Comparison of Iterative Improvement Techniques for Schedule Optimization

**Jürgen Dorn, Mario Girsch,
Günther Skele, Wolfgang Slany**

CD-TR 94/61

Expert Systems

Technical Report

Christian Doppler Laboratory
for Expert Systems

Vienna University of Technology
Institute for Information Systems

Abstract*

Due to complexity reasons of realistic scheduling applications, often iterative improvement techniques that perform a kind of local search to improve a given schedule are proposed instead of enumeration techniques that guarantee optimal solutions. In this paper we describe an experimental comparison of four iterative improvement techniques for schedule optimization that differ in the local search methodology. Namely, these techniques are iterative deepening, random search, tabu search and genetic algorithms.

To enable a comparison of the performance of these techniques we use the same evaluation function, knowledge representation and data from the application. The evaluation function is defined on the gradual satisfaction of explicitly represented domain constraints and optimization functions. It allows us the combination of different kinds of constraints. The satisfaction of individual constraints is weighted and aggregated for the whole schedule.

We have applied these techniques on data of a steelmaking plant in Linz (Austria). In contrast to other applications of iterative improvement techniques reported in the literature, our application is constrained by a greater variety of criteria that are partly contradictionary.

* This paper is also published in Proceedings of the 13th UK Planning Special Interest Group, Strathclyde, Schottland, 1994

1 Introduction

Many years of research in scheduling were dominated by the search for techniques to achieve optimal schedules. Especially for flow shops there was the hope of finding efficient algorithms because flow shops are structured more strictly than general job shops (Dudek *et al.* 1992). With the rise of complexity theory (Garey and Johnsson 1979), it was found that even for flow shops with a number of jobs and machines typical for real applications, the problem is intractable.

Consequently, problem solving methods often apply heuristics to find solutions that are not necessary optimal. We can distinguish between heuristic domain knowledge and general heuristics. Although domain heuristics are often more efficient, one tries to find general heuristics to allow a reuse of scheduling methods.

From another view, heuristics can be divided in order-centred and machine-centred heuristics. In an order-centered heuristic one selects an order or job due to its characteristics. For example, the job with the earliest due date can be selected first. In (Panwalkar and Iskander 1977) over hundred of such heuristic rules are cited. Machine-centered heuristics can concentrate on bottleneck machines (Adams *et al.* 1988). Although these heuristics claim to be general, they are often limited to certain evaluation functions and constraints. Moreover, sometimes the solution is not so good as expected.

Recently, some new scheduling procedures occurred that are based on the idea to improve an existing schedule iteratively. By simple modifications that do not cost much a schedule is changed and evaluated anew. If this new schedule is better than the old, this procedure is repeated. By iteration a search for a good schedule is performed.

How a given schedule is modified influences of course the efficiency of the iterative improvement technique. The first idea would be to apply a kind of hill-climbing search strategy. Unfortunately, in scheduling problem there are usually many local maxima and therefore the search would be stuck in such a local maxima. Some experiences have shown that one can find good solutions with random techniques. This observation may have two reasons. The success may be due to the smaller amount of reasoning to find a modification, because a more clever selection of a modification can have considerable costs. Another explanation is that with such a randomized modification local optima may be escaped easier. It seems reasonable that a successful iterative improvement method applies a mixture of random and knowledge-based decisions. The basic parts of such an approach are an evaluation function, a set of modification operators, a procedure to select a modification, and the overall control algorithm.

Usually, the new problem solving methods like *Genetic Algorithms* (Davis 1985), *Simulated Annealing* (van Laarhoven *et al.* 1992), *Tabu Search* (de Werra and Hertz 1989) or *MinConflicts heuristic* (Minton *et al.* 1990) are only compared to other enumerative algorithms like branch-and-bound methods. We have started to compare some of the iterative improvement techniques. We have used an application in a steelmaking plant as a benchmark example. This example gives also new insights in how one can adapt proposed iterative improvement techniques in a different domain because the steelmaking application differs considerably from the typical examples in the literature. Our examination is motivated by a new project where we want to apply the best optimization approach.

In the next section we describe the application. Then we explain the representation of the domain constraints and the evaluation function. In the fourth section we present the four problem solving methods: iterative deepening search, pure randomized modification and search, tabu search and genetic algorithms. Finally, we compare these algorithms and conclude.

2 Application

The steelmaking plant LD3 of the VOEST Alpine Stahl AG in Linz produces steel in two continuous casters. Each day a cast sequence of about 35 charges for the next 24 hours is planned for one caster. Hot pig iron is delivered from the blast furnace to the LD-converters. The steel is then poured into ladles and processed further in secondary metallurgy aggregates and is finally delivered in time to the casters. The casters produce continuously strands of steel that are cut into slabs of ordered length. Although the steel is produced in different aggregates in a fixed flow, the casters are the bottleneck resource of the shop. After a duration of approximately 240 minutes the tundish, a part of the caster, has to be maintained. A second tundish is used, because this maintenance takes about 100 minutes. Before one tundish is worn out the tundishes are exchanged to allow a continuous operation of the caster.

The main criteria that constrains the sequence of charges (jobs) are compatibility constraints between jobs. There are three compatibility aspects that must be regarded. The steel grade (the chemical analysis) of subsequent charges must be similar, the casting format may vary only in certain ranges and the degassing procedure in the secondary metallurgy must be compatible. Rules about which steel qualities may be cast one after the other are given explicitly. Usually, there are hard constraints saying that two qualities may not be sequenced, but there are also soft rules that say that it is not so good to sequence two qualities.

