Stochastic Local Search for Multiprocessor Scheduling for Minimum Total Tardiness

Michael Pavlin[1], Holger H. Hoos[1]*, and Thomas Stützle[2]

[1] Department of Computer Science, University of British Columbia,
Vancouver, B.C., V6T 1Z4, Canada
{mpavlin,hoos}@cs.ubc.ca

[2] Department of Computer Science, Darmstadt University of Technology,
Alexanderstr. 10, D-64289 Darmstadt, Germany
stuetzle@informatik.tu-darmstadt.de

Abstract. The multi-processor total tardiness problem (MPTTP) is an $\mathcal{NP}$-hard scheduling problem, in which the goal is to minimise the tardiness of a set of jobs that are processed on a number of processors. Exact algorithms like branch and bound have proven to be impractical for the MPTTP, leaving stochastic local search (SLS) algorithms as the main alternative to find high-quality schedules. Among the available SLS techniques, iterated local search (ILS) has been shown to be an effective algorithm for the single processor case. Here we extend this technique to the multi-processor case, but our computational results indicate that ILS performance is not fully satisfying. To enhance ILS performance, we consider the use of population-based ILS extensions. Our final experimental results show that the usage of a population of search trajectories yields a more robust algorithm capable of finding best known solutions to difficult instances more reliably than a single ILS trajectory.

1 Introduction

Given a set of tasks to be processed on a set of machines, we find ourselves faced with a scheduling problem. Scheduling problems arise in many situations ranging from fulfilling orders in a factory or scheduling processes on a multitasking computer to automated decision making in a robot. In this paper we consider the identical multi-processor scheduling problem, where the objective function is to minimise the total tardiness. This problem is referred to as the multi-processor total tardiness problem (MPTTP); the single processor variant, where only a single processor is available for processing the tasks, will be referred to as single-processor total tardiness problem (SPTTP).

The SPTTP was shown to be $\mathcal{NP}$-hard [1] and based on this result, the MPTTP can be shown to be at least as hard via reduction [2]. This implies that it is unlikely that polynomial time algorithms exist to solve these problems [3]. The MPTTP is also very difficult to solve in practice, which is reflected by the fact that with growing instance size, this problem rapidly becomes intractable for exact optimisation algorithms. Consequently, approximation algorithms of various forms have been developed. Construction heuristics and stochastic local search (SLS) algorithms form the bulk of this work. We concentrate on a particular SLS algorithm, Iterated Local Search (ILS), which

* To whom correspondence should be addressed.

was shown to be the top performer on the SPTTP [4, 5]. However, on the MPTTP, our experimental results indicate stagnation behaviour for the single search trajectory ILS. To achieve improved performance, we apply population-based extensions of ILS, where a population of ILS search trajectories is evolved using operators gleaned from evolutionary computation, as well as a memetic algorithm [6].

The remainder of this paper is organised as follows. In Section 2 we introduce the SPTTP and the MPTTP and briefly review algorithmic approaches for their solution. Section 3 introduces ILS, the primary SLS technique considered in this paper, and Section 4 gives details on the single-trajectory ILS algorithms as well as the population-based extensions. Section 5 presents the computational results and we conclude in Section 6.

2 The Multiprocessor Total Tardiness Problem

Problem Formulation. In the MPTTP we are given n jobs $\{job_0, job_1, \ldots, job_{(n-1)}\}$ that have to be processed on a set of m identical processors. Each job_i has associated an integer processing time p_i and an integer due date d_i. A schedule consists of a list of jobs for each processor and a start time for each job such that (i) each job is assigned to exactly one processor, (ii) jobs cannot be preempted, and (iii) jobs cannot overlap on a single processor. Given a schedule, the *tardiness* of job_i is $T_i := max\{0, C_i - d_i\}$, where C_i is the completion time of job_i and the total tardiness is given by $\sum T := \sum_{i=0}^{n-1} T_i$. The objective in the MPTTP is to find a schedule with minimal total tardiness. Note that the SPTTP is a special case of the MPTTP with $m = 1$.

Total tardiness is a regular objective function, which means that it is monotonic with respect to all C_i. In the MPTTP, we only need to consider schedules without idle time on any single processor. Solving the MPTTP involves finding an assignment from jobs to processors and for each single processor finding a permutation of all the jobs assigned to it. One possible way to represent schedules is as an assignment of a permutation of the jobs to each processor (*permutation representation*). The order of the jobs assigned to a processor is determined by the order in which the jobs appear in the permutation.

An alternative to the permutation representation is to translate a schedule into a single list of all jobs ordered by non-decreasing start times. From this list, a new schedule at least as good as the initial schedule can be created by greedily assigning jobs from the start of the list to processors. Together, the greedy algorithm and the list form a valid representation which we will call the *priority-list representation*. This representation will be considered for both population and single trajectory algorithms.

