Design of Iterated Local Search Algorithms

An Example Application to the Single Machine Total Weighted Tardiness Problem

Matthijs den Besten[1], Thomas Stützle[1] and Marco Dorigo[2]

[1] Darmstadt University of Technology, Intellectics Group, Alexanderstr. 10, 64283 Darmstadt, Germany
[2] Université Libre de Bruxelles, IRIDIA, Avenue Franklin Roosevelt 50, CP 194/6, 1050 Brussels, Belgium

Abstract. In this article we investigate the application of iterated local search (ILS) to the single machine total weighted tardiness problem. Our research is inspired by the recently proposed iterated dynasearch approach, which was shown to be a very effective ILS algorithm for this problem. In this paper we systematically configure an ILS algorithms by optimizing the single procedures part of ILS and optimizing their interaction. We come up with a highly effective ILS approach, which outperforms our implementation of the iterated dynasearch algorithm on the hardest benchmark instances.

1 Introduction

In the single machine total weighted tardiness problem (SMTWTP) n jobs have to be sequentially processed on a single machine. Each job j has a processing time p_j, a weight w_j, and a due date d_j associated, and the jobs become available for processing at time zero. The tardiness of a job j is defined as $T_j = \max\{0, C_j - d_j\}$, where C_j is the completion time of job j in the current job sequence. The goal is to find a job sequence which minimizes the sum of the weighted tardiness given by $\sum_{i=1}^{n} w_i \cdot T_i$.

The SMTWTP is an $\mathcal{NP}$-hard [9] scheduling problem and instances with more than 50 jobs can often not be solved to optimality with state-of-the-art branch & bound algorithms [1, 4]. Therefore, several heuristic methods have been proposed for its solution. These include simple construction heuristics like the Earliest Due Date or the Apparent Urgency heuristics (see [17] for an overview) and metaheuristics like simulated annealing [14, 17], tabu search [4], genetic algorithms [4], ant colony optimization (ACO) [5, 15], and iterated local search (ILS) [3].

ILS appears to be a very promising approach for solving the SMTWTP, because the ILS algorithm by Congram, Potts, and de Velde, called *iterated dynasearch* [3], has shown so far, together with the recent ACO algorithm due to den Besten, Stützle, and Dorigo[5], the best performance results for the SMTWTP. Despite the very good performance of iterated dynasearch, it is not very clear, whether this ILS algorithm has been designed in a best possible way. Therefore, this paper examines the systematic, experimentally driven configuration of an ILS algorithm for the SMTWTP. In particular, we first optimize the single ILS components and derive in this way a highly effective algorithm. This step-by-step methodology can also be used as a guideline for the development of ILS algorithms for other combinatorial optimization problems.

Algorithm 1 Algorithmic outline of iterated local search.

s_0 = GenerateInitialSolution
s^* = LocalSearch(s_0)
repeat
  s' = Perturbation(s^*, *history*)
  $s^{*\prime}$ = LocalSearch(s')
  s^* = AcceptanceCriterion(s^*, $s^{*\prime}$, *history*)
until termination criterion met

The paper is structured as follows. Section 2 introduces ILS and Section 3 studies the influence of the single procedures which are part of an ILS algorithm on its performance and gives experimental results with the final ILS algorithm. We end with some concluding remarks in Section 4.

2 Iterated local search

The underlying idea of ILS [2, 13, 12] is that of building a random walk in $\mathcal{S}^*$, the space of local optima defined by the output of a given local search algorithm. Four basic "ingredients" are needed to derive an ILS algorithm: a procedure GenerateInitialSolution, which returns some initial solution, a local search procedure LocalSearch, a scheme of how to perturb a solution, implemented by a procedure Perturbation, and an AcceptanceCriterion, which decides from which solution the search is continued. An algorithmic outline for ILS is given in Algorithm 1. The particular walk in $\mathcal{S}^*$ followed by the ILS algorithm can also be depend on the search history, which is indicated by *history* in Perturbation and AcceptanceCriterion.

