *Dist1* calculation works as follows. For every point in $y \in R$, the point in M with the lowest value of $c(x,y)$ (lowest "distance" in a sense) is determined. These distances are then summed up, and give a measure of the average proximity of the closest points in X from R. The calculation for *Dist2* gives the worst case value: it first determines the points in M closest to points in R and then determines the farthest one among them.

		GA			**S-L-S**		
Inst.	**Y**	***Dist1***	***Dist2***	**No. of pts**	***Dist1***	***Dist2***	**No. of pts**
1	22	0	0	22	0.006	0.023	10
2	18	0	0	18	0.011	0.043	11
3	16	0	0	16	0.008	0.048	9
4	16	0	0	16	0.010	0.048	9
5	24	0	0	24	0.006	0.022	11
6	20	0	0	20	0.007	0.037	11
7	25	0	0	25	0.007	0.029	11
8	20	0	0	20	0.004	0.015	9
9	20	0	0	20	0.005	0.021	10
10	17	0	0	17	0.003	0.012	10
	Avg	0	0		0.007	0.030	
	Std. Dev	0	0		0.002	0.013	
		Extreme Points Only (from IP)			**IP: LinComb**		
Inst.	**Y**	***Dist1***	***Dist2***	**No. of pts**	***Dist1***	***Dist2***	**No. of pts**
1	22	0.041	0.108	7	0.062	0.222	5
2	18	0.040	0.096	7	0.091	0.261	3
3	16	0.011	0.048	6	0.080	0.238	3
4	16	0.024	0.136	6	0.081	0.153	3
5	24	0.037	0.089	8	0.065	0.200	5
6	20	0.039	0.148	7	0.087	0.200	3
7	25	0.025	0.111	8	0.124	0.304	3
8	20	0.043	0.105	7	0.083	0.222	3
9	20	0.052	0.108	6	0.076	0.222	4
10	17	0.037	0.101	7	0.088	0.222	3
	Avg	0.035	0.105		0.084	0.224	
	Std. Dev	0.012	0.027		0.017	0.040	

Table 3: *Dist1* and *Dist2* values for the various approaches for the 20-job, 2-machine instances, and also the number of solutions generated by each approach. Note that Y denotes the total number of non-dominated points generated by the IP (ϵ-constraint approach) for each instance.

Tables 3 and 4 give the *Dist1* and *Dist2* values for the comparison of the IP (ϵ-constraint)

		GA			S-L-S		
Inst.	**Y**	***Dist1***	***Dist2***	**No. of pts**	***Dist1***	***Dist2***	**No. of pts**
1	17	0.0003	0.0044	17	0.0157	0.0622	13
2	19	0.0020	0.0127	19	0.0282	0.0844	11
3	15	0.0010	0.0052	15	0.0449	0.1094	11
4	16	0.0014	0.0147	16	0.0074	0.0368	12
5	16	0.0008	0.0043	15	0.0165	0.0667	12
6	16	0.0003	0.0053	15	0.0249	0.0667	12
7	19	0.0003	0.0052	19	0.0139	0.0942	14
8	18	0.0012	0.0105	18	0.0219	0.0588	13
9	15	0.0000	0.0000	15	0.0068	0.0290	13
10	19	0.0005	0.0087	19	0.0059	0.0261	13
	Avg	0.001	0.007		0.019	0.063	
	Std. Dev	0.001	0.004		0.012	0.028	
		Extreme Points Only (from IP)			**IP: LinComb**		
Inst.	**Y**	***Dist1***	***Dist2***	**No. of pts**	***Dist1***	***Dist2***	**No. of pts**
1	17	0.0477	0.1422	6	1.2967	0.0763	3
2	19	0.0429	0.1561	7	5.5084	0.2899	2
3	15	0.0655	0.1979	4	3.7500	0.2500	2
4	16	0.0687	0.2000	5	1.8152	0.1135	3
5	16	0.0460	0.1333	7	2.4026	0.1502	2
6	16	0.0622	0.2000	6	1.5958	0.0997	3
7	19	0.0360	0.1414	8	5.0288	0.2647	2
8	18	0.0653	0.1765	5	2.0835	0.1157	3
9	15	0.0801	0.2143	5	1.4244	0.0950	3
10	19	0.0558	0.1667	6	3.9275	0.2067	2
	Avg	0.057	0.173		2.883	0.166	
	Std. Dev	0.014	0.029		1.551	0.079	

