Metaheuristics for Group Shop Scheduling

Michael Sampels[1], Christian Blum[1], Monaldo Mastrolilli[2], Olivia Rossi-Doria[3]

[1] IRIDIA, Université Libre de Bruxelles, CP 194/6,
Av. Franklin D. Roosevelt 50, 1050 Bruxelles, Belgium
{msampels|cblum}@ulb.ac.be

[2] IDSIA, Galleria 2, 6928 Manno, Switzerland
monaldo@idsia.ch

[3] School of Computing, Napier University,
10 Colinton Road, Edinburgh, EH10 5DT, Scotland
o.rossi-doria@napier.ac.uk

Abstract. The Group Shop Scheduling Problem (GSP) is a generalization of the classical Job Shop and Open Shop Scheduling Problems. In the GSP there are m machines and n jobs. Each job consists of a set of operations, which must be processed on specified machines without preemption. The operations of each job are partitioned into groups on which a total precedence order is given. The problem is to order the operations on the machines and on the groups such that the maximal completion time (makespan) of all operations is minimized. The main goal of this paper is to provide a fair comparison of five metaheuristic approaches (i.e., Ant Colony Optimization, Evolutionary Algorithm, Iterated Local Search, Simulated Annealing, and Tabu Search) to tackle the GSP. We guarantee a fair comparison by a common definition of neighborhood in the search space, by using the same data structure, programming language and compiler, and by running the algorithms on the same hardware.

1 Introduction to the Group Shop Scheduling Problem

A general scheduling problem can be formalized as follows: We consider a finite set of operations O, partitioned into m subsets $\langle M_1, \ldots, M_m \rangle =: \mathcal{M}$ $(\dot{\bigcup}_{i=1}^{m} M_i = O)$ and into n subsets $\langle J_1, \ldots, J_n \rangle =: \mathcal{J}$ $(\dot{\bigcup}_{k=1}^{n} J_k = O)$, together with a partial order $\preceq \subseteq O \times O$ such that $\preceq \cap J_i \times J_j = \emptyset$ for $i \neq j$, and a function $p : O \to \mathbf{N}$. A feasible solution is a refined partial order $\preceq^* \supseteq \preceq$ for which the restrictions $\preceq^* \cap M_i \times M_i$ and $\preceq^* \cap J_k \times J_k$ are total $\forall\, i, k$. The cost of a feasible solution is defined by

$$C_{\max}(\preceq^*) := \max\{\sum_{o \in C} p(o) \mid C \text{ is a chain in } (O, \preceq^*)\} \ .$$

We aim at a feasible solution which minimizes $C_{\max}$.

M_i is the set of operations that have to be processed on machine i. J_k is the set of operations that belong to job k. Each machine can process at most one operation at a time. Operations must be processed without preemption. Operations belonging to the same job must be processed sequentially. This is expressed in the constraints for $\preceq^*$. $C_{\max}$ is the makespan of the schedule defined by $\preceq^*$.

This brief problem formulation covers well known scheduling problems: The restriction $\preceq \cap J_i \times J_i$ is total in the Job Shop Scheduling Problem (JSP), trivial ($= \{(o,o) \mid o \in J_i\}$) in the Open Shop Scheduling Problem (OSP), and either total or trivial for each i in the Mixed Shop Problem.

In this paper, we consider a weaker restriction on $\preceq$ which includes the above scheduling problems by looking at a refinement of the partition $\mathcal{J}$ to a partition into *groups* $\langle G_1, \ldots, G_g \rangle =: \mathcal{G}$. We demand that $\preceq \cap G_i \times G_i$ has to be trivial and that for $o, o' \in J$ ($J \in \mathcal{J}$) with $o \in G_i$ and $o' \in G_j$ ($i \neq j$) either $o \preceq o'$ or $o \succeq o'$ holds. We call this problem Group Shop Scheduling Problem (GSP).Note that the coarsest refinement $\mathcal{G} = \mathcal{J}$ (groups sizes are equal to job sizes) is equivalent to the OSP and the finest refinement $\mathcal{G} = \{\{o\} \mid o \in O\}$ (group sizes of 1) is equivalent to the JSP. In the following, for $G \in \mathcal{G}$ we denote $o \in G$ by $g(o) = G$; for $M \in \mathcal{M}$ we denote $o \in M$ by $m(o) = M$. An example for a GSP instance is given in Fig. 1.