To maximize the production the casters should process the steel continuously without interruptions. If no feasible sequence can be found, three operations may help:

- To separate two incompatible jobs, a plate may be inserted into the strand. This results in the production of a useless slab. These costs are quantifiable. However, this operation has also some compatibility constraints. If the difference between the subsequent steel grades is too strong, no *quality separation* is possible.
- Also a *change of tundishes* may help. Such a change has higher costs than a quality separation. However, if the change must be performed anyway due to the required maintenance operation, the costs are less.
- The most expensive operation is a new *set-up* of the caster. This means some hours of interruption without casting. Set-ups are required when an interuption in the delivery occurs or the duration between two changes of tundishes is less than 100 minutes.

Besides compatibility constraints, also due date constraints have to be regarded. However these are relaxable. In general, it is no problem to find a feasible schedule because a set-up is always possible. However, to find a good schedule is a matter of minimizing the number of tundish changes, quality separations and set-ups. Furthermore, due date violations and the sum of format changes is to be minimized.

In a first expert system approach, Stohl *et al.* (1991) applied a domain heuristic for this application. Groups of similar compatible jobs are generated and scheduled together. Although the expert system finds good feasible schedules, they believe that these schedules can be improved more. Additionally, the disadvantage of this domain dependent strategy is that it is not so easily reusable in a new application.

In the following experiments we apply four different improvement techniques to construct from a randomly generated preliminary schedule a good one. Furthermore, we have also measured the algorithm combined with this domain heuristic, i.e. we have initialized the search with two schedules found with the domain heuristic.

3 Knowledge Representation

We represent the scheduling domain by objects like machines, orders, jobs, operations and schedules. Jobs can have complex process plans, however, from the view of the iterative improvement techniques only a sequence of operations on the caster is modeled. If a conflict occurs on another machine this is reflected by a bad value of the evaluation function.

We distinguish between hard and soft constraints in the application. For example, a hard constraint is that after 240 minutes an exchange of the tundishes must be scheduled. On the other hand, due dates are considered but they are only soft constraints. We construct only schedules without violations of hard constraints. Soft constraints are subject to optimization and their satisfaction is considered in the evaluation function.

To evaluate given schedules we describe explicitly the satisfaction of all constraints and objectives. For example, if a due date is given for an order, the satisfaction of this due date in a certain schedule is represented by a real value[1] between 0 and 1 whereby 0 means a strong violation of the constraint and 1 full satisfaction. Due date constraints and an objective like minimizing the number of set-ups have of course different importance. In our application due dates are not so important. Therefore different kinds of constraint violations get different weights. Although compatibility constraints are the most important constraints in the application, they get a relatively low weight in our evaluation function because we avoid strong violations by automatic insertion of one of the operations like quality separation, tundish change, or set-up. We obtain for a given schedule a number of constraint violations that can be aggregated and standardized. The following table shows the considered constraints and their weights.

Table 1: Considered constraints and their weights

c_i	type	weight (χ_i)
c_1	minimization of set-ups	1.0
c_2	minimization of number of tundish changes	0.9
c_3	minimization of number of quality separation	0.8
c_4	maximization of the length of tundishes	0.7
c_5	sum of all format changes	0.7
c_6	due dates	0.6
c_7	compatibility	0.6
c_8	negative format changes	0.2

1 We believe that for a better modeling of this satisfaction degrees fuzzy sets are appropriate as proposed in (Dorn and Slany 1994), however for simplicity in this paper we restrict the expressiveness on real values.

The evaluation function $eval(S)$ is now the mean of the weighted constraint violations $violation(C_i)$, whereby M is the number of constraints in the schedule S and χ_j is the weight of the constraint:

$$eval(S) = \frac{1}{M} * \sum_{i=1}^{M} \chi_i * violation(C_i)$$

The value of this evaluation function is always between 0 and 1. The optimal value is 1, but it is sure if there are different steel grades involved that this value can be never reached. Good values for the data that we have used are between .92 and .93.

4 Iterative Improvement Techniques

Several algorithms or problem solving methods that are described in the literature can be classified as iterative improvement techniques. What they have in common is that they start with a preliminary schedule that is improved in several iterative steps. However, they were developed independently. Therefore their descriptions are not so coherently that they can be compared easily. We try to change the description without changing their behaviour. We describe them so that they have a common part namely the loop that is iterated. Moreover, we use for all algorithms the same initial schedules with 35 jobs, the same evaluation function and the same stopping criteria. We stop every algorithm after five minutes (300 seconds) run time. However, the algorithms are all designed so that they can be interrupted anytime and return the best solution found so far thus achieving the functionality of an *anytime algorithm* (Boddy and Dean 1989).

In principle, an iterative improvement technique can be trapped in a local maximum. To escape from such a maximum, schedules with a worser evaluation have to be accepted as candidates for further search, too. We model the search phase between two actual best schedules with a second iteration. This iteration consists of several modifications that are found in a second loop and we call the modification from a currently best schedule to a better schedule a *repair step*. The search for the modifications are performed with different techniques. In the following we apply four different techniques: exhaustive search by iterative deepening, random search, tabu search and search by genetic algorithms. In the original description of these algorithms there were no nested iteration.

The applied modifications are similar in all approaches. We assume the schedule is a sequence of jobs whose order is modified only. Hard constraints (compatibility constraints) that are violated by such a modification are repaired automatically through the insertion of necessary quality separations, tundish changes or set-ups. To check due date constraints the operations are always scheduled as early as possible. The operators that are considered in the different approaches are described in the following table.