Table 4: Dist1 and Dist2 values for the various approaches for 30-job, 5-machine instances, and also the number of solutions generated by each approach. Note that Y denotes the total number of non-dominated points generated by the IP (ϵ-constraint approach) for each instance.

method, the GA, the S-L-S heuristic, the set of extreme points, and IP:LinComb, which is an approach that models the criteria as a linear combination $\alpha C_{max} + (1 - \alpha) * \sum C_j$, and reports the optimal solution for every α between 0 and 1 in increments of 0.05. The IP:LinComb thus will generate a set of extreme points but may not generate all of them as it may skip certain α ranges over which an extreme point is optimal. Nevertheless, IP:LinComb is included for comparison purposes; it is meant to simulate the situation where a solution set is generated by attempting different convex combinations of the criteria.

In both types of instances (20-job 2-machine and 30-job, 5-machine), the GA solution set outperforms all the other solution sets. Indeed, in each case, it generates just as many solutions as the ϵ-constraint IP does (compare columns Y and the No. of pts column for the GA). The distance measures attest to the fact that the points generated by the GA are very close to ϵ-constraint IP set, and, for 20-job, 2-machine instances, match exactly with the IP set.

The S-L-S approach, while not as good as the GA, still provides better solution sets in terms of both the number of points generated and the values of *Dist1* and *Dist2* than the set of extreme points and the IP:LinComb approach. The reasons for this can be traced back to the discussion on Figures 6 and 7; it is clear that the extreme points do not provide enough representation in certain regions of the efficient frontier. It can also be concluded that the IP:LinComb set, in spite of providing non-dominated solutions, performs poorly since it is able only to generate a few solutions. Indeed, in some of the instances, it is able to generate only 2 points (these are the lexicographic points), despite the extensive search over the range of α values.

5.3 A note on computation times

The IP approach for generating the extreme points or for generating the entire set of non-dominated solutions is computationally prohibitive and is feasible only for small sized instances. For 20-job 2-machine cases, the IP consumed, on an average, around 30,000 seconds of CPU time (nearly 9 hours) to generate a set of non-dominated solutions (see Table 5 for average computation times). We used the CPLEX 9.1 solver (the model was coded in AMPL) with default settings on a Linux machine with 2.4 Ghz and 1 GB RAM. The S-L-S heuristic was the quickest approach in terms of computation time; each set of solutions was generated in just a fraction of a second of CPU time. In comparison, the GA is computationally more intensive, but for the small instances tested in this section, each solution set was generated in approximately half a second of CPU time. Thus both the S-L-S heuristic and the GA not only provide very good solution qualities but are also computationally feasible.

	IP		**S-L-S**		**GA**	
	Avg	**Std.Dev**	**Avg**	**Std.Dev**	**Avg**	**Std. Dev**
20-job, 2-mc	30,560.9	54399.5	0.006	0.010	0.4646	0.036
30-job, 5-mc	14,442.3	10,023.3	0.012	0.012	0.5477	0.044

Table 5: Computation time in seconds for the various approaches

6 Experimental results for large-sized Instances

We now test the GA and the heuristic on large-sized problem instances. We use 2 different settings: 100-job 5 machine instances and 100-job 10-machine instances. In each case, the number of jobs in the two sets are equal. Processing times, as before, are generated using a discrete uniform distribution from 1 to 10. For comparison purposes, we use an integer program, but since for these large instances, the ϵ-constraint IP approach is computationally infeasible, we stop each run of the IP after 30 minutes (1800 seconds) of CPU time, and use the best integer solution. Thus the solutions generated by this approach are not guaranteed to be optimal. We used the same experimental platform as above. For the S-L-S heuristic and the GA we used Microsoft Visual C++ programs on a 256-mb CPU with 256 mb RAM.

