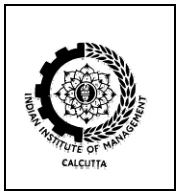

INDIAN INSTITUTE OF MANAGEMENT CALCUTTA

WORKING PAPER SERIES

WPS No. 686/ November 2011

Application of Graph Search and Genetic Algorithms for the Single Machine Scheduling Problem with Sequence-Dependent Setup Times and Quadratic Penalty Function of Completion Times

by

Viswanathan Kodaganallur
Associate Professor, Seton Hall University, 400 South Orange Avenue, NJ 07079, USA

Anup K. Sen
Professor, IIM Calcutta, Diamond Harbour Road, Joka P.O., Kolkata 700 104 India

&

Subrata Mitra
Professor, IIM Calcutta, Diamond Harbour Road, Joka P.O., Kolkata 700 104 India

Application of Graph Search and Genetic Algorithms for the Single Machine Scheduling Problem with Sequence-Dependent Setup Times and Quadratic Penalty Function of Completion Times

Viswanathan Kodaganallur

Seton Hall University

400 South Orange Avenue

NJ 07079, USA

E-mail: viswa.viswanathan@shu.edu

Anup K Sen

E-mail: sen@iimcal.ac.in

Subrata Mitra[1]

E-mail: subrata@iimcal.ac.in

Indian Institute of Management Calcutta

D. H. Road, Joka, Kolkata 700104, India

[1] Corresponding author

Application of Graph Search and Genetic Algorithms for the Single Machine Scheduling Problem with Sequence-Dependent Setup Times and Quadratic Penalty Function of Completion Times

Abstract

In this paper, we consider the single machine scheduling problem with quadratic penalties and sequence-dependent (QPSD) setup times. QPSD is known to be NP-Hard. Only a few exact approaches, and to the best of our knowledge, no approximate approaches, have been reported in the literature so far. This paper discusses exact and approximate approaches for solving the problem, and presents empirical findings. We make use of a graph search algorithm, Memory-Based Depth-First Branch-and-Bound (MDFBB), and present an algorithm, QPSD_MDFBB that can optimally solve QPSD, and advances the state of the art for finding exact solutions. For finding approximate solutions to large problem instances, we make use of the idea of greedy stochastic search, and present a greedy stochastic algorithm, QPSD_GSA that provides moderately good solutions very rapidly even for large problems. The major contribution of the current paper is to apply QPSD_GSA to generate a subset of the starting solutions for a new genetic algorithm, QPSD_GEN, which is shown to provide near-optimal solutions very quickly. Owing to its polynomial running time, QPSD_GEN can be used for much larger instances than QPSD_MDFBB can handle. Experimental results have been provided to demonstrate the performances of these algorithms.

Keywords: Single machine scheduling; Sequence-dependent setup; Quadratic penalty; Graph search; Genetic algorithm

Application of Graph Search and Genetic Algorithms for the Single Machine Scheduling Problem with Sequence-Dependent Setup Times and Quadratic Penalty Function of Completion Times

1. Introduction

Single machine scheduling problems have been widely studied (Pinedo, 1995). One version of the problem that has been extensively dealt with in the literature is the consideration of sequence-dependent setup times (Choi and Choi, 2002; Gupta and Smith, 2006; Luo et al., 2006; Liao and Juan, 2007; Luo and Chu, 2007; Biskup and Herrmann, 2008; Koulamas and Kyparisis, 2008; Lin and Ying, 2008; Valente and Alves, 2008; Wang, 2008; Ang et al., 2009; Anghinolfi and Paolucci, 2009; Kim and Lee, 2009; Tasgetiren et al., 2009; Wang and Tang, 2010; Zhao and Tang, 2010; Nekoiemehr and Moslehi, 2011). Another version of the problem that has drawn limited attention is the consideration of quadratic penalty functions of job completion times (Townsend, 1978; Bagga and Kalra, 1980; Gupta and Sen, 1984; Bagchi et al., 1987a, 1987b; Szwarc et al., 1988; Sen et al., 1990; Croce et al., 1995; Mondal and Sen, 2000a). In this paper, we consider the single machine scheduling problem with quadratic penalties and sequence-dependent (QPSD) setup times. In QPSD, there are N jobs, J_i, i = 1..N, all of them available at time 0. These jobs are to be processed on a machine one after the other. Associated with J_i are the processing times, a_i, penalty coefficients, p_i, and setup times, $s_{i,j}$ (being the setup time for J_j when it is immediately preceded by J_i). The objective is to minimize the total penalty across all jobs, i.e. to minimize the weighted sum of the squares of the completion times. When the setup times are sequence-independent, they can simply be added to the processing times for the corresponding jobs and thus possess no additional complexity over problems without setup times. However, when the setup times are sequence-dependent, the quadratic penalty problem becomes extremely difficult to solve. It is interesting to note that minimizing the weighted sum of the completion times (i.e. the linear penalty case) can be transformed and treated as if the setup times are sequence-independent (Sen and Bagchi, 1996). However, no such transformation is possible for the non-linear penalty case with sequence-dependent setup times. In this paper, we attempt to advance the state of the art for solving the QPSD problem.

Single machine scheduling problems with N jobs have N! possible distinct sequences, and except for special cases, it is known that these problems are NP-Hard (Rinooy Kan, 1976), i.e. finding an optimal solution requires an implicit enumeration of all possible sequences. QPSD has some similarities to the asymmetric travelling salesman problem (ATSP) (Choi et al., 2003). However, unlike ATSP, rotations of permutations are not equivalent in QPSD, and the processing sequence affects the job completion times. It is, therefore, a more complex problem, and finding an optimal solution requires exploring a larger search space. GREC (Sen and Bagchi, 1996) is a general graph search algorithm that has been used to solve QPSD optimally. Although GREC has been shown to be faster than Depth-First Branch-and-Bound (DFBB) for this problem, it runs out of memory even for moderate-sized problems. To our knowledge, approximate approaches to solve large instances of QPSD have not been reported in the literature so far. In this paper, we present exact and approximate approaches for QPSD. In particular:

- We discuss the characteristics of QPSD. The presence of sequence-dependent setup times makes conventional graph-search algorithms like A^* (Hart et al., 1968) and dynamic programming approaches (French, 1982) inapplicable.

- We describe QPSD_MDFBB, an application to QPSD of a Memory-Based Depth-First Branch-and-Bound (MDFBB) approach proposed by Mondal and Sen (2001). QPSD_MDFBB optimally solves QPSD instances up to 30 jobs.

- In order to propose approximate approaches for solving large instances, we first present a greedy approach for QPSD and describe the concept of greedy stochastic search (Viswanathan and Sen, 2010). We present an effective greedy stochastic algorithm, QPSD_GSA that can generate moderately good solutions rapidly even for large instances. The general idea of GSA is applicable to a wide range of combinatorial optimization problems like the travelling salesman, knapsack, combinatorial auction and other problems for which greedy solutions can be conceived.