Table 2: Modification Operators

Operator	Explanation
a job is shifted to another place	123456789 → 123567849
two jobs are exchanged	123456789 → 128456739
a group of jobs is shifted	123456789 → 156723489
two groups are exchanged	123456789 → 167845239

By means of our evaluation function that is defined on single constraint violations we have the possibility to focus the search. Minton *et al.* (1990) have shown that it is a good heuristic to modify a part of the schedule that is involved in the greatest constraint conflict.

In each of the described algorithm some decisions are generated randomly. Therefore, we have performed series of 10 measurements. In all experiments we use the same data set, but we start with different preliminary schedules. We have one schedule (A) that was generated randomly and has a bad initial score of .58139. The second schedule (B) is better (.78312) because the jobs are grouped by the domain heuristic of (Stohl *et al.* 1991). The third schedule (C) has a score of .90. It was constructed by placing the groups with a domain heuristic and due dates are satisfied, too.

4.1 Iterative Deepening

We have used a kind of *iterative deepening* search strategy (Korf 1987) because this seems to be the best exhaustive search technique for this problem. Pure depth-first or backtracking search is not so good because it is not know how deep one must search and breadth-first search is ruled out because it would require too much memory.

In iterative deepening search in a first attempt one searches with backtracking to depth 1. If no improvement is found, in a second attempt, a backtracking search to depth 2 is performed. The depth limit is incremented iteratively until a repair step is found. On each depth level all generated schedules are evaluated. The modification operator is always applied to jobs that are involved in the greatest conflict of the actual schedule.

In each iteration first these operators that move single jobs that are involved in the greatest constraint violation are applied first. If no improvement is found, groups of jobs are moved. To move all possible groups would be intractable, but it is a good heuristic to build groups consisting of the jobs between two tundish changes or the jobs between a quality separation and a tundish change.

We distinguish three variations of the iterative deepening algorithm. In the first version it is tried to repair the greatest constraint violation by means of operators that move single jobs that are involved in this constraint violation. If no improvement is found groups of jobs are moved. If no improvement is found, a search with depth 2 is started. This version of the iterative deepening approach is deterministic and always finds the same solution. We have implemented two variations that try to apply all possible modifications related to the *greatest* constraint violation. If no improvement is found, randomly n other constraint violations are examined. We have tested this version with n = 5 and n = 10. In the following figure we show the performance of the three variations whereby the deterministic version is depicted by 0. We see that this incorporation of randomness brings no improvement in the solution quality or in the required time.

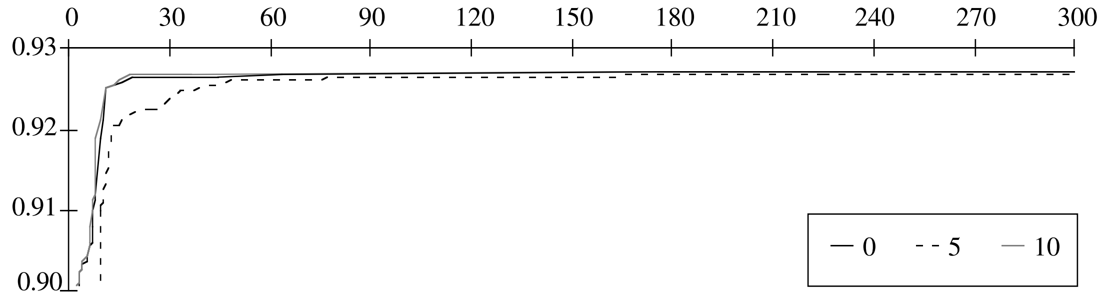

Figure 1: Performance of three variations of iterative deepening

In the following diagram the performances for the three preliminary schedules with the deterministic version of the iterative deepening approach are plotted. For all schedules a very good evaluation was found after about 30 seconds and this solution is for each case better than the solution of the original expert system. The value of .90 that was achieved by the original expert system was reached for the preliminary schadue A in 3 seconds with 45 repair steps. This value was achieved for the second schedule in 10 seconds in 10 repair steps. However, the best solutions of all was found with schedule B in 72 seconds with a value of .92725. We see that the goodness of the initial schedule has relatively small influence on the solution if we have enough time.

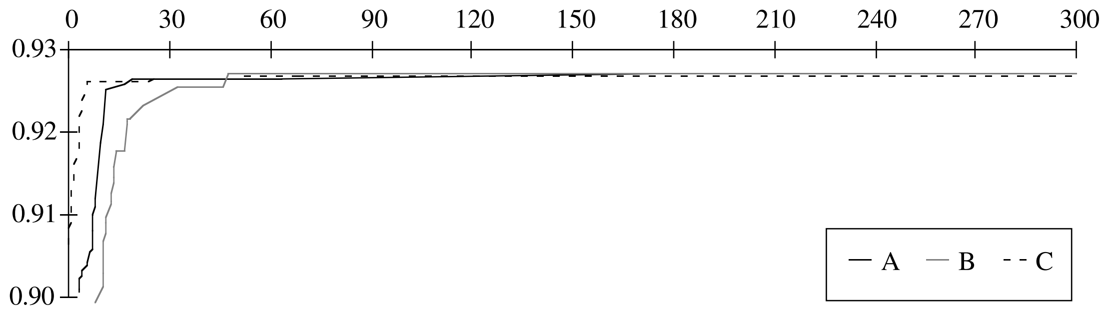

Figure 2: Performance of iterative deepening

4.2 Random Search

The iterative deepening approach searches deterministically the search tree and is exhaustive. It seems therefore promising to find guaranteed the best neighbour. However, this can be very time consuming if many levels of the search tree must be examined. Also the number of neighbours may be very large. In our experiments the number of neighbours was restricted because we try to repair the greatest constraint conflict. Often a randomized search is proposed that does not examine all branches of the search tree.