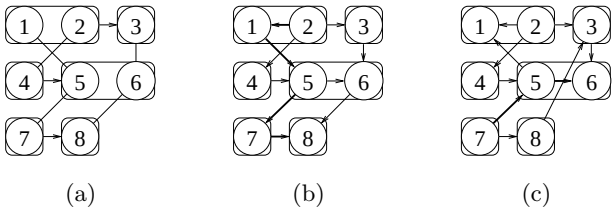

Fig. 1. (a) An example for a GSP instance on 8 operations: $O = \{1, \ldots, 8\}$, $\mathcal{J} = \{J_1 = \{1,2,3\}, J_2 = \{4,5,6\}, J_3 = \{7,8\}\}$, $\mathcal{M} = \{M_1 = \{1,5,7\}, M_2 = \{2,4\}, M_3 = \{3,6,8\}\}$, $\mathcal{G} = \{G_1 = \{1,2\}, G_2 = \{3\}, G_3 = \{4\}, G_4 = \{5,6\}, G_5 = \{7\}, G_6 = \{8\}\}$, $G_1 \prec G_2, G_3 \prec G_4, G_5 \prec G_6$, $p(1) = \cdots = p(4) = 1, p(5) = \cdots = p(8) = 2$; (b) a valid solution with $C_{\max} = 8$ on the chain $2 \preceq^* 1 \preceq^* 5 \preceq^* 7 \preceq^* 8$; (c) an optimal solution with $C_{\max} = 6$ on the chain $7 \preceq^* 5 \preceq^* 6$

2 Common Neighborhood and Local Search

For a feasible solution $\preceq^*$ a chain C is called critical path iff $\sum_{o \in C} p(o) = C_{\max}(\preceq^*)$. $\mathcal{M}$ induces on a critical path $o_1 \preceq^* \cdots \preceq^* o_q$ a subdivision into *machine blocks* of consecutive operations belonging to the same machine, as well as $\mathcal{G}$ induces a subdivision into *group blocks* of consecutive operations belonging to the same group. Brucker et al. [6] proved for the JSP that if there is a feasible solution $\preceq^{*\prime}$ with $C_{\max}(\preceq^{*\prime}) < C_{\max}(\preceq^*)$, then there is a machine block $B^i_M = o^i_1 \preceq^* \cdots \preceq^* o^i_{m_i}$ on a critical path C of $\preceq^*$ such that $\exists\, o \in B^i_M, o \neq o^i_1$ with

$o_1^i \succeq^{*\prime} o$ or $\exists\, o \in B_M^i, o \neq o_{m_i}^i$ with $o_{m_i}^i \preceq^{*\prime} o$. For the GSP, we generalize the above result.

Theorem 1. *Let $\preceq^*$ be a feasible solution to a GSP instance. If there is a solution $\preceq^{*\prime}$ with $C_{\max}(\preceq^{*\prime}) < C_{\max}(\preceq^*)$, then there is a machine or a group block $B^i = o_1^i \preceq^* \cdots \preceq^* o_{n_i}^i$ in C such that $\exists\, o \in B^i, o \neq o_1^i$ with $o_1^i \succeq^{*\prime} o$ or $\exists\, o \in B, o \neq o_{n_i}^i \in B$ with $o_{n_i} \preceq^{*\prime} o$.*

For the proof we refer to an extended version of this paper [14]. By this theorem it is reasonable to define the neighborhood of a feasible solution $\preceq^*$ as follows: A feasible solution $\preceq^{*\prime}$ is a neighbor of $\preceq^*$ ($\in N(\preceq^*)$) if in a critical path C of $\preceq^*$ for exactly one machine block or exactly one group block $B = o_1 \preceq^* o_2 \preceq^* \cdots \preceq^* o_{n_k-1} \preceq^* o_{n_k}$ on C the order of o_1 and o_2 or the order of o_{n_k-1} and o_{n_k} is swapped in $\preceq^{*\prime}$. This is an extension of the neighborhood which Nowicki and Smutnicki [13] used in their tabu search for the JSP.