Since it is difficult graphically to tell the difference between the approaches for a large instance, we provide a quantitative comparison in Tables 6, 7, 8 and 9. The distance measures are calculated as explained in the previous section. Since we do not have an exact reference set for our instances, we create a reference set by combining the solutions from the GA and the IP and choosing the non-dominated solutions among them.

To put *Dist1* and *Dist2* values in the Tables in perspective consider the following example. Suppose we want to have an idea of the quality of the total completion time of the GA solution set (the quality of makespan can be determined in a similar way; however, it is observed that the GA does not deviate from the reference set in makespan: see Tables 1 and 2 for an illustration of this for small instances). The total completion time ranges from 669 to 1973 in the reference set in instance 6 of the 100-job 10-machine instances. The *Dist2* value of the GA for this instance is 0.02147; it is also the highest *Dist2* value for the GA. The value can interpreted as follows. Among all points in the GA set that are closest to the reference set, the farthest point is roughly $0.02147 \times (1973 - 669) = 28$ total completion time units away from its closest solution in the reference set. In the worst case, therefore, this translates to a 4 percent higher total completion value if we assume this deviation is from 669, and 1.3 percent if the deviation is assumed to be from 1973. *Dist1*, being an average measure, is generally smaller but can be interpreted in a similar way.

In general, based on the discussion above and from the tables, it can be concluded that the distances from the reference set are small for both the S-L-S and GA approaches. Indeed, in the 100-job 5-machine instances, the GA produces better distance values than the IP approach, and the S-L-S heuristic performs comparably. But the S-L-S heuristic falls short when it comes to the number of non-dominated points generated in these instances. However, it has an advantage over both the GA (which takes roughly 90 seconds of CPU time per instance) and the IP in terms of computation time. It must also be noted that the the convergence of the GA is faster since its initial population consists of $n_2 + 1$ solutions from the S-L-S heuristic.

To obtain further justification for these results, we tested the solution sets with an additional diversity measure proposed by Landa Silva and Burke [2004a] that does not require a reference set. For any given set of non-dominated solutions, the diversity is evaluated as follows. We first determine the "centroid" or mean of each two criteria in the set, and calculate the squared deviation of each individual point from the centroid. We used this measure to compare the non-dominated sets obtained from the three approaches. In general, the diversity measures were close for the 100-job 10-machine instances, while IP and the GA were clearly superior for 100-job 5-machine instances.

The superior performance of the IP-approach in the 10-machine instances as compared to the

5-machine instances can be explained by the nature of the formulation. In general the higher the number of machines, the smaller the value of T in the IP formulation (T is the maximum possible time point at which jobs can start). Hence, the number of variables is smaller in the 10-machine case than in the 5-machine case, and the IP computes a better integer solution in the allotted 30 minutes of CPU time. The tractability of the time-indexed IP approach is also dependent heavily on the processing times. This is because the number of variables in the IP is pseudopolynomially many. If processing times were more widely distributed (from say 1 to 100), it becomes difficult to generate the set of non-dominated solutions by solving an IP for each possible C_{max} value. We note here that our integer program is rather basic and that the exploration of alternative, polynomial-sized, formulations and refinements to current formulation (such as eliminating symmetry in the integer program, which has the potential to improve running times) are good directions for future research.

In contrast, both the S-L-S and the GA run in polynomial time; their complexities are independent of the processing time values. Table 10 gives the average CPU seconds required for both the S-L-S heuristic and the GA. Clearly, the S-L-S heuristic is extremely efficient, but the GA is able to generate a solution set within two minutes of CPU time as well, making it an attractive choice.

Finally a point also needs to be made about presenting a large number of non-dominated solutions to the decision-maker since this approach of presenting too many points can be confusing. One possible way of making this easier is to present information sequentially to the decision-maker - present, perhaps only the extreme points at first and allow the decision maker to guide the search for other points, if the need arises, in subsequent iterations.