Exact Algorithms. Several branch and bound algorithms for the SPTTP and MPTTP have been proposed and studied in the literature. Computational results show that even fairly large instances of the SPTTP with up to 500 jobs can be solved in reasonable computation time [7, 8]. This is possible by exploiting fundamental domination properties of the SPTTP reported by Emmons [9] and Lawler [2]. However, the good performance of exact algorithms for the SPTTP does not carry over to the MPTTP. The branch and bound algorithm by Azizoglu and Kirca becomes impractical for instances with more than 15 jobs on multiple processors [10]. This difference stems from the added difficulty of achieving an appropriate partitioning of jobs to processors. Because of the poor performance of exact algorithms, local search based algorithms have to be used for solving large MPTTP instances.

Note that the much worse performance of exact algorithms in the multi-processor version compared to the single-processor case appears to be rather typical and was observed also for exact algorithms attacking the related problem of minimising the number of tardy jobs [11].

Stochastic Local Search. Various approximation algorithms such as construction, list scheduling, and decomposition heuristics have been used to quickly find reasonable solutions [2]. In this paper we concentrate on Stochastic Local Search (SLS) approaches to this problem. SLS algorithms combine a local search algorithm with a probabilistic component. Well known SLS algorithms include tabu search, variable neighbourhood search, ant colony optimisation (ACO) and evolutionary algorithms. All of these have been applied to MPTTP or to closely related problems [4, 5, 11–16].

Currently, the best performing SLS algorithms for the single processor total tardiness and for the more difficult total weighted tardiness problem (SPTWTP), where each job is given an additional weight indicating its importance, are iterated local search algorithms [4, 5]. Therefore, we also considered this type of SLS algorithm first to attack the MPTTP.

3 Iterated Local Search (ILS)

Iterated Local Search is a simple yet powerful SLS method, which is witnessed by excellent computational results for a variety of combinatorial optimisation problems like the travelling salesman problem (TSP) and several scheduling problems (see [17] for an overview). In a nutshell, ILS builds a biased random walk in the space of the local optima (with respect to some local search algorithm). This is done by iteratively perturbing a locally optimal solution, then applying a local search algorithm to obtain a new locally optimal solution, and finally using an acceptance criterion for deciding from which of these solutions to continue the search. To implement an ILS algorithm, the following four procedures have to be defined: GenerateInitialSolution generates a starting point for the search, LocalSearch implements a (possibly complex) improvement procedure, Perturbation computes a perturbation of a given locally optimal solution, and AcceptanceCriterion chooses from which of two candidate solutions the search is continued. A general algorithmic outline of ILS is given in Figure 1.

An efficient ILS algorithm for the SPTTP was described by den Besten [13]; an extension of this algorithm is currently a state-of-the-art algorithm to the harder SPTWTP [4]. Excellent results for the SPTWTP were also reported for the iterated Dynasearch algorithm of Congram, Potts and Van de Velde [5]. Den Besten's ILS for the SPTTP uses the interchange neighbourhood in the local search and in the perturbation. This neighbourhood considers all moves which consist of selecting two jobs, job_i and job_j, $i \neq j$, and exchanging their positions. The local search is a best-improvement algorithm which terminates at the first local optimum encountered. The perturbation is im-

```
procedure Iterated Local Search
    $s_0$ = GenerateInitialSolution
    $s^*$ = LocalSearch($s_0$)
    repeat
        $s'$ = Perturbation($s^*$)
        $s^{*\prime}$ = LocalSearch($s'$)
        $s^*$ = AcceptanceCriterion($s^*, s^{*\prime}$)
    until termination condition met
end
```

Fig. 1. Algorithm outline of Iterated Local Search (ILS).

plemented by applying a fixed number of random interchange moves to the candidate solution. New solutions are accepted if they strictly improve over the original solution and the algorithm terminates when a certain solution quality is reached or some maximum computation time expires. This algorithm performed well and found optimal solutions reliably.

Typically, ILS is a single-trajectory method. However, ILS can easily be extended into a population-based SLS method by independently applying a standard ILS algorithm to a population, that is, a set of candidate solutions, and allowing some limited interaction between the population elements. Such extensions have strong similarities to well-known population-based search metaphors such as evolutionary algorithms [18, 19] and, in particular, memetic algorithms [6, 20].

Population-based extensions of ILS were independently proposed in [21, 22] and applied to the TSP and the QAP. In particular, Stützle reported results for different levels of interaction among the individual ILS search threads: no-interaction, replace-worst and population-ILS. *No-interaction*, as the name implies, is simply a population of ILS trajectories run in isolation; at the end of these runs, the best solution is chosen. *Replace-worst* is similar but during the evaluation sometimes replaces the worst current solution by the best solution. *Population-ILS* generates a population of solutions and chooses, based on a selection operator, one individual to pursue a number of ILS iterations. The new solution is inserted into the population and the process is iterated. Stützle reported varying degrees of success with this approach. The performance of all three variants was found to be similar and appeared to depend strongly on the given problem instance. Similarly, Hong, Kahng, and Moon [21] presented computational results that did not indicate a significant improvement of the population-based ILS over a single-trajectory ILS for the TSP.