- The major contribution of this paper is to combine the findings of QPSD_GSA with a new genetic algorithm formulation called QPSD_GEN for the problem. QPSD_GEN does not

guarantee optimal solutions, but has been seen to generate near-optimal solutions in general. Our findings indicate that QPSD_GEN can be used to generate solutions within 2.2% of the optimal solutions for 100-job problem instances. We have used QPSD_GSA to generate a subset of the initial population for the QPSD_GEN, and found this to have a great impact on the quality of the solutions obtained. This way of integrating GSA into genetic algorithms is also more generally applicable.

The paper is organized as follows. Section 2 discusses the problem with special emphasis on the complexities introduced by sequence-dependent setup times. The algorithm QPSD_MDFBB and experimental results for it are presented in Section 3. Section 4 presents QPSD_GSA and experimental results for it. The implementation of the genetic algorithm, QPSD_GEN and the associated experimental results are given in Section 5. Section 6 concludes the paper and suggests areas for further work.

2. Effect of sequence-dependent setup times on QPSD

The QPSD problem may be formulated in IP (Integer Programming). However, such a formulation may not be efficient to solve in practice. Sen and Bagchi (1996) have shown that the search space for job sequencing problems can be modelled as a tree, or as a graph, and algorithms using the graph search space run faster. For the QPSD problem under the tree formulation, two nodes with the same set of jobs but in different orders and having the same last job will generally not have the same cost because the setup times for the jobs could differ. Nevertheless, the sub-trees below them are identical in terms of the structure. Algorithms using the tree search space cannot take advantage of this fact and might wastefully traverse these identical sub-trees more than once. The graph search space has far fewer nodes and offers the potential for faster search. The node count reduction results from the fact that unlike in the tree search space, there could be multiple paths from the root node to any given node, and this helps avoid replicating the identical sub-trees. However, sequence-dependent setup times complicate traditional graph search because the identical sub-trees may not have the same costs.

The main feature of graph search algorithms like the graph version of A^* (Hart et al, 1968) is that when these reach the same node through different paths, they retain the path having the lowest

cost, discarding any other paths from the root to the node. This approach works fine when the incremental cost from a given node to a goal node is independent of the path by which the node was reached. This is the same as the principle of optimality on which the dynamic programming formulations are based (French, 1982). However, this does not hold for sequence-dependent setup times. For example, consider the following 4-job problem given in Table 1 (Viswanathan and Sen, 2010).

Insert Table 1 about here

In this example, it is assumed that the setup time for a job is zero if it is the first in the sequence. Consider the ordered sequence of jobs (1, 2, 3) and (2, 1, 3). Under the graph formulation, a node is represented by the set of completed jobs without regard to the ordering, except for the last job in the sequence. Because the set of jobs and the last job in the two ordered sequences in question are the same, the two are represented by a single node $(\{1,2\},3)$, where the first two jobs form a set (unordered) and the last job is shown separately. The cost when the node is reached through the sequence 1, 2, 3 is 182 and through the sequence 2, 1, 3 is 188. If a traditional graph search algorithm reaches the node through the two different paths considered, it would simply discard the higher cost path 2, 1, 3. However, if we look below this node, we see that the sequence 1, 2, 3, 4 has a cost of 1206, which is higher than the cost of the sequence 2, 1, 3, 4 which is 1088. A traditional graph search algorithm thus runs the risk of missing the optimal solution.

Mondal and Sen (2001) have proposed the MDFBB search algorithm which is a memory constrained graph search algorithm. MDFBB is unaffected by situations when the costs of paths from a node to a goal node depend on the path through which the node was reached, as occurs in the presence of sequence-dependent setup times. It has been proven to provide optimal solutions and can therefore be applied to QPSD to take advantage of the graph formulation and yet guarantee optimal solutions. The next section describes the QPSD_MDFBB algorithm. The algorithm works on the graph formulation.

3. QPSD_MDFBB algorithm and empirical results

Algorithm QPSD_MDFBB is shown in Figure 1. It uses the recursive function QPSD_REC. Unlike DFBB which stores only nodes on the current path and does not exploit the available additional memory on today's computers, QPSD_MDFBB uses a fixed predetermined amount of memory to store nodes. As in DFBB, QPSD_MDFBB starts with an upper bound on the cost of the optimal solution and examines all the paths from the start to the goal node in a depth-first manner. Whenever a solution is found whose cost is lower than the best one found so far, the upper bound is revised to the cost of this new solution. Whenever a path is found whose cost can be guaranteed to equal or exceed the upper bound, it is pruned. When an already generated node is encountered along a different path, QPSD_MDFBB can use the stored cost of the node to decide whether it should be searched again.

Insert Fig. 1 about here

When memory becomes full, QPSD_MDFBB attempts to replace the less promising nodes, i.e. the nodes which have been visited relatively less recently, with the newly generated promising ones. QPSD _MDFBB maintains as much of the graph structure as memory will permit by storing generated nodes.

A node has an associated cost denoted by its *b*-value. The *b*-value $b(n)$ of a node n is the cost of the currently known least cost path from n to a goal node. When n is generated for the first time, *b(n)* is set to the heuristic estimate *h(n)* of the corresponding sub problem (calculated using Townsend's heuristic (Townsend, 1978), but with the effective processing times as shown in Figure 1). When the successors of node n are generated or visited, the *b(n)* is updated by a bottom-up cost revision procedure based on the *b*-values of the successors.

The value of $b(n)$ is reset to the minimum of $c(n, n_i) + b(n_i)$, where the minimum is taken over all the immediate successors of n, and $c(n, n_i)$ is the cost of the arc between nodes n and n_i. As a result, as more and more nodes below a node n are searched, $b(n)$ increases from *h(n)* to attain

the optimal cost below the node, while always remaining a valid lower bound. The use of $b(n)$ helps in avoiding repetitive search below n and also increases the pruning power of the algorithm. Since algorithm QPSD_MDFBB is an adaptation of MDFBB, its optimality follows from the optimality of MDFBB.

The difficult part in the algorithm is the computation of the b-value of a node and storing it, knowing that it is path-dependent. The b-value at a node is stored using path-dependent and path-independent components, and when a node is reached again along a different path, its path dependent component and hence its b-value is recomputed. What this enables us to do even though arc costs are path dependent, is to reuse the b-value components when a node is revisited by simply modifying these values appropriately. Thus the effort expended in earlier explorations under the node is not wasted and need not be repeated.

The b-value at a node n can be expressed in the form (Sen and Bagchi, 1996):

$$b(n) = \alpha T^2 + \beta T + \gamma$$

where α, ß and γ are parameters that depend on node n but not on T (the completion time of the last job in n). For any job J_k in the set of Q jobs remaining to be processed at node n, the finish time t_k can be expressed as $T + (t_k - T)$; the coefficients of T^2 can be grouped together yielding the parameter α, the coefficients of T yield the parameter ß, and the remaining terms combine to give γ. The value of α remains constant during the search, ß and γ change as more and more jobs in Q get sequenced; α, ß and T must be stored at each node and reset from bottom and also when reached along a different path. This method can be extended to penalty functions that are higher powers of the job finish times.