The effectiveness of the walk in $\mathcal{S}^*$ depends on the definition the four component procedures of ILS: The effectiveness of the local search algorithm is of major importance, because it strongly influences the final solution quality of ILS and its overall computation time. The perturbations should allow the ILS to effectively escape local optima but at the same time avoid the disadvantages of random restart (hence, not be too strong). The acceptance criterion, together with the perturbation, strongly influence the type of walk in $\mathcal{S}^*$ and can be used to control the balance between intensification and diversification of the search. The initial solution will be mainly be important in the initial part of the search.

The configuration problem in ILS is to find a best possible choice for the four components such that best overall performance is achieved. Because of the interactions among the components, this is a difficult problem and it has to be solved in a heuristic way. Here, we do this by considering at each step only the influence of one single component, keeping the others at some fixed, "reasonable" choices. These are that (i) as the initial solution we use the best construction heuristic, (ii) the acceptance criterion forces the cost to decrease (this means the perturbation is always applied to the best solution found so far), and (iii) the perturbation uses a number of random moves in a given neighborhood. First, we will optimize the choice of LocalSearch by investigating different local search algorithms. Once found a good local search, we reconsider the choices for the solution perturbation and the acceptance criterion in that order.

3 Iterated local search for the SMTWTP

3.1 Local search

Local search for the SMTWTP starts from some initial sequence and repeatedly tries to improve the current sequence by replacing it with neighboring solutions. The simplest local search algorithm, iterative descent, repeatedly replaces the current sequence π with a better sequence found in the neighborhood of π and stops at the first local minimum encountered. For the SMTWTP we considered the following two neighborhood structures:

(1) exchanges of jobs placed at the ith and the jth position, $i \neq j$ (*interchange*)
(2) removal of the job at the ith position and insertion in the jth position (*insert*)

To allow for a fast evaluation of moves in these neighborhoods, the data structures proposed in [3] were implemented. The neighborhood structure is critical for the performance of the local search. Often, with more complex neighborhoods than the two presented above better solutions may be found. An example of a more complex neighborhood is the one used in dynasearch [3]. Dynasearch uses dynamic programming to find a best move which is composed of a set of independent interchange moves; each such move exchanges the jobs at positions i and j, $j \neq i$. Two interchange moves are independent if they do not overlap, that is if for two moves involving positions i, j and k, l we have that $\min\{i, j\} \geq \max\{k, l\}$ or vice versa. This neighborhood is of exponential size and dynasearch explores this neighborhood in polynomial time, to be more exact in $\mathcal{O}(n^3)$. In [3] very good performance with dynasearch has been reported.

To achieve further improvements of the solution quality, we considered the application of a variable neighborhood descent (VND) [16]. In our VND we concatenate iterative descent algorithms using two different neighborhoods; such an approach was also proposed in [18] for the permutation flow shop problem. VND exploits the observation that a local optimum with respect to one neighborhood structure need not be a local optimum for the other one. In fact, the variable neighborhood search (VNS) metaheuristic [16] systematically applies the idea of changing neighborhoods in the search. There are two possible ways of concatenating the two neighborhoods; these will be denoted in the following as *interchange+insert* and *insert+interchange*, depending on which neighborhood is searched first. Additionally, we also considered replacing the interchange local search algorithm with dynasearch, yielding two more variants, namely *dyna+insert* and *insert+dyna*.