4 Our ILS Algorithms for the MPTTP

The MPTTP problem intuitively involves two subproblems, the problem of partitioning the set of jobs, where each partition is assigned to one of the processors, and a single processor total tardiness scheduling problem on each processor. Unfortunately, these two subproblems cannot be solved independently of each other. Additionally, to evaluate the quality of a given partition, one needs to solve the SPTTP on each single processor. Preliminary results with an algorithm that did not explicitly search on the partition subproblem resulted in poor performance and therefore, the algorithms we present here apply local search to both subproblems. Here, we present two single trajectory ILS algorithms that mainly differ in the perturbation step as well as several population-based ILS algorithms and a memetic algorithm for the MPTTP.

4.1 Single trajectory ILS

Initialisation and Termination. In all the experiments reported here we considered random starting solutions. To obtain these, first a list is generated containing the jobs in random order. Then, the head of the list is recursively removed and assigned to the processor with the smallest total processing time. This randomised initialisation scheme is a reasonable choice for initialising a number of trajectories in a population based extension if the initial solution population should be scattered across the search space.

Additionally, we considered ordering the list of jobs by non-increasing processing time (GPT) and by non-decreasing due date (EDD) and then assigning the jobs in the same way. Both GPT and EDD were used for testing pivoting rules in the local search (see below). The EDD construction heuristic was also used as an initialisation procedure in the protocol for finding optimal solutions.

The algorithm terminates when a designated tardiness value is achieved or some maximum computation time limit is reached.

Local Search. The local search procedure is divided into two phases. The first phase applies a local search to the jobs assigned to each processor independently of the other processors and reorders the jobs to minimise the total tardiness on the individual processor (LocalSearchOnSingleProcessors). The second phase moves jobs between the different processors and, thus, modifies the assignment of jobs to processors (LocalSearchBetweenProcessors).

Both phases are iterative improvement algorithms which repeatedly interchange pairs of jobs. Both, best and first improvement pivoting rules as well as several other variations were considered in some preliminary experiments. First improvement returned slightly better solutions than best improvement, but it was significantly slower than the best-improvement algorithm, once the local search had been optimised (see Table 1 for sample results on single processor instances). Since in any ILS algorithm, local search has to be applied very frequently, we decided to use the significantly faster best-improvement local search. The “between processors” search similarly performs interchange moves but considers only pairs of jobs which are on different processors.

Acceptance Criterion. In the single trajectory ILS algorithms, a new candidate solution is accepted only if it has a smaller total tardiness value than that of the best solution seen so far. We call this acceptance criterion Better and it is defined for minimisation

Algorithm	Best Improvement		First Improvement	
init	Avg. time (s)	Avg. soln	Avg. time (s)	Avg. soln
EDD	1.47	140353	5.34	139793
GPT	1.86	140267	22.19	139807
Random	1.52	140493	10.01	139794

Table 1. Given are the average computation time and the average solution values when comparing best vs. first improvement pivoting rules on all 200 job single processor benchmark problems using three different initialisation schemes.

```
procedure ILS_I
    $s_0$ = GenerateInitialSolution
    $s^+$ = LocalSearchOnSingleProcessors($s_0$)
    $s^*$ = LocalSearchBetweenProcessors($s^+$)
    repeat
        $s'$ = PerturbationBetweenProcessors($s^*$)
        $s^{+\prime}$ = LocalSearchOnSingleProcessors($s'$)
        $s^{*\prime}$ = LocalSearchBetweenProcessors($s^{+\prime}$)
        $s^*$ = Better($s^*, s^{*\prime}$)
    until termination condition met
end
```

Fig. 2. Algorithm outline of ILS$_I$.

problems as:

$$\text{Better}(s^*, s^{*\prime}) = \begin{cases} s^{*\prime} & \text{if } f(s^{*\prime}) < f(s^*) \\ s^* & \text{otherwise} \end{cases} \tag{1}$$

where $f(s)$ denotes the objective function value of a solution s. Note that this acceptance criterion implements an iterated descent in the search space of local optima.

Perturbation and Partitioning of Jobs. The two single trajectory ILS algorithms only differ in the kind of perturbation they apply. In ILS$_I$ a number of insert moves is applied to the current candidate solution. This is done because good perturbations should somehow be complementary to the type of moves applied in the local search [17]. In particular, the perturbation allows to modify the number of jobs assigned to the processors, something which is not possible in the local search.

The perturbation proceeds as follows: First, a processor is randomly chosen, second, a job job_i is randomly selected from the jobs on this processor and removed; last, job_i is then inserted on a randomly chosen processor in a random position. (All random decisions are made according to a uniform distribution.) An outline of ILS$_I$ is shown in Figure 2.