Table 2 shows the experimental results for QPSD_MDFBB under a 512K node limit. The results were obtained on an Intel Pentium 1.79 GHz PC running Windows XP. It can be seen that QPSD_MDBB can readily solve 30-job problems. Previous experiments (Sen and Bagchi, 1996; Mondal and Sen, 2000b) have reported solutions for only up to 22-job problems. Thus QPSD_MDFBB is able to solve larger instances in reasonable time and with a relatively low memory limit. Using a higher node limit, QPSD_MDFBB will be able to solve much larger instances obviously at the cost of a considerable increase in running time.

Insert Table 2 about here

Can we solve larger instance of QPSD quickly? We suggest approximate approaches in the next two sections.

4. Greedy Stochastic Approach for QPSD

Greedy algorithms are known for many combinatorial optimization problems (Kruskal, 1956; Martello and Toth, 1990). As the name suggests, these algorithms aim to incrementally develop good solutions by taking what appears to be the best local action without making any effort to consider the global impact of these actions. For example, the nearest neighbour heuristic for the TSP (Johnson and McGeoch, 1997) creates a tour by starting at an arbitrary city and visiting the nearest unvisited city, and so on till the tour is completed. For the knapsack problem, the greedy approach is to order the items in non-increasing order of their value per unit weight, and to progressively fill the knapsack with as many units of the remaining highest valued item as will fit. Although it is well known that these approaches cannot guarantee optimal solutions, in practice they tend to give moderately good solutions.

Townsend (1978) presented sufficient conditions for optimality for sequencing problems with quadratic penalties and no setup times (QP). A modified version of this can be used to develop a greedy solution for QPSD. We first describe Townsend's sufficient conditions for QP.

As already mentioned, let a_i be the processing time for J_i and p_i be its penalty coefficient. Suppose there is a sequence S in which the jobs are ordered as

$J_{k(1)}, J_{k(2)}, .. J_{k(N)}, k(i) \in \{1..N\}, \; k(i) \neq k(j) \;\; if \; i \neq j$

Townsend showed that the sequence is optimal if it satisfies both of the following conditions:

$$\frac{p_{k(1)}}{a_{k(1)}} \geq \frac{p_{k(2)}}{a_{k(2)}} \geq .... \frac{p_{k(N)}}{a_{k(N)}} \qquad ...(1)$$

$$p_{k(1)} \geq p_{k(2)} \geq \ldots . p_{k(N)} \qquad \ldots(2)$$

Of course, the problem data might be such that both of these cannot be simultaneously satisfied, and therefore an implicit enumeration might be needed to obtain an optimal solution for QP. Scheduling the jobs solely by their ratios as in (1) above can be seen as a greedy approach. Viswanathan and Sen (2010) have shown that this approach is extremely effective for randomly generated QP instances. In their paper, a greedy approach for QPSD based on Townsend's sufficient conditions has been proposed. The greedy algorithm is shown in Figure 2. The algorithm proceeds by selecting the jobs to be scheduled one by one. At each stage the unscheduled job with the highest Townsend ratio is selected, where the ratio is calculated using the effective processing time based on the preceding job. This might not result in the jobs being ordered in decreasing order of the ratio, but at each stage we are taking the best local action.

Insert Fig. 2 about here

The experimental results for QPSD_GREEDY on randomly generated problems are reproduced in Table 3.

Insert Table 3 about here

It is not as effective as the direct application to QP, because QPSD is much more complex. However, since the approach is based on only one of the two conditions, and since the condition is sufficient, but not necessary, it seems likely that perturbing the approach and generating multiple solutions and taking the best one could lead to improvements in the solution quality. One way of perturbing the greedy approach is as follows. Rather than deterministically following Townsend's ordering, Viswanathan and Sen (2010) have suggested a probabilistic approach. In

this approach, while selecting the next job to be scheduled, the jobs are ordered as above by their Townsend ratio values, but instead of deterministically selecting the job with the highest ratio, the next job is selected based on a probabilistic model. This approach may be called the greedy stochastic approach (GSA), and its application to QPSD as the algorithm QPSD_GSA, which is presented in Figure 3.

Insert Fig. 3 about here

The performance of QPSD_GSA depends crucially on the specific discrete probability distribution used. The main issue is that of how far to deviate from a straight allocation based strictly on the Townsend ratio. We experimented with uniform, geometric and binomial distributions to reflect a complete random selection, selection based on an exponentially declining model and a bell shaped model, respectively. At any stage when there are, say m unscheduled jobs, the random choice process requires the selection of a number between 1 and m, both inclusive. Under the uniform distribution, we picked any of the unscheduled jobs with equal probability. (For this, reordering the jobs by their Townsend ratios is not necessary).

The geometric distribution, (shown in Figure 4, and suitably translated to the right by 1 to avoid choosing zero) appears intuitively to be ideal for this problem since it has a high probability of selecting the job with the highest ratio, with decreasing, but non-zero probabilities for lower ratios. Since the geometric distribution provides a finite probability of picking integers greater than m, whereas we want only values between 1 and m, we adjusted the probabilities by redistributing the cumulative probability for values greater than m proportionately to the values in the range 1 to m. The use of the binomial distribution (which has a bell shape) is straightforward, except that it too needs to be shifted to the right by 1 to avoid picking 0.

The experimental results of using each of these are reproduced from Viswanathan and Sen (2010) and shown in Table 4. For each problem instance, we generated 256 random greedy stochastic solutions and took the best one. We chose the parameters for the geometric and

binomial distributions by trial and error and retained the values that yielded the best results. For the geometric and binomial distributions studied, the parameter values of 0.85 and 0.025, respectively, gave the best results with binomial being the better of the two. Interestingly, the graph for the binomial distribution with this parameter looks very much like the geometric distribution. The left half of the bell disappears. This graph is shown in Figure 5.

Insert Table 4 about here

Insert Fig. 4 and 5 about here

The binomial distribution consistently outperformed the geometric distribution. The geometric distribution with parameter values close to 1 tends to give too much weight to the higher ranked jobs and too little to the rest, whereas, the binomial distribution with a low parameter value is somewhat less lopsided and appears to provide better stochastic perturbation.

From the results given in Table 4, it is clear that the stochastic version of the greedy algorithm provides much better solutions than the deterministic version for QPSD. Nevertheless, the solutions are not sufficiently close to optimal for standalone use. In the next section we show how this approach can be effectively integrated into a proposed genetic algorithm implementation.