We have implemented an iterative improvement algorithm that decides randomly, which jobs shall be moved and which operators shall be applied. The new generated schedule is evaluated and if no improvement is found, a next modification is generated randomly. The advantage in contrast to the iterative deepening algorithm is that not all successors of a schedule must be evaluated and in our series of ten performances for the preliminary schedule C we have found one schedule that obtains a better value (.92895) than those schedules that were found by iterative deepening. The average measurements for the three preliminary schedules are shown in the following figure. We see that the goodness of the initial schedule has a greater influence on the solution than for the iterative deepening approach.

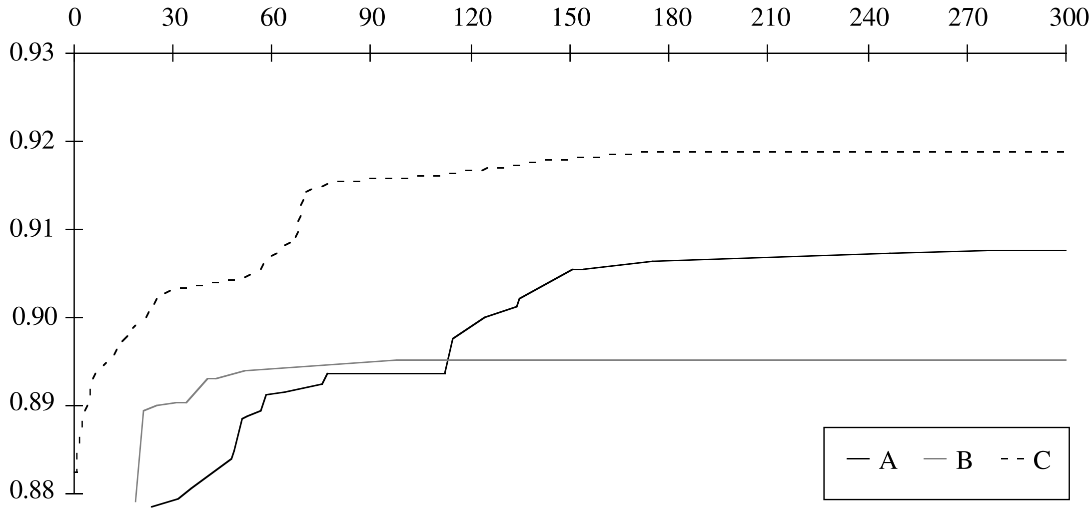

Figure 3: Performance of random search

However, the problem of this approach is the very great variance. For schedule A the algorithm has required between 61 to 85 repair steps to obtain a score between 0.89751 and 0.92011. The time has varied between 109 and 276 seconds. For schedule C the variance is shown in the following figure.

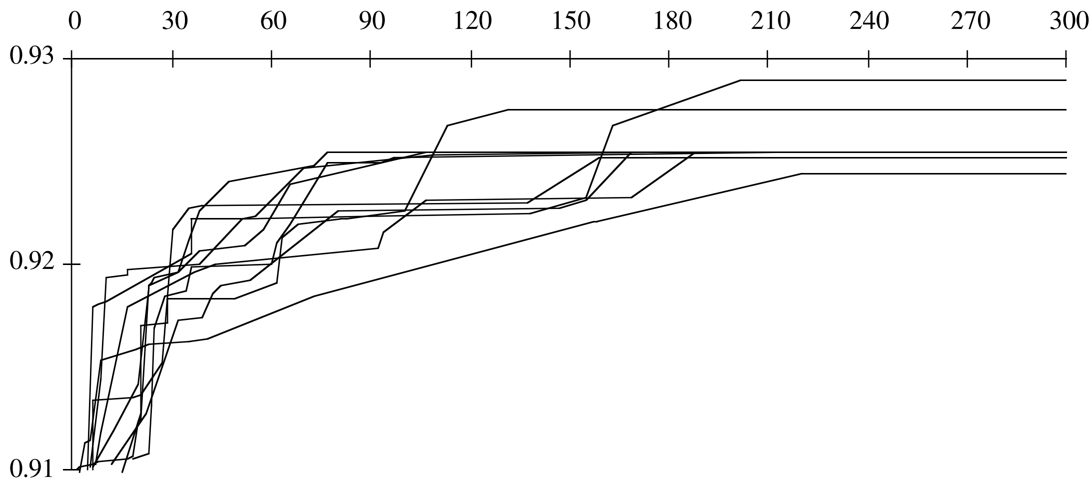

Figure 4: Variance of random search with preliminary schedule C

4.3 Tabu Search

In *tabu search* (de Werra and Hertz 1989) one selects always the best modification operator until one is trapped in a local maximum. To escape from this local maximum it is possible to select a modification that leads also to a schedule with a worse evaluation. A *tabu list* is introduced because such a search can run in cycles. This list contains the last n modifications and for every new modification one checks whether it is in the tabu list. In this case the modification is tabu, i.e. it is forbidden and another modification must be chosen. Small cycles of length n are avoided by a tabu list that is usually implemented as a ring buffer. However, after *n* iterations it is theoretically possible to return to an old schedule. We store in the tabu list the position of jobs. For example, a pair (5, 7) means, the job at position 5 was exchanged with the job at position 7. In our experiments we have restricted the length of the tabu list to 12.

We first try to “repair” the greatest constraint conflict and stop the search after 50 modifications. If no improvement is found in 50 steps, the search is started again with an arbitrary constraint. The advantage in contrast to our first approach with iterative deepening is that not so many evaluations have to be made. The following figure shows the performance of tabu search for the three preliminary schedules.