We evaluated the proposed local search algorithms using a benchmark set of randomly generated instances, available via ORLIB at `http://www.ms.ic.ac.uk/-info.html`. There are three sets of instances with 40, 50, and 100 jobs. While for the 40 and 50 job instances the optimal solutions are known, the 100 job instances are still unsolved and only the best known solutions are available. The instances are generated by drawing the processing time p_j for each job j randomly according to a uniform distribution of integers between 1 and 100 and assigning it a weight w_j randomly drawn from a uniform distribution over the integers between 1 and 10. The due dates are randomly drawn integers from the interval $[(1 - \mathrm{TF} - \mathrm{RDD}/2) \cdot \sum p_i, (1 - \mathrm{TF} + \mathrm{RDD}/2) \cdot \sum p_i]$, where TF, the tardiness factor, and RDD, the relative due date,

Table 1. Comparison of the local search effectiveness for the SMTWTP. Results on the 100 job instances without local search and using the interchange, the insert, and the VND variants. We give the average percentage deviation from the best known solutions (Δ_{avg}), the number of best-known solutions found (n_{opt}), and the average CPU time in seconds (t_{avg}) averaged over the 125 benchmark instances.

start	no local search			insert			interchange			dyna interchange		
	Δ_{avg}	n_{opt}	t_{avg}	Δ_{avg}	n_{opt}	t_{avg}	Δ_{avg}	n_{opt}	t_{avg}	Δ_{avg}	n_{opt}	t_{avg}
EDD	135	24	0.0004	1.19	38	0.29	2.09	26	0.23	1.25	26	0.41
MDD	62	24	0.0007	1.31	36	0.32	1.03	33	0.16	1.02	33	0.27
AU	62	20	0.0018	0.56	39	0.11	0.81	33	0.05	0.90	30	0.12

start	dynasearch+insert			insert + interchange			interchange+insert			insert+dynasearch		
	Δ_{avg}	n_{opt}	t_{avg}	Δ_{avg}	n_{opt}	t_{avg}	Δ_{avg}	n_{opt}	t_{avg}	Δ_{avg}	n_{opt}	t_{avg}
EDD	0.30	45	0.46	0.46	49	0.30	0.52	42	0.28	0.46	50	0.34
MDD	0.39	46	0.33	0.42	42	0.33	0.37	44	0.20	0.42	42	0.40
AU	0.67	46	0.16	0.34	48	0.12	0.63	50	0.08	0.34	48	0.15

are two parameters. There are five instances for each pair of TF and RDD from the set $\{0.2, 0.4, 0.6, 0.8, 1.0\}$. This makes three sets of 125 instances each. The tardiness factor and the relative due dates determine critically the difficulty of solving the instances. For example, we found that most of the instances with TF $= 0.2$ are solved after one single application of the best local search procedures, while for larger TF, the instances were much harder to solve. All the experiments were run on a 700MHz Pentium III CPU with 512 MB RAM. Programs were written in C++ and run under Suse Linux 6.1.

The computational results are given in Table 1 for three different construction heuristics and all the described local search variants. Some of the benchmark instances are very easily solved, as indicated by the large number of best-known solutions found by the local search algorithms alone. In general, the best performance is obtained by the VND local search algorithms, which yield significantly better solution quality at only a slight increase in computation time compared to the single neighborhood local search.[1] The dynasearch local search seems not to give significantly better solution quality than the interchange algorithm.[2]

Based on the results reported in Table 1, we run one trial on each instance for 10 seconds with the basic ILS algorithm using the VND variants for the local search and then measured the final solution quality and the number of best known solutions found. Before discussing the results, let us identify the perturbation applied in the ILS algorithm: We used either six random interchange moves or six random insert moves, depending on which neighborhood is used in the next local search. The idea is that the perturbation should be complementary to the particular local search and it should be difficult for the

[1] The single results are slightly different to those published in [5], because of minor changes in the local search implementation.

[2] This fact has also noted in [3], where it was argued that the main advantage with dynasearch comes from a repetitive application of dynasearch in an ILS.