A local search procedure can be defined recursively on the neighborhood structure as follows:

$$ls(\preceq^*) = \begin{cases} \preceq^* \text{ if } C_{\max}(\preceq^{*\prime}) \geq C_{\max}(\preceq^*)\ \forall\ \preceq^{*\prime} \in N(\preceq^*), \\ ls(\preceq^{*\prime\prime}) \text{ with } C_{\max}(\preceq^{*\prime\prime}) \leq C_{\max}(\preceq^{*\prime})\ \forall\ \preceq^{*\prime} \in N(\preceq^*) \text{ otherwise.} \end{cases}$$

3 Metaheuristic Approaches

The OSP is an NP-hard problem (Gonzalez and Sahni [11]). The JSP is an NP-hard problem as well, as was first shown by Lenstra et al. [12]. As the GSP contains both problems, it is NP-hard, too. Metaheuristics have shown to be very successful in constructing good solutions to scheduling problems. Błażewicz et al. [2] gave a survey on exact and approximation algorithms for the JSP. Fang et al. presented a genetic algorithm for the OSP, and Taillard [16] published a Tabu Search approach.

In the following, we describe five metaheuristics for the GSP, and we analyze how they compare to each other. We study their behavior on the range of GSP instances from the JSP to the OSP and focus on the influence of the group size on the quality of the algorithms. We aim at a fair comparison of the metaheuristic concepts. Therefore we use a joint implementation of the problem representation and of the neighborhood structure. We explicitly don't aim at the use of sophisticated methods to tune the algorithms.

3.1 Ant Colony Optimization

Ant Colony Optimization (ACO) is a metaheuristic approach proposed by Dorigo et al. in [8]. The basic ingredient of ACO is the use of a probabilistic solution construction mechanism. The best known technique to construct solutions to scheduling problems is the list scheduler algorithm. To construct a solution, this list scheduler algorithm builds a sequence s of all operations from left to right. In every one of the $|O|$ construction steps, the algorithm probabilistically chooses an operation from a set S_t (where $t = 1, ..., |O|$) of admissible operations.

We use the pheromone model called $\mathsf{PH}_{\mathsf{rel}}$ (proposed by Blum and Sampels in [5]) where pheromone values are assigned to pairs of *related* operations. Two operations $o_i, o_j \in O$ are called *related* if they belong to the same group, or if they have to be processed on the same machine. Formally, a pheromone value τ_{o_i,o_j} exists, iff $g(o_i) = g(o_j)$ or $m(o_i) = m(o_j)$. The meaning of a pheromone value τ_{o_i,o_j} is that if τ_{o_i,o_j} is high then operation o_i should be scheduled before operation o_j. The choice of the next operations to schedule is handled as follows. If there is an operation $o_i \in S_t$ with no related and unscheduled operation left, it is chosen. Otherwise we choose among the operations of set S_t with the following probabilities:

$$p(s[t] = o \mid s_{t-1,|O|}, \tau) = \begin{cases} \frac{\min_{o_r \in S_o^{rel}} \tau_{o,o_r}}{\sum_{o_k \in S_t} \min_{o_r \in S_{o_k}^{rel}} \tau_{o_k,o_r}} & : \text{ if } \quad o \in S_t \\ 0 & : \text{ otherwise} \end{cases}$$

where $S_o^{rel} = \{o' \in O \mid m(o') = m(o) \vee g(o') = g(o),\ o' \text{ not scheduled yet}\}$. We also use the earliest starting time of operations with respect to the partial solution $s_{t-1,|O|}$ as heuristic information to bias these probabilities.
We implemented our algorithm in the Hyper-Cube Framework [4]. The Hyper-Cube Framework is characterized by a normalization of the contribution of every solution used for updating the pheromone values. Our algorithm is also implemented as a $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System using an aggressive pheromone update and additional intensification and diversification strategies. To improve the solutions constructed by the ants we apply the local search defined in Sect. 2 and to the iteration best solution we apply a short Tabu Search of length $|O|/2$ based on the same neighborhood. A more detailed description of the this ACO algorithm can be found in [3].

3.2 Evolutionary Algorithm

The evolutionary algorithm (EA) implemented for the GSP is characterized by a steady-state evolution process and a Lamarckian use of local search. We use a best improvement local search on the neighborhood defined in Sect. 2. Tournament selection is used to choose which individuals reproduce at each generation and a "replace if better policy" is used to decide whether or not to accept the offspring for the new population.