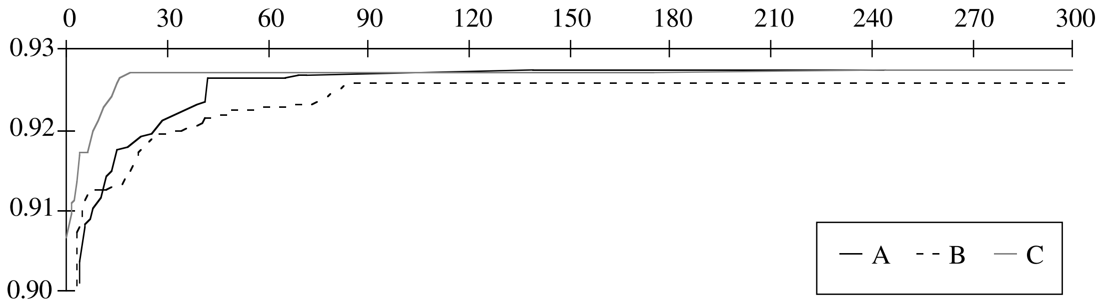

Figure 5: Performance of tabu search

The following figure shows the the performance of 10 runs with schedule A. The achieved score varies between .926 and .928. From the variance one can see that it happens often that 50 modifications are not enough to find an improvement.

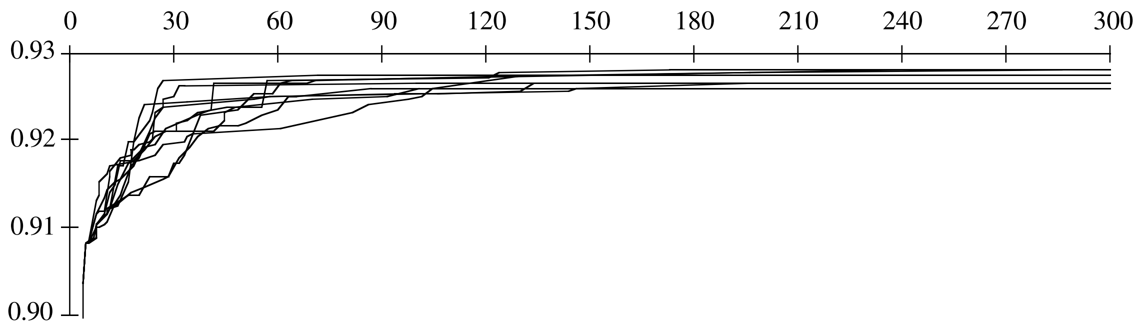

Figure 6: Variance on the performance for Tabu Search

We have performed a second set of experiments with tabu search whereby not always the best modification is looked for. If a modification is found that improves the actual schedule, the resulting schedule is used as new best schedule. We have examined this case because the search for the best modification can be very time consuming. However, the experiments show that for our data set it is usually better to search for the best modification. Only for the preliminary schedule B we have obtained a very good value in one case. This is shown in the next figure.

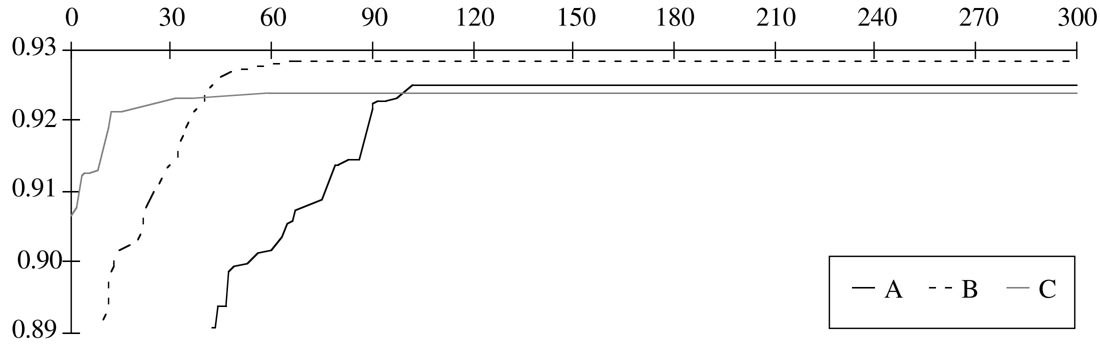

Figure 7: Performance of second version of tabu search

4.4 Genetic Algorithms

Genetic Algorithms first proposed by (Holland 1975) are a class of algorithms that imitate natural selection and genetics. Genetic Algorithms maintain a set of candidate solutions called population. Members of the population are chromosomes. The first population is initialized at random and evolves in generations. In each generation the population is affected by genetic operators and selection mechanism. Genetic operators such as crossover, inversion and mutation provide information flow among chromosomes while selection promotes survival of the fittest chromosomes. A fitness function evaluates individual chromosomes.

Starting with (Davis 1985) several proposals for the application of Genetic Algorithms in scheduling were made. Originally chromosomes were represented as bit strings. Because this representation is unpracticable for scheduling, usually integers representing a job in a sequence of jobs are used. To generate only meaningful chromosomes special operators are designed on these sequences of integers. Genetic Algorithms differ from the other described search techniques in that they maintain always a set of schedules.

In our experiments we start with a population of 60 randomly generated chromosomes. However, the three preliminary schedules (A, B, C) that we have used before are also in this population. These are evaluated and 20 chromosomes are deleted whereby the chance for bad chromosomes to be deleted is higher than for better chromosomes. Then we select randomly pairs of these chromosomes and generate children by choosing randomly a crossover operator. Afterwards a mutation operator is applied on the children. The result of these operations is that we have again 60 chromosomes. We support furthermore elitism in that the fittest chromosome of the ancestors and children always survive which guarantees that the algorithm can be stopped anytime and is able to deliver always the best schedule found so far.