Table 2. Comparison of ILS algorithms using different choices for local search. Results on the 100 job instances with one trial per instance of 10 secs. We give the average percentage deviation from the best known solutions (Δ_{avg}), the number of best-known solutions found (n_{opt}), and the average CPU time in seconds to find the best solution in a trial (t_{avg}) averaged over the 125 benchmark instances.

start	ILS-inter+insert			ILS-dyna+insert			ILS-insert+inter			ILS-insert+dyna			ILS-dynasearch		
	Δ_{avg}	n_{opt}	t_{avg}	Δ_{avg}	n_{opt}	t_{avg}	Δ_{avg}	n_{opt}	t_{avg}	Δ_{avg}	n_{opt}	t_{avg}	Δ_{avg}	n_{opt}	t_{avg}
AU	0.0006	116	1.95	0.0019	109	2.62	0.0057	107	2.79	0.0046	107	2.97	0.0848	79	5.02

local search to undo the perturbation. For example, when applying ILS using the *interchange+insert* local search, we use random insert moves for the perturbation, because they are complementary to the following interchange local search.

The results of these latter experiments are reported in Table 2. In general, VNDs *interchange+insert* and *dyna+insert* appear to perform best when used inside an ILS algorithm. The *insert+interchange* VND performs only slightly worse; significantly worse results are obtained with the ILS-dynasearch. The results with ILS-dynasearch obtained with our re-implementation of dynasearch also appear to be worse than those presented in [3], especially when taking into account computation time (the experiments in [3] were run on a much slower computer). We verified that our dynasearch implementation works properly from a solution quality point of view, but it appears to be slower than the interchange local search, while in [3] interchange was slower than dynasearch. Therefore, we conjecture, that in particular the dynamic programming algorithm used in dynasearch to examine the neighborhood could still be speed up and our results with dynasearch should be taken as preliminary. Additionally, the use of C++ and some of its features like inheritance and templates may make our code significantly slower.

In a second experiment we analyzed the ILS run-time behavior by using run-time distributions (RTDs). RTDs give the cumulative empirically observed probability of finding an optimal solution (or a solution within a specific solution quality bound) as a function of the CPU time [7, 20]. Here, for each instance 25 runs have been performed. In total we examined the run-time behavior of ILS algorithms with different choices for LocalSearch on 10 instances which were known to be relatively hard; in particular they are not solved by applying one single local search. Figure 1 presents only results for four instances, the behavior on the others was similar. The RTDs show, that no single local search algorithm gives the best behavior on all instances. The best performance is obtained when applying the *interchange+insert* and *dyna+insert* VND (an exception is, for example, instance 42); the ILS with dynasearch is significantly worse for most instances. Because our interchange implementation is faster than our dynasearch implementation, we use the *interchange+insert* VND for the following experiments.

3.2 Perturbation

Once fixed the choice for the local search, we closer examined the role of the solution perturbation. We addressed two important issues:

Perturbation strength: We will refer to the *strength* of a perturbation as the number of solution components which are modified. In the SMTWTP this is the number