5. Genetic Algorithm – QPSD_GEN

Genetic Algorithms (GA) make use of methods developed by Holland (1975) and popularized by Goldberg (1989) to find good solutions to complex combinatorial optimization problems. The approach is motivated by the biological process of natural selection. Genetic algorithms as well

as other evolutionary approaches like simulated annealing and tabu search (Reeves, 1993) have been applied to scheduling and sequencing problems earlier (Mattfeld, 1996) with success. There is continuing work in applying GA for scheduling. Lee and Choi (1995) reported a GA application to the problem with due dates and earliness-tardiness penalties. Chang et al. (2006) reported a novel case-injected GA for single machine sequencing with release times. Tiwari and Vidyarthi (2000) described a GA for machine loading in flexible manufacturing systems. Norman and Bean (1999) proposed a random keys-based GA application for complex scheduling problems. Sadegheih (2006) described an application of GA to sequence job operations in machine shops with precedence constraints. Dowsland and Thompson (2006) reported an application of ant colony optimization to the examination scheduling problem. A few other relevant references on the application of GA to machine scheduling problems with sequence-dependent setups are Vallada and Ruiz (2011), Vela et al. (2010), Tavakkoli-Moghaddam et al. (2009), Ruiz and Maroto (2006), and Cheung and Zhao (2001). Interestingly, QPSD has not yet been addressed using such techniques. Since the problem is very complex, the size of problems that can be solved optimally is drastically limited. In an effort to make progress on large problem instances, we applied GA to QPSD and have obtained very encouraging results. Although the optimality of the solutions cannot be guaranteed, our findings indicate that near-optimal solutions can be obtained very quickly for even large problem instances. Our earlier discussion has revealed that optimal search algorithms for the problem either run out of memory (GREC) or take too long to run to completion (DFBB and QPSD_MDFBB with values of $N > 30$).

The salient aspects of our GA implementation are:

- Use of the greedy stochastic algorithm to generate a portion of the initial population
- Use of a modified version of the single point crossover for mating
- Use of an auxiliary operation based on neighbour exchanges after every fixed number of generations to improve the fitness of the solution pool

5.1 Chromosome representation

A permutation of 1 to N, N being the number of jobs, is a natural chromosomal representation for QPSD since the set of all permutations is the same as the set of all feasible schedules. Other schemes have been adopted for scheduling and related problems. For instance, in solving the

asymmetric travelling salesman problem, Choi et al. (2003) deliberately expanded the search space to consider infeasible solutions as well. They therefore used an adjacency representation that accommodated sub-tours. Lee and Choi (1995) used the permutation notation, augmented with minus signs to represent blocking information for the single machine sequencing problem with early-tardy penalty weights. We decided to restrict the search to just the feasible region and hence adopted a simple permutation representation for the chromosomes. The fitness value of a permutation is simply the cost of the associated schedule.

5.2 Initial population

It is common practice in genetic algorithms to start with a randomly generated initial population. We introduced a novelty in our experiments; our initial population is comprised of three distinct classes of schedules. The first (and largest) class consists of randomly generated schedules. The second class is made up of solutions generated using the greedy stochastic approach QPSD_GSA described in the previous section using the binomial distribution and the third class is generated using QPSD_GSA with the geometric distribution. This provided a good deal of diversity in the population.

5.3 Mating and fitness scaling

To minimize the effects of good fitness values dominating the solution pool early on in the process and reducing diversity, we chose to use a rank-weighted mating strategy (Goldberg, 1989). Under this, the chromosomes are first ranked in increasing order of their fitness values. The probability of chromosome with rank i being selected for mating is $\frac{2(N-i+1)}{N(N+1)}$. This provides chromosomes with poor fitness values a higher probability of being selected as parents than would have been possible if fitness value alone were used to determine parent selection.

5.4 Crossover

Several mating schemes have been considered for permutation representations. Prominent among these are partially matched crossover (PMX), cycle crossover (CX) and tie-break crossover (TBX). In addition, we also considered a new single point crossover operation adapted from PMX. Under this, a random crossover point is chosen and the portion to the left of this point is

first exchanged between the mating chromosomes. This process generally results in infeasible schedules (with duplicate and missing values) and an additional repair step is needed. The repair process is exactly analogous to the repair process in PMX. An example is shown in Figure 6 to illustrate the process with N = 6 (the crossover point is shown with a solid triangle in each chromosome). In the example, the first child gets the values 1, 3 and 5, and loses the values 3, 4 and 2. As a result of this, it has duplicate values for 1 and 5, and is missing the values 2 and 4. We leave the portion to the left of the crossover point intact and substitute 1 with 4 and 5 with 2 to the right of the crossover point. Child 2 is also repaired analogously. We found that the single point crossover and PMX were the best performers, with the former being marginally better.

Insert Fig. 6 about here

5.5 Elitism

At the end of each generation, the population doubles because each mating of two chromosomes produces two new offspring. Retaining only the best values from the resultant set tends to reduce the diversity and causes the algorithm to converge quickly to a local optimum. In order to retain diversity, it is common to use a process of elitism. Under this, the extended population comprising the population at the start of a generation along with the newly generated offspring is ranked and then divided into buckets, for example 40%, 40% and 20% based on the ranks. From each bucket, a specified proportion of the final population for the next generation (say 60%, 30% and 10%) is picked. This allows a certain percentage of chromosomes with poor fitness values to be included in the population so that there is a better chance of breaking out of local minima.

5.6 Mutation

We tested out several mutation options. Unlike in a binary string representation for chromosomes, where a bit is simply flipped if chosen for mutation, in a permutation representation there is the additional problem of maintaining feasibility. We tried several different mutation operators. In the 2-opt operator (Lin and Kernighan, 1971), when an element

is chosen for mutation, it is swapped with another randomly chosen element. In the neighbour swap mutation, when an element is chosen for mutation, it is swapped with its neighbour to the right (with wrap-around). In the three-point-swap operation, when an element is chosen for mutation, two others are also randomly chosen and the three are cyclically swapped. Finally, in the random-two-swap operation, the choice for mutation is done at the chromosome level rather than at the individual element. Here if a chromosome is chosen for mutation, two of its elements are randomly selected and swapped.

5.7 Auxiliary operations

Akin to the repair process adopted in Choi et al. (2003) for the ATSP, once every fixed number of generations, we carried out an enhancement operation where we took each chromosome in the population and applied all possible neighbour exchanges that increased the fitness.