The applied crossover operator selects randomly two positions p1 and p2 in one chromosome and exchanges these genes with that of the second operator. The other positions will be filled up with the genes of the first chromosome keeping their order. This operator is often called order-based crossover.

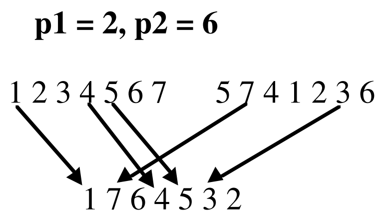

Figure 8: Crossover operators

The mutation operator selects randomly two positions of one chromosome and exchanges both genes.

As we use an heuristic to guide our search in the other approaches, we have made experiments to improve the Genetic Algorithms by a similar technique. In the other approaches we have obtained good results by repairing the greatest constraint conflict. We have therefore apply a probabilistic selection of positions in the the crossover and mutation operator. Now the probability to mutate two genes is greater if one or both of them are involved in a great constraint violation. We have made three sets of measurements for schedule A whose average performance is shown in the next figure:

- in the first experiments (w/o) we use pure random selection of the positions
- in the second performance (wm) the selection of the position in the mutation operator is selected by probability and
- in the third performance also for the crossover operator this probability model is used.

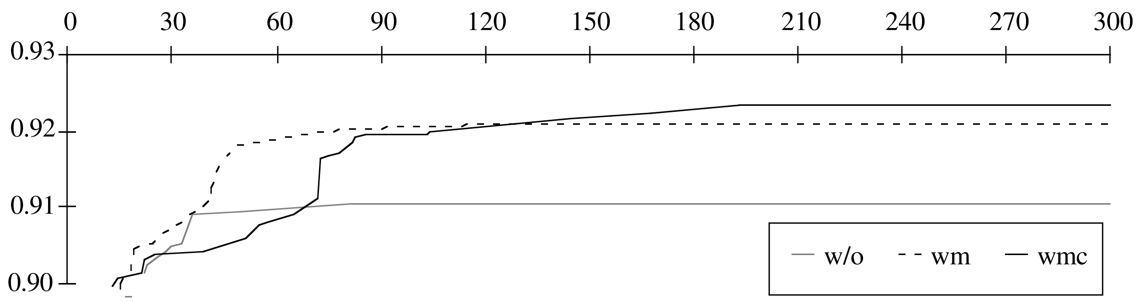

Figure 9: Performance of genetic algorithms

We see that the Genetic Algorithm enhanced by the probability model achieves a better performance and in the following figure the variance for 10 test runs with the improved operators is shown. Although the variance is great, in the worst case the performance is better than in the case of pure random selection.

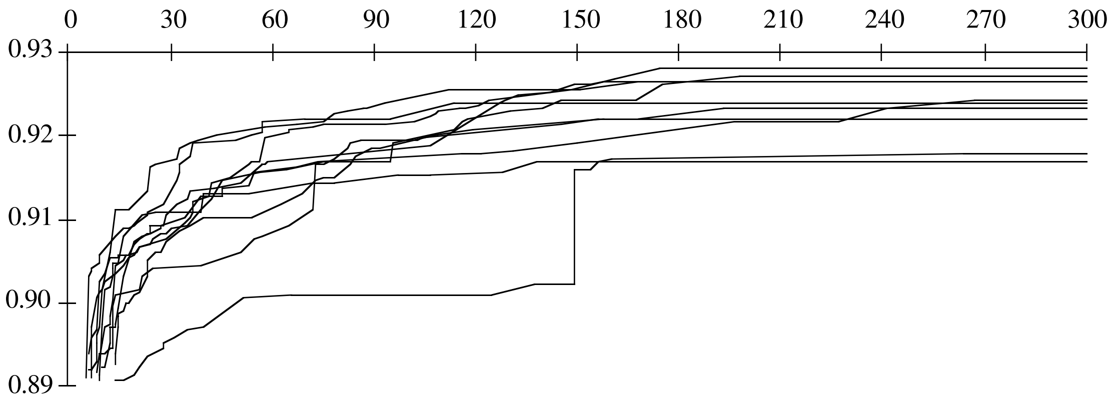

Figure 10: Variance for 10 runs on Schedule A with enhanced Operators

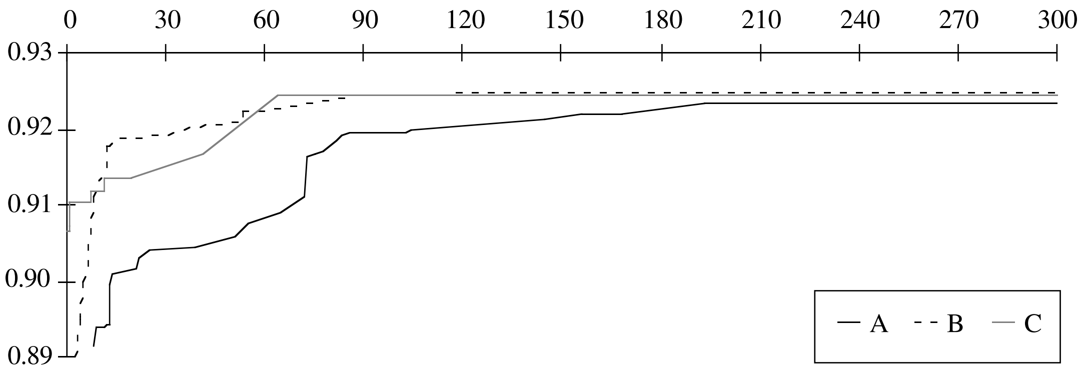

Figure 11: Comparison for different initial schedules

4.5 Comparisons of algorithms

The described experiments were part of two masters thesis. They were implemented in C++ on a SPARC-station and on a PC. We have measured the time on 386-PC because the measurements on a single user machine is more expressive than on a multi-user operating system with virtual memory management. Of course this was only possible because all algorithms do not have large space requirements.