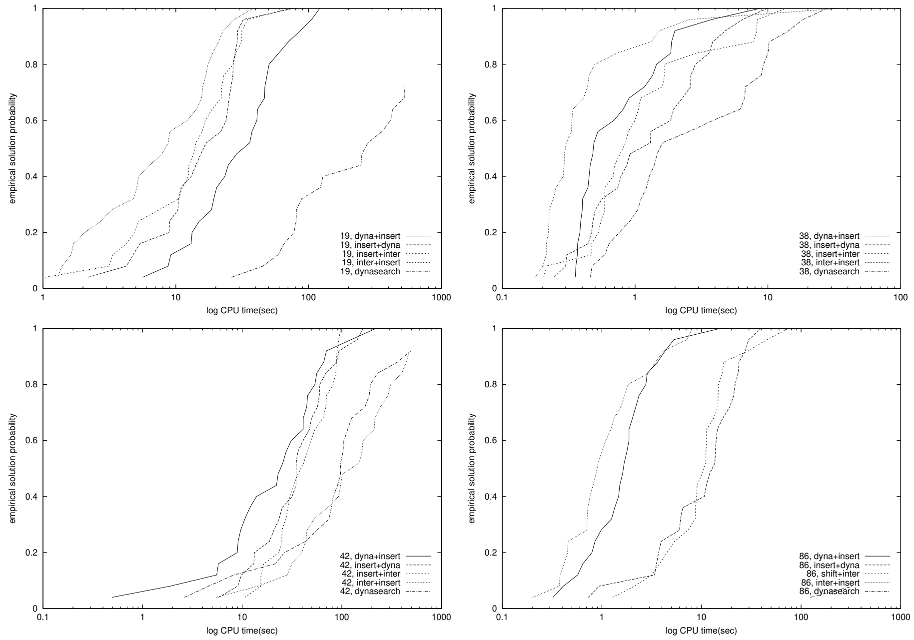

Fig. 1. We compare different choices for the local search with RTDs. The x-axis gives the logarithm of the CPU time, the y-axis the cumulative empirical solution probability (the more to the left a curve is located, the better performs the algorithm). RTDs are given for the five local searches tested in Table 2. The number in the caption gives the instances number; for example instance 42 is the 42nd instance of the 125 available one from ORLIB.

of jobs directly affected by a perturbation. Different choices for the perturbation strength, from three to twelve in steps of three, were examined.

Nature of perturbations: As said before, the perturbations should be complementary to the local search. Here, in addition we examined a variant, in which the random perturbations were required to involve only late jobs.

Again, we examined the different choices for the perturbation strength using RTDs, which are given in Figure 2 (results with perturbation focusing on late jobs are not given to keep the figures as clear as possible). As a first result we can observe that no single perturbation strength is best among all instances. Additionally, we found that a focus on late jobs in the perturbation does not significantly improve performance. Therefore, we settled for the following choice of the perturbation: (i) we do not focus on late jobs, because a simpler choice gave similar performance and (ii) we varied the perturbation strength in a range from three to twelve random moves. This latter variation was done in a scheme analogously to that introduced in the basic VNS HanMla99:mic.

3.3 Acceptance criterion

A natural choice for the acceptance criterion is to force the cost to decrease by accepting an $s^{*\prime}$ if its cost is less than that of s^* (we refer to this acceptance criterion as `Better`