5.8 Experimental results

We tested QPSD_GEN on randomly generated problems. We generated problems having between 10 and 30 jobs, and for each value we generated 100 problems with processing times varying uniformly between 1 and 100 and setup times and penalty coefficients as integers ranging uniformly between 1 and 10. We used a population size of 256. The initial population comprised 50% random solutions, 25% generated using QPSD_GSA with binomial distribution with $p = 0.025$ and 25% generated with a geometric distribution with $p = 0.85$. We ran the algorithm for 100 generations for all problems. We found that the single point crossover operator and neighbour mutation performed best. We used a fixed mutation probability of 0.03. For elitism, we retained the top 5% of the best solutions from the extended population after mating. We divided the rest of the solutions into three buckets of 40%, 30% and 25% and from these buckets we randomly picked 62.5%, 50% and 20%, respectively, at random to survive into the next generation. We used the auxiliary operation described above once every 4 generations, as well as at the end of the final generation. The results are shown in Table 5. As expected, the deviation from optimal for QPSD_GEN grows with the number of jobs, but this growth is not alarming. It is interesting to note that these results have been obtained using very conservative GA parameters (100 generations, population size 256). For larger instances these can be increased – it is very common for GA's to be run for over 200 generations and also to use larger

population sizes. Therefore, good solutions can be expected for even much larger instances. Figure 7 compares the percentage deviation from optimal for QPSD_GEN with those for the deterministic and stochastic greedy approaches. This underscores the robustness of QPSD_GEN since its solutions do not deteriorate as rapidly as those of the greedy solutions with increasing problem size.

Insert Table 5 about here

Insert Fig. 7 about here

To get an idea of how QPSD_GEN will perform beyond our experimental range, we ran a regression of the problem size with the percentage deviation. The R^2 value was 0.921 and the regression model was significant at 99%. Extrapolating based on the regression results, we can expect a 2.19% deviation from optimal for 100 job problems, provided the regression model is valid in that range. To solve larger instances by QPSD_GEN and to avoid numeric overflow, we used 64-bit 1.6 GHz Intel Itanium processor running Linux and the algorithm took around 47, 61 and 77 seconds for solving 80, 90 and 100 job instances respectively.

6. Conclusions and scope for further research

We have presented exact and approximate approaches to the QPSD problem with a view to advancing the state of the art. We have outlined the complexities that sequence-dependent setup times cause for traditional graph search algorithms. We presented QPSD_MDFBB, an exact memory-constrained algorithm that advances the state of the art by optimally solving larger problem instances than is possible with exact approaches reported earlier. We also presented a new greedy solution for QPSD and suggested the idea of greedy stochastic algorithms (GSA) with an application to QPSD called QPSD_GSA which can find moderately good solutions very quickly and can hence be applied for extremely large instances. We integrated QPSD_GSA into

a new genetic algorithm implementation QPSD_GEN which was shown to provide near-optimal solutions very quickly. The deviation from optimal for QPSD_GEN appears to be linear and hence it is useful for getting very good approximate solutions for large problem instances.

It is clear that although QPSD_MDFBB does advance the state of the art, the size of problems that can be solved optimally is still quite small. The reason seems mainly to be the lack of tight heuristic estimating functions. This is one important area for further research. There is also scope for further work in the area of GSA as applied to QPSD and also more generally to other combinatorial optimization problems. Theoretical analyses as well as empirical studies of its applications to various problems should be interesting. It will also be interesting to compare the performance of QPSD_GEN with that of other evolutionary approaches to the problem including tabu search, simulated annealing and ant colony optimization.

References

Ang, A.T.H., Sivakumar, A.I., Qi, C., 2009. Criteria selection and analysis for single machine dynamic on-line scheduling with multiple objectives and sequence-dependent setups. Computers and Industrial Engineering. 56(4), 1223-1231.

Anghinolfi, D., Paolucci, M., 2009. A new discrete particle swarm optimization for the single-machine total weighted tardiness scheduling problem with sequence-dependent setup times. European Journal of Operational Research. 193(1), 73-85.

Bagchi, U., Sullivan, R.S., Chang, Y-L., 1987a. Minimizing mean squared deviation of completion times about a common due date. Management Science. 33(7), 894-906.

Bagchi, U., Chang, Y-L., Sullivan, R.S., 1987b. Minimizing absolute and squared deviations of completion times with different earliness and tardiness penalties and a common due date. Naval Research Logistics. 34(5), 739-751.

Bagga, P.C., Kalra, K.R., 1980. A node elimination procedure for Townsend's algorithm for solving the single machine quadratic penalty function scheduling problem. Management Science. 26(6), 633-636.

Balas, E., 1985. On the facial structure of scheduling polyhedra. Mathematical Programming. 24, 179-218.

Biskup, D., Herrmann, J., 2008. Single-machine scheduling against due dates with past-sequence-dependent setup times. European Journal of Operational Research. 191(2), 587-592.

Chang, P-C., Hsieh, J-C., Liu, C-H., 2006. A case-injected genetic algorithm for single machine scheduling problems with release time. International Journal of Production Economics. 103(2), 551-564.

Cheung, W., Zhou, H., 2001. Using genetic algorithms and heuristics for job shop scheduling with sequence-dependent setup times. Annals of Operations Research. 107(1), 65-81.

Choi, I-C., Choi, D-S., 2002. A local search algorithm for jobshop scheduling problems with alternative operations and sequence-dependent set-ups. Computers and Industrial Engineering. 42(1), 43-58.

Choi, I-C., Kim, S-I., Kim, H-S., 2003. A genetic algorithm with a mixed region search for the asymmetric traveling salesman problem. Computers and Operations Research. 30(5), 773-786.

Croce, F.D., Szwarc, W., Tadei, R., Baracco, P., di Tullio, R., 1995. Minimizing the weighted sum of quadratic completion times on a single machine. Naval Research Logistics. 42(8), 1263-1270.

Dowsland, K.A., Thompson, J.M., 2005. Ant colony optimization for the examination scheduling problem. Journal of the Operational Research Society. 56(4), 426-438.

French, S., 1982. Sequencing and scheduling: An introduction to the mathematics of job shop. Ellis Horwood Ltd., Reading, Chichester.

Goldberg, D.E., 1989. Genetic algorithms in search, optimization and machine learning. Addison Wesley, Reading, MA.

Gupta, S.K., Sen, T., 1984. On the single machine scheduling problem with quadratic penalty function of completion times: An improved branching procedure. Management Science. 30(5), 644-647.

Gupta, S.R., Smith, J.R., 2006. Algorithms for single machine total tardiness scheduling with sequence-dependent setups. European Journal of Operational Research. 175(2), 722-739.

Hart, P., Nilsson, N., Raphael, B., 1968. A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Syst. Science and Cybernetics. SSC4(2), 100-107.

Holland, J. H., 1975. Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor.

Johnson, D.S., McGeoch, L.A., 1997. The traveling salesman problem: A case study in local optimization, in: Aarts, E.H.L., Lenstra, J.K. (Eds.), Local search in combinatorial optimization. John Wiley and Sons, pp. 215-310.

Kim, J-G., Lee, D-H., 2009. Algorithms for common due-date assignment and sequencing on a single machine with sequence-dependent setup times. Journal of the Operational Research Society. 60(9), 1264-1272.

Koulamas, C., Kyparisis, G.J., 2008. Single-machine scheduling problems with past-sequence-dependent setup times. European Journal of Operational Research. 187(3), 1045-1049.