When which algorithm?

Is domain heuristic or first schedule important?

Which algorithm seems best for reactive scheduling?

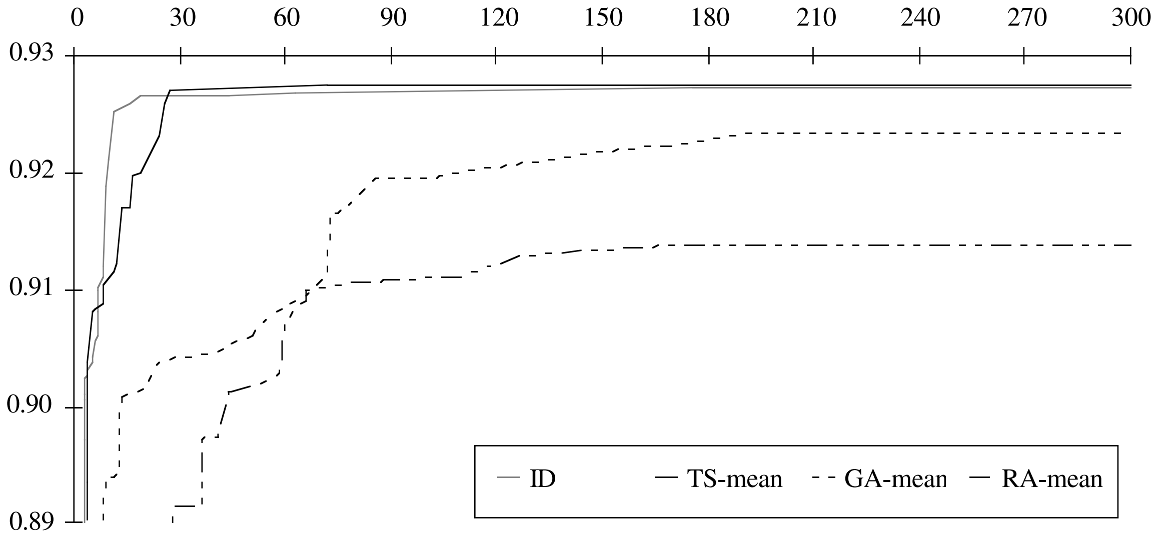

Figure 12: Comparison of four algorithms applied to schedule A

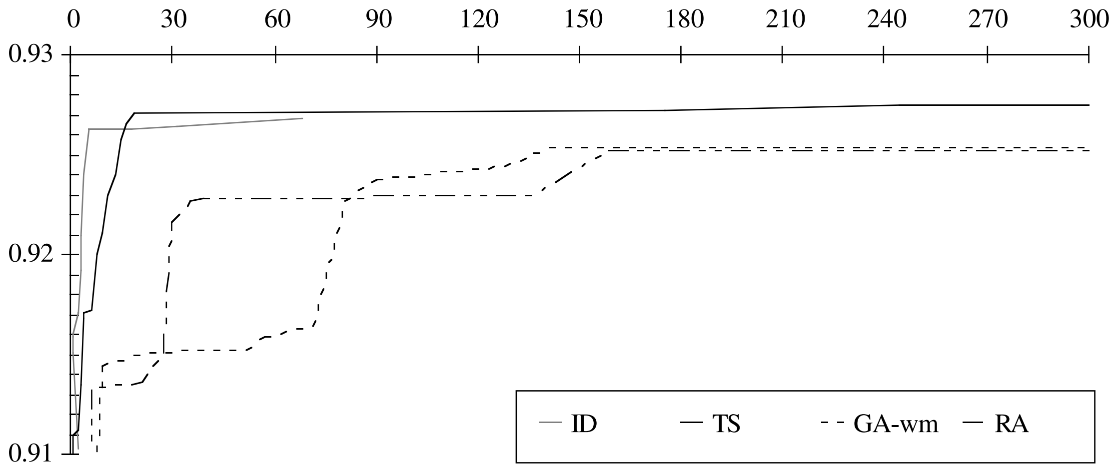

Figure 13: Comparison of the four algorithms applied to schedule C

5 Conclusions and Outlook

Our contribution to the progress in the research on scheduling systems is twofold. We have used an example coming from a real application with many different and partly redundant or contradictionary constraints. This knowledge-intensive application demands for a knowledge-based approach with explicite representation of constraints. To cope with the contradictionary and over-constrained problem, we have developed a model to describe the gradual satisfaction of given constraints thus enabling an easy relaxation of constraints. This representational model supports also a kind of “min-conflicts”-strategy (Minton et al. 1990) where one tries to reduce the violations of constraints. In contrast to the original strategy, we have constraints that can be fulfilled partially.

We have defined a framework to compare several improvement techniques. Since these algorithms have used the same knowledge representation and evaluation function a comparison of the performance of the algorithms is more expressive than other comparisons in the literature.

Benchmark tests are becoming an important mean for the evaluation of AI-concepts and algorithms. However, such benchmarks must be designed carefully to obtain relevant results. For example, one has to find a compromise between a problem of realistic size and a problem whose parameters can be controlled easily (Hanks *et al.* 1993). We have taken data from a real application and have fixed a great number of variables in the algorithm to control the tests. However, there are still many uncontrolled parameters like e.g. the selection of modification operators that influence the performance of the algorithms. We believe that conclusions can be taken from the experiments, but it seems reasonable that some of the algorithms could be improved by a more specific selection of operators.