The initial population is built using the non-delay version of the list scheduler algorithm introduced in Sect. 3.1. It builds non-delay schedules, i.e. schedules with no unnecessary idle time: No operation can be finished earlier without delaying any other one, and no machine is ever idle when there is an operation that can be started on it. The population size is set to 50. A solution is represented by a list (total order on O), which induces a total order on each $M \in \mathcal{M}$ and each $G \in \mathcal{G}$.

The crossover is a kind of uniform order based crossover respecting group precedence relations. It generates a child from two parents as follows:

1. Produce a partial child list where each position is either filled with the content of the first parent or left free, with equal probability.
2. Insert the missing operations in the partial list in the order in which they appear in the second parent.
3. Put the current operation in the first free position between the last operation of the previous group (first position in the list if the group is the first on the job) and the first of the next ones (last position in the list if the group is the last on the job), if there is any.
4. Otherwise, if there is a free position before the last operation of the previous group, shift backward all operations to fill the first free position and insert the current operation just before the first operation of the next groups.
5. Otherwise put the current operation in the position of the first operation of the next groups shifting forward the following operations until the next free position is filled.

As mutation operator we implemented a variable neighborhood search (VNS) based on the local search described in Sect. 2 for N_k, $k = 1, \ldots, 10$, where $N_k(\preceq^*) = \underbrace{N(N(\cdots N}_{k}(\preceq^*)))$. That means that a random solution in N_1 is chosen first, then the local search is applied, and if no improvement is found, a random solution in N_2 is chosen followed by local search, then a random solution in N_3 and so on until a better solution is found. The mutation rate is set to be 0.5.

3.3 Iterated Local Search

Iterated local search (ILS), in spite of its simplicity, is a powerful metaheuristic that applies a local search algorithm iteratively to modifications of the current solution. A detailed description of ILS algorithms can be found in [15]. It works as follows. First an initial locally optimal solution, with respect to the given local search, has to be built. A good starting point can be important, if high-quality solutions are to be reached quickly. Then, more importantly, a perturbation has to be defined, that is a way to modify the current solution to an intermediate state to which the local search can be applied next. Finally, an acceptance criterion is used to decide from which solution to continue the search process.

The implementation described here for the GSP works with the local search described in Sect. 2. The initial solution is generated using the same non-delay algorithm as in Sect. 3.1. The idea used for the perturbation is to modify slightly the definition of the problem instance data and apply the local search for this modified instance to the current solution regarded as a solution in the new instance; the result is the perturbed solution in the original problem instance. In the GSP the processing times of the operations, unlike group or machine data, can be easily modified so that a solution to one problem instance can be regarded as a solution to the other. For a percentage α of operations the processing time is therefore increased or decreased, with the same probability, by a certain percentage β of its value; then the local search within the modified problem instance is run for the current solution and finally the resulting locally optimal solution to the modified instance, regarded as a solution to the original

instance, is the perturbed solution. Note that it is not necessarily a local optimum for the original instance. Now the local search can be applied to the intermediate perturbed solution to reach a locally optimal solution.

Finally the acceptance criterion tells us whether to continue the search from the new local optimum or from our previous solution. Random walk, better, and simulated annealing type acceptance criteria have been tested along with different values for α and β. The random walk acceptance criterion with $\alpha = 40$ and $\beta = 40$ has been selected as it gives the best performance.

3.4 Simulated Annealing

Simulated annealing (SA) is a metaheuristic based on the idea of annealing in physics [1]. This technique can be used to solve combinatorial optimization problems, especially to avoid local minima that cause problems when using simpler local search methods. The algorithm starts out with some initial solution and moves from neighbor to neighbor. If the proposed new solution is equal to or better than the current solution, it is accepted. If the proposed new solution is worse than the current solution, it is even then accepted with some positive probability. For the GSP the latter probability is

$$P_{accept} = \exp(-\frac{\Delta}{T}) = \exp(-\frac{(C_{\max}(\preceq^{*\prime}) - C_{\max}(\preceq^{*}))/C_{\max}(\preceq^{*})}{T}),$$

where $\preceq^{*}$ denotes the current solution, $\preceq^{*\prime}$ denotes the the proposed next solution, Δ is the percent cost change, and the temperature T is simply a control parameter. Ideally, when local optimization is trapped in a poor local optimum, simulated annealing can "climb" out of the poor local optimum. In the beginning the value of T is relatively large so that many cost-increasing moves are accepted in addition to cost-decreasing moves. During the optimization process the temperature is decreased gradually so that fewer and fewer cost-increasing moves are accepted.