		GA			**S-L-S**			**IP**		
Inst.	**Ref**	**Dist 1**	**Dist 2**	**No.**	**Dist 1**	**Dist 2**	**No.**	**Dist 1**	**Dist 2**	**No.**
1	54	0.000008	0.000424	54	0.000464	0.002971	40	0.000511	0.006367	54
2	54	0.000000	0.000000	54	0.000386	0.002275	40	0.000456	0.005309	55
3	54	0.000007	0.000373	54	0.000318	0.001867	42	0.001093	0.008962	55
4	53	0.000000	0.000000	53	0.000318	0.001719	40	0.000928	0.008597	55
5	60	0.000000	0.000000	60	0.000594	0.003196	43	0.000624	0.007306	61
6	46	0.000074	0.002971	46	0.002558	0.021277	36	0.000286	0.002547	46
7	64	0.000000	0.000000	64	0.000375	0.001818	45	0.000875	0.008364	64
8	51	0.000013	0.000421	51	0.000600	0.000392	40	0.000681	0.006210	52
9	53	0.000002	0.000120	53	0.000590	0.000512	40	0.000543	0.003300	53
10	50	0.000000	0.000000	50	0.000248	0.001404	39	0.000241	0.003158	50
Avg		0.00001	0.00043		0.00065	0.00374		0.00062	0.00601	
Std. Dev		0.00002	0.00091		0.00068	0.00623		0.00028	0.00237	

Table 6: Dist1 and Dist2 values for the approaches for 100-job, 5-machine instances with number of solutions generated by each approach.

7 Conclusions and future research

The problem of interfering job sets has received surprisingly little attention in the literature despite its ability to model several real-world situations. We consider the problem of interfering job sets

		GA			S-L-S			IP		
Inst.	**Ref**	**Dist 1**	**Dist 2**	**No.**	**Dist 1**	**Dist 2**	**No.**	**Dist 1**	**Dist 2**	**No.**
1	29	0.00029	0.00254	29	0.00237	0.03571	29	0.00003	0.006367	29
2	30	0.00012	0.00272	30	0.00027	0.00363	30	0.00003	0.005309	30
3	32	0.00544	0.02095	32	0.00741	0.03226	32	0.00007	0.008962	32
4	28	0.00031	0.00236	28	0.00169	0.01969	28	0.00003	0.008597	28
5	28	0.00595	0.02061	28	0.00899	0.02509	27	0.00000	0.007306	28
6	31	0.00270	0.02147	31	0.00618	0.02454	30	0.00002	0.002547	31
7	30	0.00012	0.00212	30	0.00094	0.01976	30	0.00002	0.008364	30
8	28	0.00025	0.00236	28	0.001210	0.0063	28	0.00003	0.006210	28
9	30	0.00014	0.00174	30	0.000580	0.00522	30	0.00003	0.003300	30
10	28	0.00076	0.00550	28	0.00767	0.03704	28	0.00000	0.003158	28
Avg		0.00161	0.00824		0.00373	0.02092		0.00003	0.00064	
Std. Dev		0.00229	0.00888		0.00341	0.01247		0.00002	0.00034	

Table 7: The number of solutions generated by each approach: 100-job, 10-machine instances.

	S-L-S		GA	
	Avg	**Std.Dev**	**Avg**	**Std. Dev**
100-job, 5-mc	0.03	0.012	49.9	9.803
100-job, 10-mc	0.059	0.019	68.3	13.342