In the problem formulation, possible representations of schedules were discussed. Up to this point, we have considered only the permutation representation that is used

```
procedure ILSRES
    $s_0$ = GenerateInitialSolution
    $s^+$ = LocalSearchOnSingleProcessors($s_0$)
    $s^*$ = LocalSearchBetweenProcessors($s^+$)
    repeat
        $PL^*$ = CreatePriorityList($s^*$)
        $PL'$ = PerturbPriorityList($PL^*$)
        $s'$ = CreateSchedule($PL'$)
        $s^{+\prime}$ = LocalSearchOnSingleProcessors($s'$)
        $s^{*\prime}$ = LocalSearchBetweenProcessors($s^{+\prime}$)
        $s^*$ = Better($s^*, s^{*\prime}$)
    until termination condition met
end
```

Fig. 3. Algorithm outline of ILSRES.

in ILS$_I$. ILSRES tries to exploit the priority-list representation during the perturbation phase of each iteration. The ILSRES algorithm transforms the current candidate solution, which is in permutation representation, to the appropriate priority list before each perturbation. During the perturbation, the priority list is modified by performing a predefined number of interchange moves. Following the perturbation, a schedule is constructed by greedily assigning the jobs from the front of the list to the processor with the least load as described before. At this point, the local search phase begins. Figure 3 gives the pseudo-code for this algorithm.

4.2 Population Based Algorithms for the MPTTP

Some preliminary results indicated that the single-trajectory ILS algorithm for the MPTTP shows stagnation behaviour, something that was not observed for the SPTTP. To overcome this stagnation behaviour, we developed population-based extensions of the ILS algorithms. We also considered the extension to memetic algorithms [6, 20] by including a recombination operator that is applied with some probability to pairs of solutions and returns a new solution that combines properties of both "parents".

We developed four population-based ILS algorithms, ranging from the *no-interaction* scheme mentioned in Section 3 to a memetic algorithm. All algorithms have the same general skeleton and maintain a fixed size population of ILS trajectories. Each trajectory is obtained by performing ILSRES with identical perturbation strength after initialising with a randomly generated solution. In the main loop, solutions are selected to apply one single iteration of the standard ILS algorithm. Finally, if appropriate, the algorithm performs any necessary interaction between trajectories and repeats.

The first two approaches are adaptations of the *no-nteraction* (referred to as POPILS) and the *replace-worst* (referred to as POPREP) schemes [22] introduced in Section 3. POPILS serves as a baseline for assessing whether the interaction among the ILS trajectories improves performance. In POPREP, every μ iterations the best schedule is replaced by the worst schedule.

```
procedure Population based ILS algorithm
    for each solution $s[i]$
        $s[i]$ = GenerateRandomInitialSolution
    repeat
        if selection criteria reached
            then SelectionAlgorithm($s[]$)
        for each solution $s[i]$
            ILSRES Iteration($s[i]$)
    until termination condition met
end
```

Fig. 4. Algorithm outline of population based algorithms.

Parents	α	$=$	1	2	3	4	5	6	7
	β	$=$	7	6	5	4	3	2	1
Modified parents	α'	$=$	1	b	b	b	5	6	7
	β'	$=$	b	b	b	4	3	2	b
Offspring, step 1	γ'	$=$	1				5	6	7
Offspring, step 2	γ	$=$	1	4	3	2	5	6	7

Fig. 5. Example of application of recombination operator to two schedules α and β resulting in a new schedule γ

The following two algorithms, POPSEL and POPSTG, use the framework described in Figure 4 but employ a more advanced selection strategy which will be referred to as *genetic selection*. Genetic selection is implemented according to an elitist roulette wheel strategy. In this scheme, the selection probability is proportional to the fitness of a schedule; exceptions are that the best solution is always chosen and recombination does not make sense if two solutions are the same. The fitness of a schedule i is given by $\textit{fitness}(\textit{schedule}_i) = (T_{\textit{worst}} - T_i) + \delta \cdot (T_{\textit{worst}} - T_{\textit{best}})$, T_i is the tardiness of schedule i, $T_{\textit{worst}}$ and $T_{\textit{best}}$ are the best and the worst schedule in the population, and δ is a parameter.

In addition, POPSEL and POPSTG employ a recombination operator. The number of trajectories seeded by the recombination operator is determined by a parameter $\psi = k/n$; this parameter gives the ratio of trajectories derived by recombination to the total number of trajectories n; k is an integer between 0 and $n-1$. One trajectory is reserved for the current best solution and the remaining trajectories are selected from all current solutions with probabilities determined by their relative fitness. POPSEL and POPSTG obey the population based ILS skeleton indicated in Figure 4. POPSEL and POPSTG use the genetic selection algorithm but differ in when selection is applied. POPSEL applies selection after a constant number of ILS iterations defined by a parameter μ. POPSTG applies selection only if there has been no improvement to the best trajectory for ν ILS iterations, where ν is a parameter.