The selection of the temperature is done as follows. We set the initial temperature such that the probability to accept a move with $\Delta = \delta = 0.01$ is $P_{start} = 0.9$. Moreover, at the end of the optimization process, we would like that the probability to accept a move with $\Delta = \delta = 0.01$ is $P_{end} = 0.1$. With this requirements, we constraint the temperature at time x to be $T = r^{x}\tau_{\max}$, where $\tau_{\max} = -\delta/\ln P_{start}$, $r = \sqrt[t_{\max}]{\delta/(\ln(1/P_{end}) \cdot \tau_{\max})}$, and where $t_{\max}$ denotes the maximum time allowed for computation.

3.5 Tabu Search

Tabu search (TS) is a local search metaheuristic which relies on specialized memory structures to avoid entrapment in local minima and achieve an effective balance of intensification and diversification. TS has proved remarkably powerful in finding high-quality solutions to computationally difficult combinatorial optimization problems drawn from a wide variety of applications [1, 10]. More

precisely, TS allows the search to explore solutions that do not decrease the objective function value only in those cases where these solutions are not forbidden. This is usually obtained by keeping track of the last solutions in term of the move used to transform one solution to the next. When a move is performed it is considered *tabu* for the next T iterations, where T is the tabu status length. A solution is forbidden if it is obtained by applying a tabu move to the current solution.

According to the neighborhood defined in Sect. 2, a move for the GSP is defined by the exchange of certain adjacent critical operation pairs. We forbid the reversal of the exchange of a critical operation pair by recording the iteration number on which the exchange was performed and requiring that this number plus the current length T be strictly less than the current iteration number.

The tabu status length T is crucial to the success of the TS procedure, and we propose a self-tuning procedure based on empirical evidence. T is dynamically defined for each solution. It is equal to the number c of operations of the current critical path divided by a suitable constant d (we set $d = 5$). We choose this empirical formula since it summarizes, to some extent, the features of the given problem instance and those of the current solution. For instance, there is a certain relationship between c and the instance size, between c and the quality of the current solution. In order to diversify the search it may be unprofitable to repeat the same move often if the number of candidate moves is "large" or the solution quality is low, in some sense, when c is a "large" number.

With the aim of decreasing the probability of generating cycles, we consider a variable neighborhood set: every non tabu move is a neighbor with probability 0.8. Moreover, in order to explore the search space in a more efficient way, TS is usually augmented with some aspiration criteria. The latter are used to accept a move even if it has been marked tabu. We consider a tabu move as a neighbor with probability 0.3, and perform it only if it improves the best known solution. To summarize, the proposed TS considers a variable set of neighbors and performs the best move that improves the best known solution, otherwise performs the best non tabu move chosen among those belonging to the current variable neighborhood set.

4 Problem Instances

We tested the proposed metaheuristics on the `whizzkids97` instance. This is a GSP instance that was subject to a mathematics competition in The Netherlands in 1997 [18]. It consists of 197 operations on 15 machines and 20 jobs which are subpartitioned into 124 groups. As this is the only established GSP instance, we derived further problem instances from JSP instances.

The most prominent problem instance for the JSP is a problem with 10 machines and 10 jobs which was introduced by Fisher and Thompson [9] in 1963. It had been open for more than twenty years before the optimality of one solution was proved by Carlier and Pinson [7]. Another famous series of 80 problem instances for the JSP and 60 OSP instances was generated by Taillard [16]. We

used the Fisher-Thompson-instance `ft10` and Taillard's first JSP instance `tai1` on 15 jobs and 15 machines to generate 10 resp. 15 new benchmark instances for the GSP. For both problems, we refined the job partition into a group partition by subdividing each $J_i = o^i_1 \preceq \cdots \preceq o^i_{j_i}$ into b groups of fixed length $g = 1, \ldots, 10$ resp. $= 1, \ldots, 15$ (and possibly one last group of shorter length):

$$\{o^i_1, \ldots, o^i_g\}, \{o^i_{g+1}, \ldots, o^i_{2g}\}, \ldots, \{o^i_{(b-1)g+1}, \ldots, o^i_{j_i}\} \; (b = \lceil j_i/g \rceil) \; .$$