Kruskal, J., 1956. Greedy algorithm for the minimum spanning tree problem. Proceedings of the American Mathematical Society, pp. 48-50.

Lee, C.Y., Choi, J.Y., 1995. A genetic algorithm for job sequencing problems with distinct due dates and general early-tardy penalty weights. Computers and Operations Research. 22(8), 857-869.

Liao, C-J., Juan, H-C., 2007. An ant colony optimization for single-machine tardiness scheduling with sequence-dependent setups. Computers and Operations Research. 34(7), 1899-1909.

Lin, S., Kernighan, B., 1973. An effective heuristic algorithm for the travelling salesman problem. Operations Research. 21 (2), 498-516.

Lin, S.W., Ying, K.C., 2008. A hybrid approach for single-machine tardiness problems with sequence-dependent setup times. Journal of the Operational Research Society. 59(8), 1109-1119.

Luo, X., Chu, C., 2007. A branch-and-bound algorithm for the single machine schedule with sequence-dependent setup times for minimizing maximum tardiness. European Journal of Operational Research. 180(1), 68-81.

Luo, X., Chu, C., Wang, C., 2006. Some dominance properties for single-machine tardiness problems with sequence-dependent setup. International Journal of Production Research. 44(17), 3367-3378.

Martello, S., Toth, P., 1990. Knapsack problems: Algorithms and computer implementations. John Wiley & Sons.

Mattfeld, D.C., 1996. Evolutionary search and the job shop. Physica Verlag, Heidelberg.

Mondal, S.A, Sen, A.K., 2000a. An improved precedence rule for single machine sequencing problems with quadratic penalty. European Journal of Operational Research. 125(2), 425-428.

Mondal, S.A, Sen, A.K., 2000b. TCBB scheme: Applications to single machine sequencing problems. Proc AAAI-2000, pp. 792-797.

Mondal, S.A, Sen, A.K., 2001. Single machine weighted earliness-tardiness penalty problem with a common due date. Computers and Operations Research. 28(7), 649-669.

Nekoiemehr, N., Moslehi, G., 2011. Minimizing the sum of maximum earliness and maximum tardiness in the single-machine scheduling problem with sequence-dependent setup time. Journal of the Operational Research Society. 62(7), 1403-1412.

Norman, B.A., Bean, J.C., 1999. A genetic algorithm methodology for complex scheduling problems. Naval Research Logistics. 46(2), 199-211.

Pinedo, M. 1995. Scheduling: Theory, algorithms and systems. Prentice Hall.

Reeves C.R., 1993. Modern heuristic techniques for combinatorial problems. Orient Longman.

Rinooy Kan, A.H.G. 1976. Machine Complexity Problems: Classification Complexity and Computations. Nijhoff, The Hague.

Ruiz, R., Maroto, C., 2006. A genetic algorithm for hybrid flowshops with sequence-dependent setup times and machine eligibility. European Journal of Operational Research. 169(3), 781-800.

Sadegheih, A., 2006. Scheduling problem using genetic algorithm, simulated annealing and the effects of parameter values on GA performance. Applied Mathematical Modelling. 30, 147-154.

Sen, A.K., Bagchi, A., 1996. Graph search methods for non-order-preserving evaluation functions: Applications to job sequencing problems. Artificial Intelligence, 86(1), 43—73.

Sen, T. Dileepan, P., Ruparel, B., 1990. Minimizing a generalized quadratic penalty function of job completion times: An improved branch-and-bound approach. Engineering Costs and Production Economics. 18, 197-202.

Szwarc, W., Posner, M.E., Liu, J.J., 1988. The single machine scheduling problem with a quadratic cost function of completion times. Management Science. 34(2), 1480-1488.

Tasgetiren, M.F., Pan Q-K, Liang, Y-C., 2009. A discrete differential evolution algorithm for the single machine total weighted tardiness problem with sequence-dependent setup times. Computers and Operations Research. 36(6), 1900-1915.

Tavakkoli, R., Taheri, F., Bazzazi, M., Izadi, M., Sassani, F., 2009. Design of a genetic algorithm for bi-objective unrelated parallel machines scheduling with sequence-dependent setup times and precedence constraints. Computers & Operations Research. 36(12), 3224-3230.

Tiwari, M.K., Vidyarthi, N.K., 2000. Solving machine loading problems in a flexible manufacturing system using a genetic algorithm based heuristic approach. International Journal of Production Research. 38(14), 3357-3384.

Townsend, W., 1978. The single machine problem with quadratic penalty function of completion times: A branch-and-bound solution. Management Science. 24(5), 530-534.

Valente, J.M.S., Alves, R.A.F.S., 2008. Beam search algorithms for the single machine total weighted tardiness scheduling problem with sequence-dependent setups. Computers and Operations Research. 35(7), 2388-2405.

Vallada, E., Ruiz, R., 2011. A genetic algorithm for the unrelated parallel machine scheduling problem with sequence-dependent setup times. European Journal of Operational Research. 211(3), 612-622.

Vela, C.R., Varela, R., Gonzalez, M.A., 2010. Local search and genetic algorithm for the job shop scheduling problem with sequence-dependent setup times. Journal of Heuristics. 16(2), 139-165.

Viswanathan, K.V., Sen, A.K., 2010. Greedy by chance – Stochastic greedy algorithms. Proceedings of the Sixth International Conference on Autonomic and Autonomous Systems (ICAS 2010), pp. 182-187.

Wang, J-B., 2008. Single-machine scheduling with past-sequence-dependent setup times and time-dependent learning effect. Computers and Industrial Engineering. 55(3), 584-591.

Wang, X., Tang, L., 2010. A hybrid metaheuristic for the prize-collecting single machine scheduling problem with sequence-dependent setup times. Computers and Operations Research. 37(9), 1624-1640.

Zhao, C., Tang, H., 2010. Single machine scheduling with past-sequence-dependent setup times and deteriorating jobs. Computers and Industrial Engineering. 59(4), 663-666.