The recombination operator is applied to two parent schedules, α and β, represented by priority lists of jobs, and works as follows (see also Figure 5 for an illustration): A random set of jobs in α are determined to be static and the nonstatic jobs in α are replaced by blanks resulting in α' (indicated by b in Figure 5). Next, the static jobs are removed from β, resulting in β'. The offspring γ is build by first copying the static jobs of α into their positions and then filling the empty position with the jobs of β' maintaining their order in β'. The recombination operator has the properties that (i) all the jobs originating from α and β maintain their relative orderings and (ii) the jobs originating from α also maintain their positions in the priority list. Hence, we would expect that the start time of jobs in γ that were copied from α remain approximately similar to the ones they had in α. This operator is based upon the strategy of França [16]. The two approaches differ mainly in that França chooses an interval in α which remains static, while we choose the jobs randomly, and França's implementation is for a single processor scheduling problem, while ours is for the MPTTP.

5 Empirical Analysis

This sections describes the benchmark instances we used in our analysis, the experimental design and our computational results.

5.1 Benchmark instances

We generated a set of forty 100 and 200 job instances. 20 of these instances were taken from a problem library maintained by Bahar Kara [23] with a single instance per range of due date (RDD) and tardiness factor (TD) for each value pair for RDD in $\{0.2, 0.4, 0.6, 0.8, 1.0\}$ and TF in $\{0.2, 0.4, 0.6, 0.8\}$. (RDD and TF are parameters that are used in a widely applied algorithm for generating instances of single processor tardiness problems [2, 23].) In addition, we generated 20 instances using the same parameter value pairs. The so derived SPTTP instances were adapted to the multi processor case by dividing their due dates by the number of processors and rounding to the nearest integer.

To evaluate the performance of our algorithms, we first tried to find very high quality solutions using the following protocol. A single trajectory ILS algorithms with perturbation set at 10% of the number of jobs was run ten times on each instance, five times with the earliest due date initialisation routine and five times with random initialisation. The termination condition requires that at least 1000 iterations have been completed. If this criteria has been satisfied, the termination condition will stop the ILS if it did not find a better solution for $10n$ iterations. The best solution found was deemed to be the best-known solution and used as a goal for subsequent experiments.

This protocal reliably returned the known optimal solutions for the 100 and 200 job SPTTP instances of Bahar Kara; generally 90 to 100% of the runs returned the same potentially optimal solution quality. This is a strong indication that the solutions found for the additional SPTTP instances using this protocol are likely to be optimal. However, this did not generally translate to the multi-processor case where for the most difficult problems the runs resulted in different solutions; here we use the best solution returned by the 10 runs to evaluate the other algorithms.

5.2 Experimental Design

Most of the results reported in this paper are from runs on only a few instances, which were among the hardest ones. The instances were selected from the pool based on how quickly and easily known optimal solutions were found using the procedure described in the previous section. If the problem was solved in the first iteration, it was deemed trivial and not further studied. If the best solution was found in 80% or more of the runs, the problem was classified as easy. If on the other hand the best solution was found less than 20% of the trials, the problem was classified as difficult. The forty 200 job, 2 processor instances divided into 57.5% difficult and 40% easy instances with the remaining two instances being between these two extremes. We found that instances with RDD of 0.2 and 0.4 are rather easy and found consistent solutions 100% of the time in all but one case. On the instances with other RDD values, in almost all cases the best solution was only obtained in one trial, suggesting that there still is some gap to the true optimum.

Four instances were selected from these sets; `bk131` is an easy instance and both `bk151` and `bk181` are difficult. `mp101` was selected because its RDD parameter of 0.6 has been shown to be particularly difficult for the SPTTP [13].

All experiments were performed on the BETA laboratory compute cluster at UBC. At the time of this work, the cluster consisted of 12 PCs running Redhat Linux version 7.1 with 733MHz and 1GHz Pentium III processors with 256KB CPU cache and 1–4GB RAM.

The cost model used throughout the experiments is based on the number of local search moves completed. The average CPU time per search step is dependent on the specific instance. However, on a given instance the CPU time per search step is not dependent on the particular ILS algorithm variant considered here. For instance `bk151`, on a 1GHz processor with 1GB RAM running a Linux executable written in C++ and compiled with g++, ILS performs 839 search steps per second. This has been tested with both ILS$_I$ and ILSRES and several predecessors. For `bk131` the results are less consistent but indicate approximately 700 search steps per second.

5.3 Computational Experiments

Evaluation of ILS$_I$ and ILSRES. An evaluation of perturbation parameters for MPTTP was performed with the ILS$_I$ and ILSRES algorithms. The experiments were performed by running the ILS algorithms 25 times for each algorithm / instance / perturbation level combination. The instances `bk131` and `bk151`, adjusted to two processors, were considered. Perturbation levels of 2,4,6,8,10 were tested. This was repeated for ILSRES on instance `bk151` adjusted for 3 and 4 processors. All runs were limited to a maximum CPU time of 500 seconds. Analysis was performed by acquiring summary statistics for computational requirements and solution qualities as well as by analysing run-time distributions (RTDs).