5 Evaluation and Conclusion

We tested the five developed metaheuristics on a PC with an AMD Athlon 1100 Mhz CPU under Linux using the GNU C++ compiler gcc version 2.95.3.[2] For the `whizzkids97`, we tested each metaheuristic for 30 trials of 18000 seconds each (see Fig. 2). TS, although not finding the best solutions found by SA and ILS, showed the best overall performance, followed by ILS, SA, ACO, and EC. We tested the statistical significance of the differences between the algorithms by a pairwise Wilcoxon rank test, which was adjusted by Holm's method [17] for 5 samples. They are significant at a p-value of less than 0.01.

We further tested our metaheuristics on the 10 GSP instances derived from `ft10` for a time limit of 60 seconds per try (see Fig. 3). TS showed the best result for most group sizes, and ACO was on the second rank. EC performed well for small and large group sizes, but was worse than SA for groups of medium size.

We observed a similar behavior for the 15 instances derived from `tai1`, which we tested for running times of 600 and 1800 seconds per try. ACO is for nearly all group sizes quite close to the performance of TS. However, the TS is the only algorithm that finds (even within 600 seconds) the optimal solution for the original JSP version of `tai1`. EC again performs rather poorly on medium group sizes and performs well on small and big group sizes.

We noticed that the SA in general compared well to the other algorithms. This indicates the power of the neighborhood structure defined in Sect. 2. Although the TS approach yielded the overall best performance, for some group sizes other metaheuristics showed advantages. Our fair comparison showed that depending on the position of the problem instance between the JSP and the OSP different heuristic techniques are helpful. The GSP might best be tackled by a hybrid metaheuristic approach that combines the elements of the algorithms described in this work according to the results of our analysis.

Acknowledgements. We would like to thank Anthony Liekens and Huub ten Eikelder for the implementation of the problem representation and the local search. Our work was supported by the *Metaheuristics Network,* a Research Training Network funded by the Improving Human Potential Programme of the CEC, grant HPRN-CT-1999-00106, and by Swiss National Science Foundation

[2] All algorithms, the test instances and a detailed evaluation of the generated results can be found on `http://iridia.ulb.ac.be/~msampels/gsp.data` .

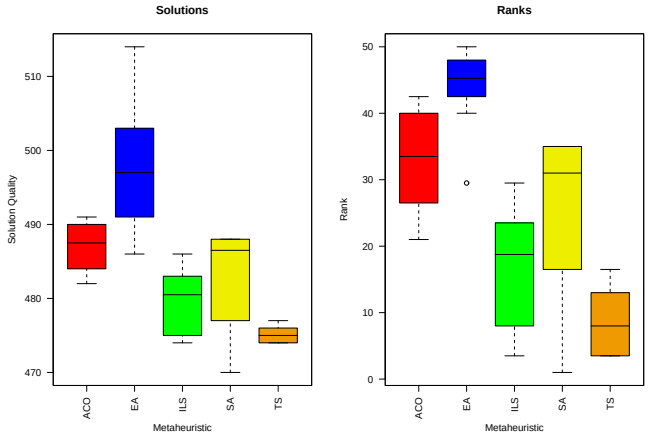

Fig. 2. The absolute values of the solutions to the `whizzkids97` problem instance generated by the five metaheuristics (left) and their relative rank in the comparison among each other (right) are depicted in two boxplots. A box shows the range between the 25 % and the 75 % quantile of the data. The median of the data is indicated by a bar. The whiskers extend to the most extreme data point which is no more than 1.5 times the interquantile range from the box. Extreme points are indicated as circles