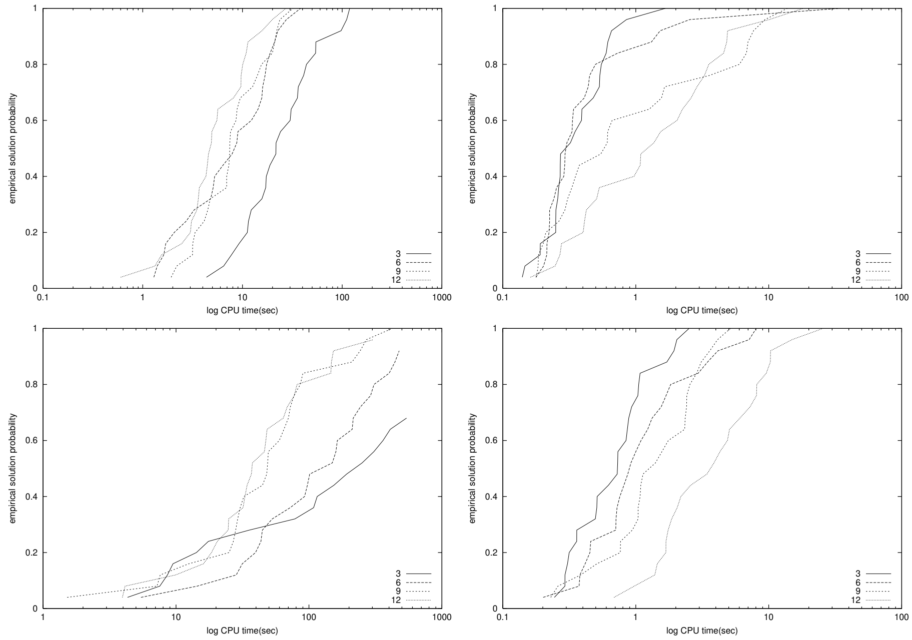

Fig. 2. We compare different choices for the perturbation with RTDs. The x-axis gives the logarithm of the CPU time, the y-axis the cumulative empirical solution probability. RTDs are given for the four different perturbation strengths for pure random perturbations. Results are given for instances 19 (top left), 38 (top right), 42 (bottom left) and 86 (bottom right).

in the following). Such a choice leads to a very strong intensification of the search and may lead to bad behavior for long run-times, when diversification of the search becomes more important. Diversification of the search is extremely favored if every $s^{*\prime}$ is accepted as the new solution, resulting in a random walk in $\mathcal{S}^*$. We call this acceptance criterion `RW`. In [3], a `Backtrack` acceptance criterion is proposed, which is a combination of the `Better` and the `RW`: For β iterations `RW` is used. If no improved solution is found, the ILS continues again from the best solution seen so far.

The results of the RTD-based analysis of the acceptance criteria with RTDs is plotted in Figure 3. We found that with respect to the acceptance criteria the results were somewhat clearer compared to the findings on the other two components: For almost all instances ILS with the `Better` acceptance criterion showed best behavior. Hence, this acceptance criterion was also chosen for our final ILS algorithm.

3.4 Experimental results

The final ILS algorithm has the following shape: (i) it uses the AU construction heuristic to generate the initial solution, (ii) it uses the *interchange+insert* VND local search, (iii) it varies the perturbation strength between three to twelve random insert moves, and (iv) it accepts only better solutions in the random walk in $\mathcal{S}^*$. We conducted some experiments with this ILS algorithm on all 40, 50, and 100 job SMTWTP instances available

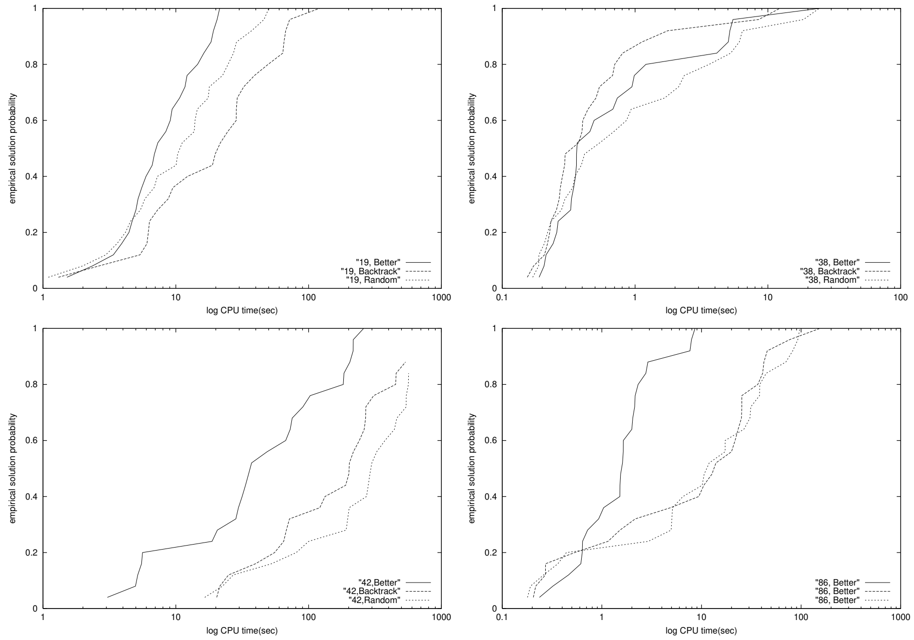

Fig. 3. We compare different acceptance criteria with RTDs. The x-axis gives the logarithm of the CPU time, the y-axis the cumulative empirical solution probability. RTDs are given for the three different acceptance criteria. The number in the caption gives the instances number.