Tables

Table 1: 4-job QPSD problem

Job	Setup Times				Proc. Times	Penalty Coeff.
	1	2	3	4		
1	-	1	1	3	1	2
2	1	-	3	2	4	1
3	5	4	-	10	3	1
4	3	6	9	-	10	1

Table 2: QPSD_MDFBB with 512k node limit

Node limit = 512k nodes			
Job size	QPSD MDFBB		
	Optimal Cost	Node Gen	Time (secs)
20	17953992	163459	0.70
22	21587074	461720	2.32
24	31098438	1088649	6.3
26	36795831	3357289	23.15
28	48335631	13514071	110.72
30	58034706	39696969	398.57

Table 3: Performance of greedy algorithm, QPSD_GREEDY

No. of jobs (N)	% Deviation from optimal: 100*(value-opt)/opt
10	7.70
11	9.06
12	10.02
13	11.04
14	11.13
15	11.02
16	11.05
17	12.22
18	12.50
19	13.64
20	13.91
21	14.66
22	14.45
23	14.51
24	14.41
25	14.63
26	15.64
27	15.22
28	15.55
30	15.71

Table 4: Performance of greedy and stochastic greedy algorithms under binomial and geometric distributions (averages over 100 problem instances for each value of N)

No. of jobs (N)	% Deviation from optimal: 100*(value-opt)/opt		
	Greedy	SGA – Binomial (p = 0.025)	SGA – Geometric (p = 0.85)
10	7.70	1.67	2.07
11	9.06	2.53	3.10
12	10.02	2.81	3.48
13	11.04	3.04	4.05
14	11.13	3.70	4.65
15	11.02	3.66	4.73
16	11.05	4.01	5.14
17	12.22	4.70	6.18
18	12.50	4.93	6.20
19	13.64	5.90	7.57
20	13.91	6.10	7.61
21	14.66	6.95	8.12
22	14.45	7.11	8.34
23	14.51	7.51	8.59
24	14.41	7.43	8.93
25	14.63	7.70	9.05
26	15.64	8.31	9.92
27	15.22	8.13	9.70
28	15.54	8.53	9.75
30	15.71	9.22	10.56

Table 5 - Performance of QPSD_GEN

No of jobs	CPU Time (secs)		% Deviation from optimal: 100*(value-opt)/opt	
	QPSD_MDFBB (512K node limit)	QPSD_GEN	QPSD_GEN	GREEDY
10	0.001	0.157	0.009	7.704
11	0.003	0.181	0.027	9.063
12	0.005	0.207	0.012	10.020
13	0.010	0.236	0.022	11.043
14	0.017	0.266	0.103	11.131
15	0.031	0.298	0.078	11.017
16	0.061	0.337	0.071	11.051
17	0.125	0.374	0.132	12.223
18	0.205	0.412	0.153	12.497
19	0.382	0.457	0.136	13.645
20	0.700	0.500	0.180	13.906
21	1.303	0.548	0.229	14.664
22	2.328	0.598	0.224	14.447
23	3.791	0.654	0.301	14.508
24	6.332	0.705	0.260	14.413
25	16.240	0.764	0.361	14.626
26	22.934	0.817	0.496	15.643
27	44.129	1.480	0.398	15.223
28	110.645	1.533	0.427	15.549
30	382.3298	2.148	0.483	15.715

Figures

Algorithm *QPSD_MDFBB* {

Compute the effective processing times $e_{i,j} = s_{i,j} + a_j \forall i \in \{0..N\}, j \in \{1..N\}$.

Sort the $\frac{p_j}{e_{i,j}}$ values in non-ascending order $\forall i \in \{0..N\}, j \in \{1..N\}$ (for heuristic computation and successor ordering).

Generate the start node *s*, represented as $(J(s), L(s))$ where *J(s)* is the set of jobs completed and L(s) represents the last job; *J(s)* is empty and *L(s) is null*

$T(s) \leftarrow 0$ /* completion time of jobs in s */

$b(s) \leftarrow h(s)$ /* *h(s)* is the heuristic estimate of node s */

$bound \leftarrow \infty$

Call $QPSD\_REC(s, bound)$

}

Function *QPSD_REC(n, bound)* {

if n represents a complete schedule *then* return 0

$newbound \leftarrow \infty$

for each of the successors n_i of *n do*

Generate successor n_i by scheduling an unscheduled job *k*

$J(n_i) \leftarrow J(n) \oplus k$ and $L(n_i) \leftarrow k$

$T(n_i) \leftarrow T(n) + e_{L(n),k}$

$c(n, n_i) \leftarrow p_k T(n_i)^2$

if n_i is not already stored set $b(n_i) \leftarrow h(n_i)$

if $c(n, n_i) + b(n_i) < bound$ *then*

$b(n_i) \leftarrow QPSD\_REC(n_i, bound - c(n, n_i))$

Store node n_i if memory is available or if a less recently used node can be replaced by n_i

$newbound \leftarrow \min\{c(n, n_i) + b(n_i), newbound\}$

$bound \leftarrow \min(bound, newbound)$

return newbound

}

Fig. 1: Algorithm QPSD_MDFBB

Algorithm *QPSD_GREEDY* {

Let $s_{0,j}$ represent the setup time for J_j when it is the first job to be scheduled.

For job J_j, calculate its effective processing time when it follows job J_i as

$e_{i,j} = a_j + s_{i,j} \forall 0 \le i \le N; 1 \le j \le N$

Order the jobs such that $\frac{p_{k(1)}}{e_{0,k(1)}} \ge \frac{p_{k(2)}}{e_{0,k(2)}} \ge .... \frac{p_{k(N)}}{e_{0,k(N)}}$

Schedule job $J_{k(1)}$ as the first job

$r \leftarrow k(1)$

For $i \leftarrow 2..N$ do the following {

Order the $N-i+1$ unscheduled jobs such that $\frac{p_{k'(1)}}{e_{r,k'(1)}} \ge \frac{p_{k'(2)}}{e_{r,k'(2)}} \ge .... \frac{p_{k'(N-i+1)}}{e_{r,k'(N-i+1)}}$

Schedule job $J_{k'(1)}$ as the i[th] job

$r \leftarrow k'(1)$

}

Fig. 2: Algorithm QPSD_GREEDY

Algorithm *QPSD_GSA*

Let $s_{0,j}$ represent the setup time for J_j when it is the first job to be scheduled.

For job J_j, calculate its effective processing time when it follows job J_i as

$e_{i,j} = a_j + s_{i,j} \forall 0 \le i \le N; 1 \le j \le N$

Order the jobs such that $\frac{p_{k(1)}}{e_{0,k(1)}} \ge \frac{p_{k(2)}}{e_{0,k(2)}} \ge .... \frac{p_{k(N)}}{e_{0,k(N)}}$

Pick job $J_{k(x)}$ randomly based on a predefined probability distribution.

Schedule job $J_{k(x)}$ as the first job

$r \leftarrow k(1)$

For $i \leftarrow 2..N$ do the following {

Order the $N-i+1$ unscheduled jobs such that $\frac{p_{k'(1)}}{e_{r,k'(1)}} \ge \frac{p_{k'(2)}}{e_{r,k'(2)}} \ge .... \frac{p_{k'(N-i+1)}}{e_{r,k'(N-i+1)}}$

Pick job $J_{k(x)}$ randomly based on a predefined probability distribution.