Table 8: Computation time in seconds for the various approaches

where the jobs belong to one of two disjoint sets; the makespan criterion needs to minimized for one of the sets, while the total completion time needs to be minimized for the other. Our goal is to generate the set of non-dominated solutions. We extend some of the single machine structural insights of Baker and Smith [2003] to parallel machines, and develop an iterative SPT-LPT-SPT heuristic approach for this NP-hard problem. We also propose a bicriteria genetic algorithm to solve the problem. Both these heuristic approaches are compared with a time-indexed integer programming formulation for small and large sized instances. Results indicate that the heuristic approaches provide reasonable solution qualities while also being computationally efficient. This research also shows that exploiting problem structure in the different aspects of the genetic algorithm can produce good results. While the decomposition of jobs into different sets is possible only for makespan criteria and because of certain properties of the $P||\sum C_j$ problem, the broader idea of alternating job sets with regard to their *relative positions in their schedule* to control the generation of non-dominated points can be used for problems involving more criteria. The heuristic approaches proposed in this paper can also be adapted to other bicriteria pairs - in particular the $P|inter|(C_{max}, \sum w_j C_j)$, $P|inter|(C_{max}, L_{max})$, and $P|inter|(\sum C_j, L_{max})$ problems. Future research in the area would involve a further exploration of these extensions.

The GA proposed in this research can be also tested for various modifications and settings. For instance, Q, the pre-determined number of solutions for each k need not be a fixed value since it is clear from the results that the number varies with k. While we use a sufficiently high value of Q so as not to affect solution quality (in our case computation time is negligible as well) for future extensions and larger problem instances, more efficient implementations could be used.

The complexity of $Pm|inter|\epsilon(\sum C_j, C_{max})$ (i.e. when we assume that the number of machines is a fixed constant value and not a variable input) is also an interesting open question. Work is also needed to determine how the introduction of release dates affects the properties of the problem.

8 Appendix: Proof of Theorem 1

Theorem: *For every non-dominated point in the criteria space, there exists a schedule that can be divided into three sets: S_1 consisting of the k shortest jobs in $C2$ (i.e. the $\sum C_j$ jobs); S_2 consisting of jobs in $C1$ (i.e. the C_{max} jobs); and S_3 consisting of remaining $n_2 - k$ jobs in $C2$.*

Proof. Consider any non-dominated schedule $\mathbf{S}$. Let C^*_{max} be the makespan value of jobs in $C1$. We next define a vector $\mathbf{A}' = (A'_1, A'_2, ..., A'_m)$ of completion times on the machines; the purpose of $\mathbf{A}'$, as we shall see, is to help determine the decomposition described in the theorem into the three different sets. For each machine j that has no makespan jobs scheduled on it, A'_j is the latest possible completion time such that $A'_j \leq C^*_{max}$. On machines that have at least one makespan job scheduled on it, A'_j is simply the latest finish time of the jobs in $C1$ assigned to that machine. Thus, the highest element of $\mathbf{A}'$ will have a value of C^*_{max}.

We now demonstrate how $\mathbf{S}$ can be divided into sets S_1, S_2 and S_3. All jobs in $C2$ with completion times less (greater) than or equal to the A'_j value corresponding to the machine they are scheduled on belong to S_1 (S_3). Note that in any schedule, either S_1 or S_3 can be a null set, but not both. S_2 is simply the set of all jobs in $C1$.

We next show that there exists a schedule such that S_1 consists of the k shortest jobs in $C2$. We show only for the cases where S_1 and S_3 are not null - because if either S_1 or S_3 are null, $k = 0$ or $k = n$, and the concept of k shortest jobs is not applicable.

Before we present our proof, the following observations are in order:
Observation 1: The vector $\mathbf{A}^{'}$ represents the finish times on the machines after jobs in S_1 and S_2 have been scheduled - it is a partial schedule $\mathbf{S}_{12}$ of jobs. Given the finish time vector $\mathbf{A}^{'}$, we know that the jobs in S_3 are scheduled in SPT order, with the next shortest job assigned to the machine with the earliest finish time (Sanlaville and Schmidt [1998]). Note that at this stage the problem is identical to minimizing total completion time when machines have different availability times.
Observation 2: Let l be the machine in the partial schedule $\mathbf{S}_{12}$ with the lowest finish time. The finish time on machine l is thus $A_l^{'}$. Let z be the shortest job in S_3. We note that owing to the SPT ordering of jobs in S_3, z has a start time of $A_l^{'}$. We also note that by the definition of S_1 and S_3 and because $\mathbf{S}$ is non-dominated, $A_l^{'} + p_z > C_{max}^{*}$. Because of this property and because of the SPT ordering, the number of jobs in S_3 that are assigned to the machine l can never be 2 more than the number of S_3 jobs assigned any of the other machines. As an extension of this it is also true that on *any* machine the number of S_3 jobs assigned is at most one more than the number assigned to other machines.