from ORLIB. On each instance 25 trials were performed with a large computation time limit, which was enough that each instance could be solved to the best-known solutions, which we conjecture to be optimal, in each single trial. Of the 125 instances with 100 jobs (those with 40 or 50 jobs are very easily solved, see Table 3), only 15 took an average time to optimal larger than 10 seconds; only 7 of these longer than 20 seconds. The large majority of the benchmark instances was either solved with a single local search or within very few seconds.

Our ILS algorithm also compares very favourably to our earlier ACO algorithm presented in [5]. This shows that ILS may be an easily adaptable alternative to other, often more complex metaheuristics, showing an excellent performance after some straightforward optimizations. A more detailed investigation of different metaheuristics applied to the SMTWTP and a detailed search space analysis of the SMTWTP are the next steps we will take.

4 Conclusion

The results of this research can be summarized as follows:

1. The independent optimization of the single components of an ILS algorithm for the SMTWTP has led to a high performing ILS algorithm for the SMTWTP.

Table 3. We give some basic statistics on the distribution of the computation times to solve instances of the three problem sets. We indicate the number of jobs (n), the average time (averaged over the 125 instances) to solve the benchmark set (t_{avg}) and its standard deviation (σ_t), the average time to solve the easiest and the hardest instance (t_{min} and t_{max}, respectively), and the quantils of the average time to solve a given percentage of the instances. Q_x indicates the average time to solve $x\%$ of the benchmark instances.

n	t_{avg}	σ_t	t_{min}	t_{max}	Q_{25}	Q_{50}	Q_{75}	Q_{90}
100	5.75	14.50	0.0076	105.50	0.052	0.98	5.14	13.12
50	0.20	0.86	0.0017	8.71	0.0064	0.018	0.118	0.28
40	0.040	0.13	0.0011	1.23	0.0031	0.0072	0.033	0.062

2. VND [16] leads to a very powerful local search for the SMTWTP.
3. For the SMTWTP the optimization of the ILS algorithm leads to improved performance. Yet, the improvement due to these optimizations is not as spectacular as observed for other problems [6, 19, 21], possibly due to the powerful local search.
4. The SMTWTP instances from ORLIB do not pose a challenge for state-of-the-art algorithms.

Clearly, these conclusions do raise further questions. The optimization of ILS for the SMTWTP was certainly rather straightforward and heuristic. We conjecture that for harder problems, this process will be much more important: it will need more iterations through the choices for the single components and statistical methods of experimental design will become more important. The excellent performance of VND for the SMTWTP naturally lends to the question whether VND can improve the efficiency of local search also for other scheduling problems. Preliminary results suggest, that the answer strongly depends on the particular problem. For example, in [18] encouraging results have been reported with a VND for the flow shop problem, but some experiments with an ILS algorithm [19] suggest that this improvement does not carry over to a significant improvement of ILS.

Future work will include tests of the ILS algorithms on larger instances, and an extension of our approach to other single machine scheduling problems. The results of this paper and the very good performance of a variety of ILS algorithms on several classes of scheduling problems [3, 8, 11, 10, 19] suggest that ILS is a very appropriate metaheuristic to obtained very high quality solutions in scheduling applications.