The easy problem was solved to the best-known solution (which we conjecture to be optimal) within the given time frame. This is not true for the difficult instance and here we can only consider results for solution qualities inferior to the best-known. Notably, in both instances and for both algorithms, the optimal perturbation appears at the low

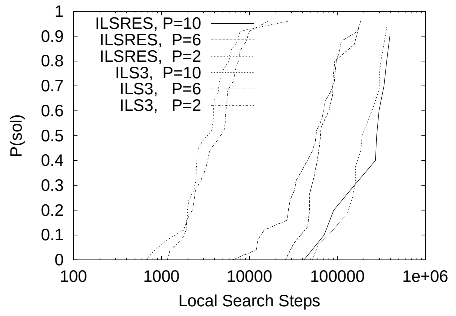

Fig. 6. Run time distribution for ILSRES and ILS$_I$ run on 2 processor `bk151`.

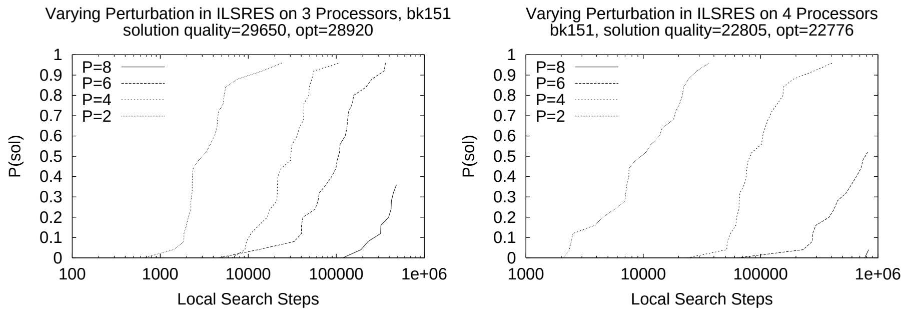

Fig. 7. Run time distribution for ILSRES run on 3 and 4 processor `bk151`.

end of the spectrum, at level 2 in ILSRES and ILS$_I$ (see Figure 6 for instance `bk151` on two processors). As demonstrated in Figure 6, these two algorithms perform almost identically on both of these instances. Figure 7 shows that for instances with three or four processors, the perturbation strength should also be rather small.

Evaluation of Population Based Algorithms. All algorithms were evaluated with multiple parameter settings across 25 trials on instances `bk131` and `bk151` on 2 processors using a maximum processing time of 500 seconds. In addition, ILSRES and POPILS with parameters optimized based on the `bk151` experiments were run on `bk181` and `mp101`.

Throughout these experiments all population based algorithms' internal ILSRES trajectories used a perturbation strength of size 2 and the population size was set to 20. POPREP was evaluated with μ equal to 1, 10 and 100. POPSEL was evaluated with μ equal to 5, 10 and 100 and a constant recombination ratio of $\psi = 0.2$. Similarly, POPSTG was considered with ν at 10, 100 and 500 with $\psi = 0.2$, and $\nu = 10$ with ψ at 0.2 and 0.4. The results of the application of these algorithms to `bk151` are shown in Table 2 and Figure 9. Table 3 shows results for the other three instances. Most sig-

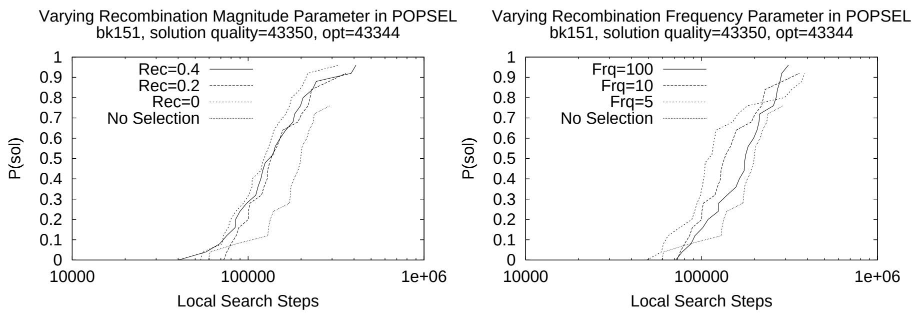

Fig. 8. Altering parameters of the selection operation.

nificantly, on the difficult `bk151` instance, the population based algorithms find the best-known solution more often than single trajectory ILS algorithms. Given the steep slope of the RTD in Figure 9 for POPSEL, there is a strong indication that a small increase in alotted time will be sufficient to find this solution quality in all trials. In contrast, the slope of the single trajectory algorithms is less steep and we can expect little benefit from increasing the allotted time. The stagnation of the single trajectory algorithms is supported by the fact that with a perturbation strength of 2, the single trajectory ILS algorithms usually perform most of their improvements before 250 seconds have elapsed. This is supported by the low average time to best in Tables 2 and 3 indicating stagnation on the difficult instances. Given these results, it appears that the population based strategy is more promising than concentrating the search effort on a single trajectory.