We proceed with our proof. Suppose that there exists a job b in S_1 such that it is longer than at least 1 job in S_3. Let a be the largest job in S_3 such that $p_a < p_b$. Exchange the two jobs a and b. Since the sets are altered by the exchange, we now refer to them as $S_1^{'}$ and $S_3^{'}$. The following statements are true after exchange:
1) In the vector of finish times exactly *one* element will reduce by an amount $\delta_p = p_b - p_a$. In other words the finish time of exactly one machine will reduce by δ_p in the partial schedule $\mathbf{S}_{12}$.
2) The number of $S_3^{'}$ jobs assigned to each machine is the same as the number of S_3 jobs that were assigned to each machine before the exchange. Indeed the job assignments and positions on the machines remain identical except that job b has now replaced job a.

Let the new schedule after the exchange be called $\mathbf{S}^{'}$. Consider now the change in $\sum C_j$ and C_{max} after the exchange. It is easily seen that $C_a + C_b$ before and after the exchange remain the same, while the $\sum C_j$ of jobs in $S_1^{'} \setminus b$ remains the same or decreases. The C_{max} of the S_3 jobs will decrease at most by δ_p or remain the same. Next, we analyze the change in the objective function of the jobs in $S_3^{'}$, excluding job b. The two possible cases are:

1. *Case I* - The machine whose finish time in the partial schedule reduces by δ_p is also the same machine on which job b is scheduled after the exchange (or the case where both job a and b in are scheduled on the same machine in schedule $\mathbf{S}$): Let us refer to this machine as h. Each job in $S_3^{'}$ that is in an earlier position than b on machine h will improve its completion time by δ_p. On the other hand, the completion time of each job in $S_3^{'}$ that in is a later position than b on machine h will remain unaltered. This is because the the decrease in δ_p of machine finish time is offset by the increase in δ_p caused by the insertion of job b. Therefore in the worst case (which happens when b is the first position on machine h), the $\sum C_j$ of jobs in $S_3^{'} \setminus b$ remains the same. In all other cases, it improves.

1. *Case II* - The machine whose finish time in the partial schedule reduces by δ_p is different from the machine on which job b is scheduled after the exchange (or the case where jobs a and b are scheduled on different machines in $\mathbf{S}$): Let us refer to the first machine as h and the second machine as q. Let the number of $S_3^{'}$ jobs scheduled on q be x. Therefore at most $x-1$ jobs will have their completion times increased by δ_p. But from Observation 2, we also know that on h there will be at least $x-1$ jobs from $S_3^{'}$ assigned to it. Moreover, since the finish time of h (in the partial schedule) was reduced by δ_p, there will be at least $x-1$ jobs whose completion times reduce by δ_p. Thus in this case as well,the $\sum C_j$ of jobs in $S_3^{'} \setminus b$ remains the same or reduces.

From the above it is clear that after the exchange the $\sum C_j$ of the overall schedule either decreases or remains the same, and the same holds true for C_{max}. The schedule $\mathbf{S}'$ is still non-dominated. Thus we can continue to do such exchanges until there is no job in S_1 that is longer than the jobs in S_3, while still keeping the schedule non-dominated. This proves our claim.

(As an aside, we note here that the original schedule $\mathbf{S}$ can also be non-dominated. This indicates that the there can exist a non-dominated schedule that does not follow the properties described in the theorem. Our proof, however, shows that for every such schedule we can construct a different schedule with the same $\sum C_max$ and $\sum C_j$ values that does follow these properties.)

□

References

A Agnetis, P. Mirchandani, D. Pacciarelli, and A. Pacifici. Nondominated schedules for a job shop with two competing users. *Compt. Math. Organ. Theory*, 6:191–217, 2003.

A Agnetis, P. Mirchandani, D. Pacciarelli, and A. Pacifici. Scheduling problems with two competing agents. *Operations Research*, 52:229–242, 2004.