Acknowledgments Marco Dorigo acknowledges support from the Belgian FNRS, of which he is a Senior Research Associate. This work was partially supported by the "Metaheuristics Network", a Research Training Network funded by the Improving Human Potential programme of the CEC, grant HPRN-CT-1999-00106. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

References

1. T. S. Abdul-Razaq, C. N. Potts, and L. N. Van Wassenhove. A survey of algorithms for the single machine total weighted tardiness scheduling problem. *Discrete Applied Mathematics*,

26:235–253, 1990.
2. E. B. Baum. Towards practical “neural” computation for combinatorial optimization problems. In J. Denker, editor, *Neural Networks for Computing*, pages 53–64, 1986. AIP conference proceedings.
3. R. K. Congram, C. N. Potts, and S. L. Van de Velde. An iterated dynasearch algorithm for the single–machine total weighted tardiness scheduling problem. INFORMS Journal on Computing, to appear, 2000.
4. H. A. J. Crauwels, C. N. Potts, and L. N. Van Wassenhove. Local search heuristics for the single machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, 10(3):341–350, 1998.
5. M. den Besten, T. Stützle, and M. Dorigo. Ant colony optimization for the total weighted tardiness problem. In M. Schoenauer et al., editor, *Proceedings of PPSN-VI*, volume 1917 of *LNCS*, pages 611–620. Springer Verlag, Berlin, Germany, 2000.
6. I. Hong, A. B. Kahng, and B. R. Moon. Improved large-step Markov chain variants for the symmetric TSP. *Journal of Heuristics*, 3(1):63–81, 1997.
7. H. H. Hoos and T. Stützle. Evaluating Las Vegas algorithms — pitfalls and remedies. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence 1998*, pages 238–245. Morgan Kaufmann Publishers, 1998.
8. S. Kreipl. A large step random walk for minimizing total weighted tardiness in a job shop. *Journal of Scheduling*, 3(3):125–138, 2000.
9. J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problem. In P. L. Hammer et al., editor, *Studies in Integer Programming*, volume 1 of *Annals of Discrete Mathematics*, pages 343–362. North-Holland, Amsterdam, NL, 1977.
10. H. R. Lourenço. Job-shop scheduling: Computational study of local search and large-step optimization methods. *European Journal of Operational Research*, 83:347–364, 1995.
11. H. R. Lourenço and M. Zwijnenburg. Combining the large-step optimization with tabu-search: Application to the job-shop scheduling problem. In I.H. Osman and J.P. Kelly, editors, *Meta-Heuristics: Theory & Applications*, pages 219–236. Kluwer, 1996.
12. O. Martin and S.W. Otto. Combining Simulated Annealing with Local Search Heuristics. *Annals of Operations Research*, 63:57–75, 1996.
13. O. Martin, S.W. Otto, and E.W. Felten. Large-Step Markov Chains for the Traveling Salesman Problem. *Complex Systems*, 5(3):299–326, 1991.
14. H. Matsuo, C. J. Suh, and R. S. Sullivan. A controlled search simulated annealing method for the single machine weighted tardiness problem. Working paper 87-12-2, Department of Management, University of Texas at Austin, TX, 1987.
15. D. Merkle and M. Middendorf. An ant algorithm with a new pheromone evaluation rule for total tardiness problems. In S. Cagnoni et al., editor, *Proceedings of EvoWorkshops 2000*, volume 1803 of *LNCS*, pages 287–296. Springer Verlag, Berlin, Germany, 2000.
16. N. Mladenović and P. Hansen. Variable Neighborhood Search. *Computers & Operations Research*, 24:1097–1100, 1997.
17. C. N. Potts and L. N. Van Wassenhove. Single machine tardiness sequencing heuristics. *IIE Transactions*, 23:346–354, 1991.
18. C. R. Reeves. Landscapes, operators and heuristic search. To appear in *Annals of Operations Research*, 2000.
19. T. Stützle. Applying iterated local search to the permutation flow shop problem. Technical Report AIDA–98–04, FG Intellektik, TU Darmstadt, August 1998.
20. T. Stützle. *Local Search Algorithms for Combinatorial Problems — Analysis, Improvements, and New Applications*. PhD thesis, Darmstadt University of Technology, Department of Computer Science, 1998.
21. Thomas Stützle. Iterated local search for the quadratic assignment problem. Technical Report AIDA–99–03, FG Intellektik, FB Informatik, TU Darmstadt, March 1999.