On the easier `bk131` instance, most runs achieved the best known solution quality. Generally, less than 5 iterations were required and there was no evidence of stagnation for the single trajectory algorithms. On this instance, very little difference could be observed between the techniques. As shown in Table 2, runs on instance `bk151` rarely complete in the 500 allotted seconds. In this time period, a population based algorithm performs up to 1000 iterations and characteristics of the algorithms become apparent. Both the run-time distributions in Figure 9 and Table 2 indicate the benefits of a genetic selection mechanism. The mean solution of POPILS is more than a standard deviation from the best POPSEL and POPSTG settings. In addition, the number of optimal solutions found by strategies employing genetic selection is consistently greater than zero, the number found by POPILS. The run-time distributions in Figure 7 also show that strategies based on genetic selection achieve the suboptimal solution quality 43350 faster and more reliably.

Establishing the possible benefits of recombination is more difficult. We observe that in the run time distribution in Figure 8, for reaching a suboptimal solution quality, the algorithm with no recombination appears to fare best. This is supported by the mean solution quality found which is smallest for $\psi = 0$. An important measure contradicting this point is the number of optimal solutions found, which is greatest when recombination is at its peak.

	Time to find best		Solution Quality		
Algorithm	mean	SD	mean	SD	opt found(of 25)
ILS_I p=10	269.93	119.87	43368.52	5.15	0
p=6	274.94	143.49	43358.31	4.61	0
p=2	243.49	130.54	43347.68	3.06	2
ILSRES p=10	266.56	143.88	43371.56	5.96	0
p=6	288.66	113.36	43359.26	4.77	0
p=2	224.71	146.34	43347.28	2.03	1
POPILS s=20	368.81	74.47	43348.88	2.02	0
POPREP $\mu = 1$	376.04	113.31	43348.92	1.7	0
POPSEL μ=10 ψ=0	336.29	90.67	43345.76	1.33	3
μ=10 ψ=0.2	371.69	97.88	43346.52	1.82	4
μ=10 ψ=0.4	342.63	116.47	43346.16	1.86	7
μ=5 ψ=0.2	331.51	108.6	43346.36	2.03	4
μ=100 ψ=0.2	351.62	118.64	43347.28	1.33	0
POPSTG ν=10 ψ=0.2	391.62	62.82	43346.29	1.38	2 of 20
ν=10 ψ=0.4	341.11	102.26	43345.8	1.22	4
ν=100 ψ=0.2	345.59	96.25	43347.64	1.43	0
ν=500 ψ=0.2	358.77	102.79	43348.16	1.74	1

Table 2. Computational results measured across 25 trials of each SLS algorithm on instance `bk151` (best-known solution 43344).

6 Conclusions And Future Work

In this paper we have considered the application of single trajectory ILS, population-based ILS algorithms, and memetic algorithms to the MPTTP, a very difficult scheduling problem arising in multi-processor environments. We have evaluated the algorithms on a large set of MPTTP benchmark instances, and focused our presentation of the results on some of the hardest instances we have encountered in our experiments. The main conclusion is that on these hardest instances, population-based ILS algorithms can offer advantages over single trajectory ILS methods. Regarding the addition of recombination, we observed for single instances that the frequency of finding the best-known solutions for our instances increased slightly. However, only a minor effect on the average solution quality was observed.

The population based framework is generally applicable to all scheduling problems with regular objectives. In order to take advantage of the recombination operator introduced in this work, additional conditions need to be satisfied. It must be possible to translate candidate solutions to and from a priority list such that their objective function values are non-increasing. This implies conditions not only on the objective function, but also on the processor and job constraints. Furthermore, the machines must be identical and the processing time of a job must generally be independent of the schedule.

There are several ways for extending this research. One possibility is the use of local search algorithms based on variable neighbourhood descent (VND). For example, a VND local search was essential for the excellent performance of an ILS algorithm for the SPTWTP, the weighted version of the SPTTP [4]. However, it should be noted

Instance	Algorithm	Time mean	Solution best	median	worst	successes (%)
bk101	ILSRES	263.7	28127	28139.1	28195	10
TF=0.6,RDD=0.6	POPILS $\mu = 0$	430.9	28127	28128.5	28134	30
best=28127	POPILS $\mu = 0.4$	405.2	28127	28129.1	28135	30
bk131	ILSRES	0.8784	284	284	284	100
TF=0.8,RDD=0.4	POPILS $\mu = 0$	1.0348	284	284	284	100
best=284	POPILS $\mu = 0.4$	1.01	284	284	284	100
bk181	ILSRES	334.6	14159	14164.3	14177	0
TF=1.0,RDD=0.6	POPILS $\mu = 0$	452.8	14168	14176.4	14182	0
best=14160	POPILS $\mu = 0.4$	413.8	14163	14171.8	14180	0

Table 3. Results for application of various ILS algorithms to other problem instances.