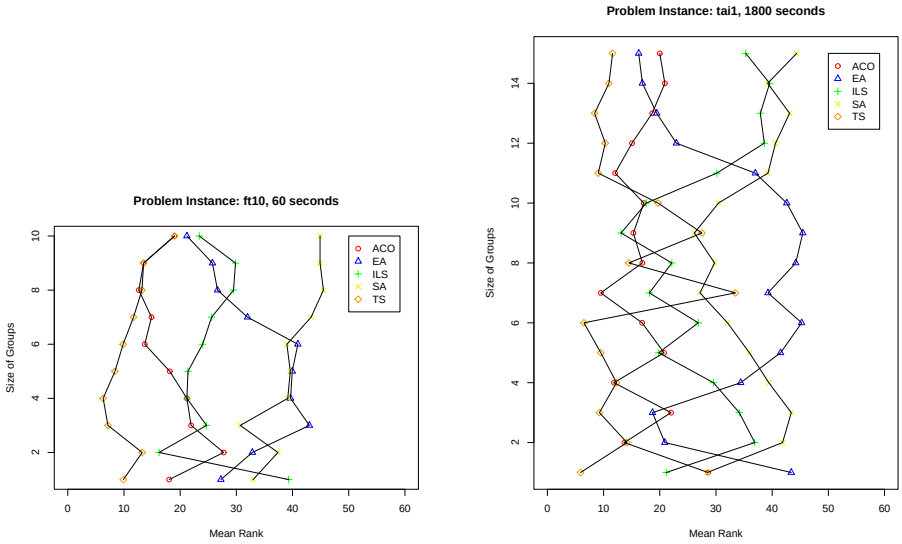

Fig. 3. Mean ranks of the solutions generated by ACO, EA, ILS, SA, and TS to instances derived from `ft10` and `tai1`. For `ft10` (left) the group size was varied from 1 to 10, and for `tai1` (right) it was varied from 1 to 15. For the `ft10` instances we ran the algorithms for 60 seconds, for the `tai1` instances for 1800 seconds

project 20-63733.00/1, *Resource Allocation and Scheduling in Flexible Manufacturing Systems.* 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. E. H. L. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Chichester, 1997.
2. J. Błażewicz, W. Domschke, and E. Pesch. The job shop scheduling problem: Conventional and new solution techniques. *European Journal of Operational Research*, 93:1–33, 1996.
3. C. Blum. ACO applied to Group Shop Scheduling: A case study on Intensification and Diversification. In *Proceedings of the 3rd International Workshop on Ant Algorithms (ANTS 2002) (to appear)*, 2002. Also available as technical report TR/IRIDIA/2002-08, IRIDIA, Université Libre de Bruxelles.
4. C. Blum, A. Roli, and M. Dorigo. HC-ACO: The hyper-cube framework for Ant Colony Optimization. In *Proceedings of the 4th Meta-heuristics International Conference (MIC 2001)*, volume 2, pages 399–403, 2001.
5. C. Blum and M. Sampels. Ant colony optimization for FOP shop scheduling: A case study on different pheromone representations. In *Proceedings of the 2002 Congress on Evolutionary Computation (CEC 2002)*, volume 2, pages 1558–1563, 2002.
6. P. Brucker, B. Jurisch, and B. Sievers. A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics*, 49:107–127, 1994.
7. J. Carlier and E. Pinson. An algorithm for solving the job-shop problem. *Management Science*, pages 164–176, 1989.
8. M. Dorigo and G. Di Caro. The Ant Colony Optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*. McGraw-Hill, 1999.
9. H. Fisher and G. L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In J. F. Muth and G. L. Thompson, editors, *Industrial Scheduling*. Prentice-Hall, Englewood Cliffs, NJ, 1963.
10. F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Boston et al., 1998.
11. T. Gonzalez and S. Sahni. Open shop scheduling to minimize finish time. *Journal of the ACM*, 23(4):665–679, Oct. 1976.
12. J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.
13. E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42(6):797–813, June 1996.
14. M. Sampels, C. Blum, M. Mastrolilli, and O. Rossi-Doria. Metaheuristics for Group Shop scheduling. Technical Report TR/IRIDIA/2002-07, IRIDIA, Université Libre de Bruxelles, 2002.
15. T. Stützle. *Local Search Algorithms for Combinatorial Problems – Analysis, Improvements, and New Applications*. PhD thesis, TU Darmstadt, Germany, 1998.
16. E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64:278–285, 1993.
17. S. P. Wright. Adjusted p-values and simultaneous inference. *Biometrics*, 48:1005–1013, 1992.
18. http://www.win.tue.nl/whizzkids/1997 .