For scheduling, Drummond (1993) has listed a handful of benchmark test sites were data can be obtained by ftp or email. However, these examples have only very simplistic structures and usually only temporal constraints are regarded. On the other side, Fox (1983) demands knowledge-based solutions to the scheduling problem because one of the advantages of knowledge-based systems is the easiness of representing and processing different constraints.

Our current work is motivated strongly by the design of reusable scheduling software. In the given framework it is relatively simple to apply a different modification technique. If we have used tabu search for an application and we want to try genetic algorithms we can reuse most of the software that was implemented for tabu search. However, if we go to another application it is not so easy. For example, the constraints in the steelmaking application described in (Dorn and Slany 1993) must be expressed differently because it is not possible to define a simple procedure to construct a feasible schedule after a modification. Constraints that have to be repaired in this application can also be hard constraints.

Another open problem is the evaluation function based on the explicit representation of constraint violations. If we are going to an expert in the plant he can neither give us weights of his constraints nor satisfaction degrees. These values must be estimated and tested. If the system generates a schedule, the expert can say whether this schedule is satifying important constraints. Especially, if the scheduler has to react on a dynamically changing production process, (i.e. reactive scheduling) the evaluation function will be very complicated. The advantage of all proposed algorithms is that not only predictive scheduling but also reactive scheduling can be easily implemented (Dorn et al. 1993). If something happens in the production process this has to be mapped on the current schedule resulting in some new constraint violations. With an improvement strategy this schedule can be repaired. However, the performance of such a reactive scheduler cannot be tested easily. A test-bed with simulation of failures in the production process has to be used. Because one of our goals is to use the improvement operators for reactive scheduling, one of our next steps will be the definition of such a test-bed. This environment should support furthermore the testing of interactive scheduling between a user and a system reasoning with an improvement strategy.

6 References

Adams, J., Balas, E. and Zawack, D. (1988) The Shifting Bottleneck Procedure for Job Shop Scheduling, *Management Science* **34** (3), pp. 391–401.

Boddy , M. and Dean, T. (1989) Solving Time Dependent Planning Problems, *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pp. 979–984.

de Werra, D. and Hertz, A. (1989) Tabu Search Techniques: A Tutorial and an Application to Neural Networks, *OR Spektrum* **11**, pp. 131–141.

Davis, L. (1985) Job Shop Scheduling with Genetic Algorithms, *Proceedings of the 1st International Conference on Genetic Algorithms*, Lawrence Erlbaum.

Dorn, J. and Shams, R. (1991) An Expert System for Scheduling in a Steelmaking Plant, *Proceedings of the 1st World Congress on Expert Systems*, Orlando, Fla, Pergamon Press, pp. 395–401.

Dorn, J,, Kerr, R. and Thalhammer, G. (1993) Reactive Scheduling in a Fuzzy-Temporal Framework, *Proceedings of the IFIP TC5/WG5.7 Workshop on Knowledge-based Reactive Scheduling*, Athens.

Dorn, J. and Slany, W. (1994) A Flow Shop with Compatibility Constraints in a Steelmaking Plant, in Fox and Zweben (eds) *Intelligent Scheduling*, Morgan Kaufmann (to be published).

Drummond, M. (1993) Scheduling Benchmarks and Related Resources, *Newsletter of Special Interest Group in Manufacturing of the American Association for Artificial Intelligence* 6 (3), pp. 1–4.

Dudek, R.A., Panwalkar, S.S. and Smith, M.L. (1992) The Lessons of Flowshop Scheduling Research, *Operations Research* **40** (1), pp. 7–13.

Garey, M. R. and Johnson, D. S. (1979) Computers and Intractability: A Guide to the Theory of NP-Completeness, New York: Freeman.

Filipic, B. (1993) Enhancing Genetic Search to Schedule a Production Unit, in *Scheduling of Production Processes*, J. Dorn and K. A. Froeschl (eds), Chichester: Ellis Horwood, pp. 61–69.

Fox, M.S. (1983) *Constraint-directed search: a case study of job-shop scheduling*, Ph.D. Computer Science Dept., Carnegie-Mellon University.

Hanks, S., Pollack, M.E., and Cohen, P.R. (1993) Benchmarks, Test Beds, Controlled Experimentation, and the Design of Agent Architectures, *AI Magazine* **14** (4) pp 17–42.

Holland, J.H. (1975) *Adaptaion in Natural and Artificial Systems*, Ann Arbor: University of Michigan Press.

Korf, R.E. (1987) Depth-First Iterative-Deepening, *Artificial Intelligence* **27** (1), pp. 97–109

Laarhoven, P.L.M.van, Aarts, E.H.L. and Lenstra, L.K. (1992) Job Shop Scheduling by Simulated Annealing, *Operations Research* **40**, pp.113–125.

Lawton, G. (1992) Genetic Algorithms for Schedule Optimization, *AI Expert*, pp. 23–27.

Minton, S. Johnston, M. Philips, A. Laird, P. (1990) Solving Large-Scale Constraint Satisfaction and Scheduling Problems Using a Heuristic Repair Method, *Proceedings of the 8th National Conference on Artificial Intelligence*, pp. 17-24.

Panwalkar, S.S. and Iskander, W. (1977) A Survey of Scheduling Rules, *Operations Research* **25** (1), pp. 45–61.

Stohl, K., Snopek, W., Weigert, T., and Moritz, T. (1991) Development of a Scheduling Expert System for a Steelplant, *Proceedings of the IFIP Conference on Experts Systems in Mineral and Metallurgy*, Finnland.