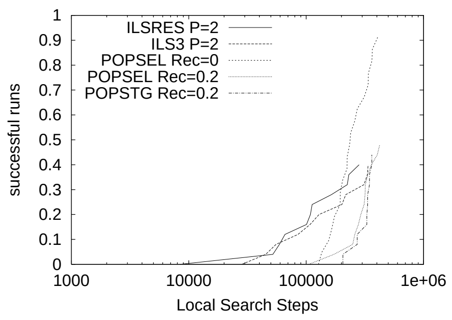

Fig. 9. RTDs of population based and single trajectory ILS algorithms on instance `bk151`.

that we are already using local search in two different neighbourhoods; however, we could imagine other ways of combining the neighbourhood searches. Another possibility is the usage of different acceptance criteria in the single trajectory ILS, which was shown to be essential for the performance of ILS algorithms for the TSP [24]; however, for the SPTWTP and the SPTTP, the Better acceptance criterion gave overall best performance [4]. A further possible extension is to consider priority levels of the jobs by assigning them different weights. In the single processor case, this is known to strongly increase the difficulty of the tardiness problem and the same is to be expected for the multi-processor case. Our algorithms can be extended to this generalised tardiness problem in a straightforward way.

References

1. Du, J., Leung, J.Y.T.: Minimizing total tardiness on one machine is NP-hard. Mathematics of Operations Research **15** (1990) 483–495
2. Koulamas, C.: The total tardiness problem: Review and extensions. Operations Research **42** (1994) 1025–1041

3. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness. Freeman, San Francisco, CA, USA (1979)
4. den Besten, M., Stützle, T., Dorigo, M.: Configuration of iterated local search: An example application to the single machine total weighted tardiness problem. In: Applications of Evolutionary Computing. Volume 2037 of Lecture Notes in Computer Science., Springer Verlag, Berlin, Germany (2001) 441–451
5. Congram, R.K., Potts, C.N., de Velde, S.L.V.: An iterated dynasearch algorithm for the single–machine total weighted tardiness scheduling problem. INFORMS Journal on Computing **14** (2002) 52–67
6. Moscato, P.: On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report 790, Caltech Concurrent Comp. Program (1989)
7. Croce, F.D., Tadei, R., Baracco, P., Grosso, A.: A new decomposition approach for the single machine total tardiness scheduling problem. Journal of the Operational Research Society **49** (1998) 1101–1106
8. Tansel, B., Kara, B., Sabuncuoglu: Single machine total tardiness scheduling problem. IIE Transactions **33** (2001) 661–674
9. Emmons, H.: One-machine sequencing to minimize certain functions of job tardiness. Operations Research **17** (1969) 701–715
10. Azizoglu, M., Kirca, O.: Tardiness minimization on parallel machines. International Journal of Production Economics **55** (1998) 163–168
11. Sevaux, M., Thomin, P.: Heuristics and metaheuristics for a parallel machine scheduling problem: a computational evaluation. Technical Report 01-1-SP, University of Valenciennes (2001)
12. Davidovic, T., Hansen, P., Mladenovic, N.: Variable neighborhood search for multiprocessor scheduling with communication delays. (2001)
13. den Besten, M.L.: Ants for the single machine total weighted tardiness scheduling problem. Master's thesis, Universiteit van Amsterdam (2000)
14. Sivrikaya-Serifoglu, F., Ulusoy, G.: Parallel machine scheduling with earliness and tardiness penalties. Computers and Operations Research **26** (1999) 773–787
15. Frana, P.M., Gendreau, M., Laporte, G., Muller, F.M.: A tabu search heuristic for the multiprocessor scheduling problem with sequence dependent setup times. International Journal of Production Economics **43** (1996) 79–89
16. Frana, P.M., Mendes, A., Moscato, P.: A memetic algorithm for the total tardiness total tardiness single machine scheduling problem. European Journal of Operational Research **132** (2001) 224–242
17. Lourenço, H.R., Martin, O., Stützle, T.: Iterated local search. In Glover, F., Kochenberger, G., eds.: Handbook of Metaheuristics. Volume 57 of International Series in Operations Research & Management Science. Kluwer Academic Publishers, Norwell, MA (2002) 321–353
18. Bäck, T.: Evolutionary Algorithms in Theory and Practice. Oxford University Press, New York, NY (1996)
19. Mitchell, M.: An Introduction to Genetic Algorithms. MIT Press, Cambridge, MA (1996)
20. Moscato, P.: Memetic algorithms: A short introduction. In Corne, D., Dorigo, M., Glover, F., eds.: New Ideas in Optimization. McGraw Hill, London, UK (1999) 219–234
21. Hong, I., Kahng, A.B., Moon, B.R.: Improved large-step Markov chain variants for the symmetric TSP. Journal of Heuristics **3** (1997) 63–81
22. Stützle, T.: Local Search Algorithms for Combinatorial Problems – Analysis, Improvements, and New Applications. PhD thesis, Darmstadt University of Technology (1998)
23. Kara, B.: SMTTP problem library, http://www.bilkent.edu.tr/ bkara/start.html (2002)
24. Stützle, T., Hoos, H.H.: Analysing the run-time behaviour of iterated local search for the travelling salesman problem. In Hansen, P., Ribeiro, C., eds.: Essays and Surveys on Metaheuristics. Kluwer Academic Publishers (2001) 589–611