A. Agnetis, D. Pacciarelli, and A. Pacifici. Multi-agent single machine scheduling. *Annals of Operations Research*, 150(1):3–15, 2007.

K. Baker and J.C. Smith. A multiple criterion model for machine scheduling. *Journal of Scheduling*, 6:7–16, 2003.

A. Brindle. *Genetic algorithms for function optimization.* PhD thesis, University of Alberta, Edmonton, Department of Computer Science, 1981.

P. Brucker and S. Knust. Complexity results for scheduling problems, url http://www.mathematik.uni-osnabrueck.de/research/or/class/, 2006.

M Carlyle, J. Fowler, E. Gel, and B. Kim. Quantitative comparison of approximate solution sets for bicriteria optimization problems. *Decision Sciences*, 34(1), 2003.

T.C.E. Cheng, C.T. Ng, and J.J. Yuan. A note on the complexity of the two-agent scheduling problem on a single machine. *Journal of Combinatorial Optimization*, 12:387–394, 2006a.

T.C.E. Cheng, C.T. Ng, and J.J. Yuan. Multi-agent scheduling on a single machine to minimize total weighted number of tardy jobs. *Theoretical Computer Science*, 12:273–281, 2006b.

R. Conway, W. Maxwell, and L. Miller. *Theory of Scheduling.* Addison-Wesley, Reading, Massachussetts, 1967.

P. Cyzak and A. Jaszkiewicz. Pareto simulated annealing - a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multicriteria Decision Analysis*, 7:34–47, 1998.

K. Deb, S Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multiobjective optimization: NSGA II. In *Parallel Problem Solving from Nature – PPSN VI*, pages 849–858. Springer, 2000.

D Goldberg. *Genetic algorithms in search, optimization and machine learning.* Kluwer Academic Publishers, Boston, MA, 1989.

D Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: motivation, analysis and first results. *Complex Systems*, 3:493–530, 1990.

R. Graham, E. Lawler, J. Lenstra, and A. Rinnooy Kan. Optimization and approximation of deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

R.L Graham. Bounds for certain multiprocessor anomalies. *Bell System Technical Journal*, 17: 1563–1581, 1966.

H. Hoogeveen. Multicriteria scheduling. *European Journal of Operational Research*, 167(3):592–623, 2005.

J. Landa Silva and E. Burke. A tutorial on multiobjective metaheuristics for scheduling and timetabling. In X. Gandibleux, Sevaux. M., K. Sorenson, and V. T'Kindt, editors, *Multiple Objective Metaheuristics*. Springer, 2004a.

J. Landa Silva and E. Burke. Using diversity to guide the search in multi-objective optimization. In C. Coello Coello and G. Lamont, editors, *Applications of Multi-Objective Evolutionary Algorithms*. World Scientific, 2004b.

C. Lee. Parallel machines scheduling with nonsimultaneous machine available time. *Discrete Applied Mathematics*, 20:53–61, 1991.

T. Loukil, J. Teghem, and D. Tuyttens. Solving multi-objective production scheduling problems using metaheurisitics. *European Journal of Operational Research*, 161:42–61, 2005.

T. Pasupathy, C. Rajendran, and R. Suresh. A multi-objective genetic algorithm for scheduling flow shops to minimize the makespan and total flow time of jobs. *International Journal of Advanced Manufacturing Technology*, 27:804–815, 2006.

J. Peha. Heterogeneous-criteria scheduling: minimizing weighted number of tardy jobs and weighted completion time. *Journal of Computers and Operations Research*, 22(10):1089–1100, 1995.

E. Sanlaville and G. Schmidt. Machine scheduling with availability constraints. *Acta Informatica*, 35(9):795–811, 1998.

J. Sousa and L. Wolsey. A time-indexed integer programming formulation for non-preemptive single machine scheduling problems. *Mathematical Programming*, 54A(3):353–367, 1992.

V. T'Kindt and J. Billaut. *Multicriteria Scheduling.* Springer, 1 edition, 2002.