Schedule job $J_{k'(x)}$ as the i[th] job

$r \leftarrow k'(x)$

}

Fig. 3: Algorithm QPSD_GSA

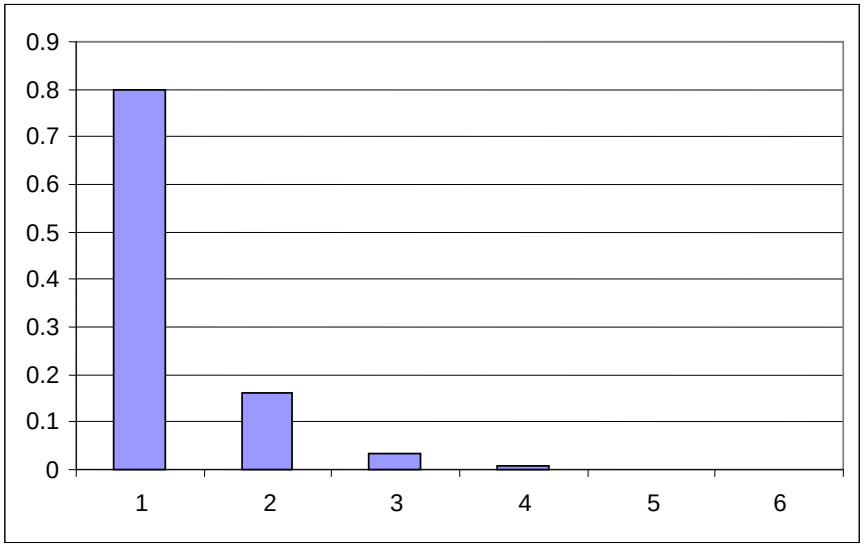

Fig. 4: Geometric distribution with p = 0.85 (translated to the right by 1 to avoid 0)

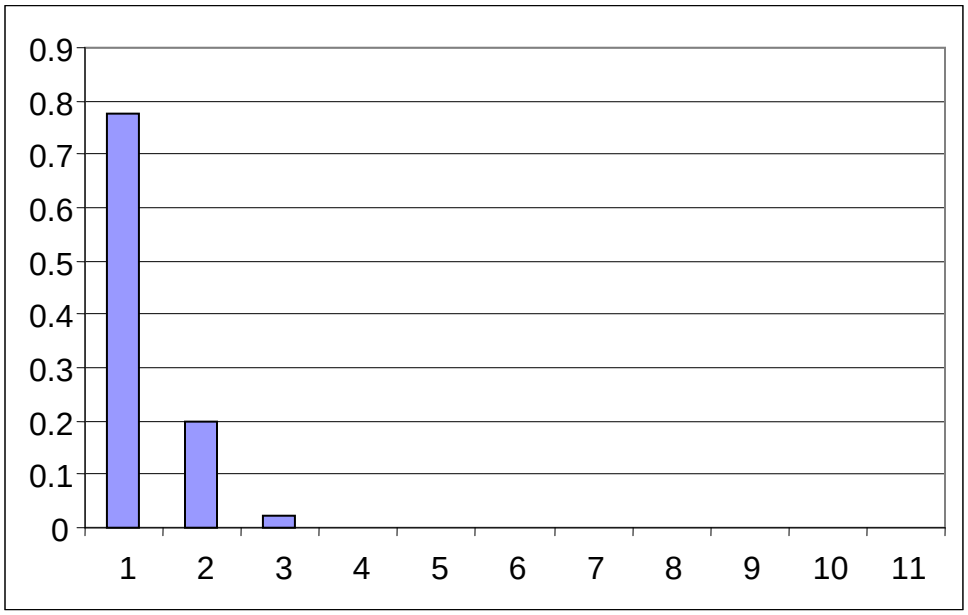

Fig. 5: Binomial distribution with p = 0.025 (translated to right by 1 to avoid 0)

Initial configuration:

Chromosome 1	3, 4, 2, ▲ 1, 6, 5
Chromosome 2	1, 3, 5, ▲ 4, 2, 6

Immediately after exchange:

Intermediate child 1	1, 3, 5, ▲ 1, 6, 5
Internediate child 2	3, 4, 2, ▲ 4, 2, 6

Final configuration (after repair):

Child 1	1, 3, 5, ▲ 4, 6, 2
Child 2	3, 4, 2, ▲ 1, 5, 6

Fig. 6: Example of crossover

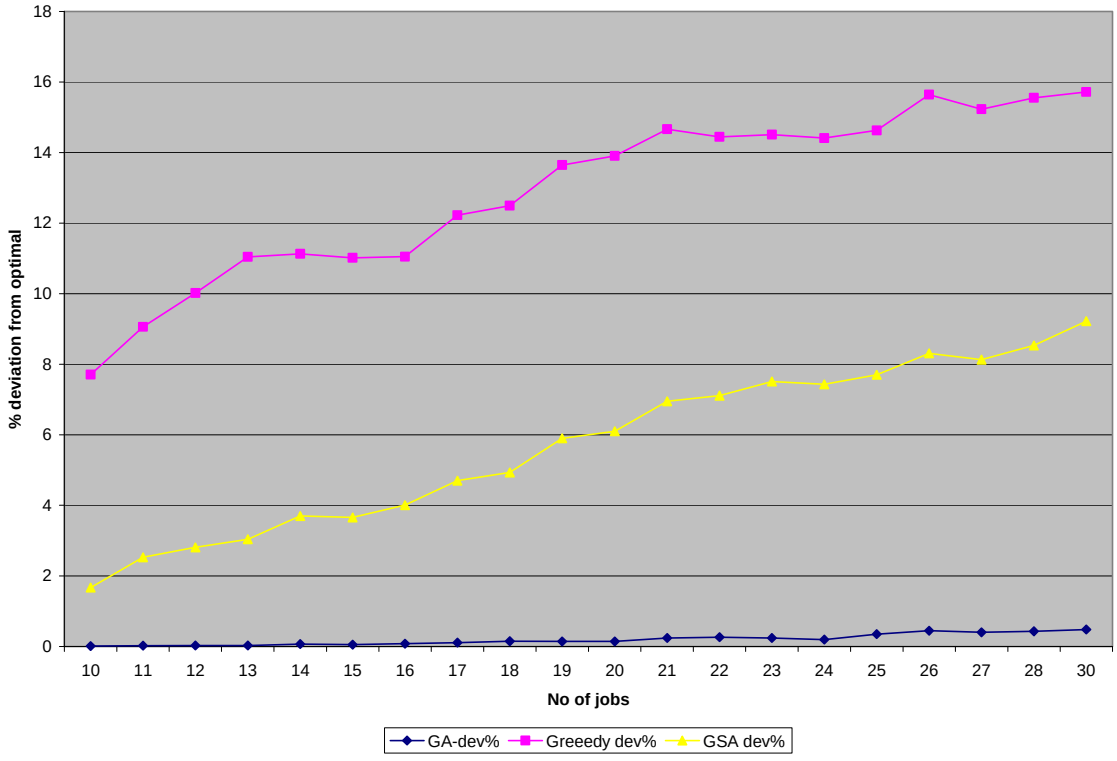

Fig. 7: Percentage deviation from optimal for QPSD_GEN and the greedy approaches