Comparison of Iterative Improvement Techniques for Schedule Optimization*

Jürgen Dorn, Mario Girsch, Günther Skele, Wolfgang Slany

Christian Doppler Laboratory for Expert Systems
Technical University of Vienna
Paniglgasse 16, A-1040 Vienna, Austria
Email: {dorn | girsch | skele | slany}@vexpert.dbai.tuwien.ac.at

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 compare the performance of these techniques, we use the same evaluation function, knowledge representation and data from one application. The evaluation function is defined on the gradual satisfaction of explicitly represented domain constraints and optimization functions. The satisfactions of individual constraints are weighted and aggregated for the whole schedule.

We have applied these techniques on data of a steel making 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 antagonistic criteria that are partly contradictory.

Keywords: Scheduling Theory; Optimization; Tabu Search; Genetic Algorithms

---

* The reported work was partly funded by the Austrian Industries in the framework ot the Christian Doppler Laboratory for Expert Systems

1 Introduction

Many years of research in scheduling were dominated by the search for techniques to achieve optimal schedules. Especially for flow shops some hoped to find 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 also for flow shops with a number of jobs and machines typical for real applications, the problem is intractable in most cases.

Consequently, problem solving methods often apply heuristics to find satisfying 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.

Recently, some new scheduling procedures that are based on the idea to improve an existing schedule iteratively were introduced. A schedule is changed by simple inexpensive modifications and then evaluated anew. If this new schedule is better than the old one, this procedure is repeated. This iteration performs a search for a good schedule that can be interrupted anytime. Additionally, such an iterative improvement method can be combined with a repair meta-heuristic. Minton *et al.* (1990) have shown that their *MinConflicts heuristic* that tries to repair the greatest constraint conflict is an approach that helps to master the inherent complexity in large problems like the Hubble Space Telescope problem.

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 problems there are usually many local maxima and therefore the search could be stuck in such a local maxima. Some experiments 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 cleverer selection of a modification can have considerable costs in terms of computation time. Another explanation is that, with such a randomized modification, local optima may be escaped more easily. 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, 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, because the advantage of these stochstic techniques in contrast to deterministic techniques shall be shown. It seems to be widely acknowledged that these stochastic techniques perform better for applications of realistic size, but it is not clear which one of these methods is best. We compare some of the iterative improvement techniques against each other. We have used an application in a steel making plant as a benchmark example. This example gives also new insights in how one can adapt iterative improvement techniques in a different domain because the steel making application differs considerably from the typical examples in the literature. Our examination is motivated by a project for which we want to apply the best approach. Of course, the results of our examination should not be understood as a general rule which of these techniques should be applied because results may depend on the problem type and also on our representation. We will discuss the results in our conclusions.

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 steel making 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 (jobs) is scheduled for one caster. Hot pig iron is delivered from the blast furnace to the LD-converters. The steel is then poured into ladles, processed further in secondary metallurgy aggregates and finally delivered in time to the casters. The casters produce continuously strands of steel that are cut into slabs of specific length. Although the steel is produced in different aggregates in a fixed flow, the casters are the bottleneck resource of the shop. The tundish, a part of the caster, has to be maintained after approximately 240 minutes. The maintenance takes about 100 minutes, therefore a second tundish is used while the first one is maintained. Before one tundish is worn out the tundishes are exchanged to allow a continuous operation of the caster.

The main criteria that constrain the sequence of charges are compatibility constraints between jobs. 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 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 two steel grades are too different, no *quality separation* is possible.
- A *change of tundishes* may also 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 lower.
- The most expensive operation is a new *set-up* of the caster. This means several hours of interruption, without casting. Set-ups are required when an interruption in the delivery occurs, or when the duration between two changes of tundishes is less than 100 minutes.

Besides compatibility constraints, due date constraints have to be satisfied. These constraints are posted to enable a direct rolling of the cast steel. Therefore finishing too early is also suboptimal. However, due date constraints are relaxable. In general, it is not difficult to find a feasible schedule because as a last resort a set-up of the caster 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 have to be minimized.

In a first expert system approach, Stohl *et al.* (1991) applied a constructive domain heuristic to the problem. Groups of similar compatible jobs are generated and scheduled together. Although the system finds good feasible schedules, Stohl *et al.* believe that these schedules can be further improved. 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 the described application. These techniques all start with a randomly generated preliminary schedule and construct by iteratively modifying a near optimal schedule. Whether the optimal schedule is found cannot be decided. We have also evaluated a combination of these algorithms with the constructive domain heuristic by initializing 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 point of view of the iterative improvement techniques, only a sequence of operations on the caster is modeled. If a conflict occurs on another machine, it 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 as 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 between 0 and 1, whereby 0 means a violation of the constraint, 1 signifies full satisfaction of the constraint, and intermediate values means partial satisfaction of the constraint. We believe that for a better modeling of this satisfaction degrees, fuzzy sets are appropriate as proposed in (Dorn and Slany 1994). However for the sake of simplicity we restrict the expressiveness in this paper, and model the satisfaction degree of constraints by real values. 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 strictest constraints in the application, their consistency is achieved through insertion of a quality separation, a tundish change, or a set-up. We obtain for a given schedule a number of constraint evaluations 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

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

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

The value of this evaluation function is always between 0 and 1. The optimal value is theoretically 1, but when different steel grades are involved, this value can never be reached. Good evaluations for the data that we have used range between .91 and .93.

Our technique of representing constraints and their satisfaction explicitly, makes it easy to decide where the greatest constraint violation exists. All our improvement techniques apply the heuristic to focus on this greatest conflict and to repair this constraint violation.

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 complete, but preliminary schedule that is improved in several iterative steps. In principle, such an iterative improvement technique can be trapped in a local optimum. To escape, schedules with a worse evaluation have to be sometimes accepted as candidates for further search. The problem solving techniques that we have tested and that will be described in the following, differ how they escape from such local optima. Because these algorithms were developed independently, their descriptions are not so coherent that they can be compared easily. We change here the description slightly without changing their behavior and give in algorithm 1 a general framework for an iterative improvement method.

The algorithm consists of a main loop. In each iteration the schedule is improved until some stopping criterion is reached which may be the time consumed. The search from one intermediate best schedule to the next schedule that becomes the new best schedule is described here as a nested iteration. We call the modification from a currently best schedule to a better schedule a *repair step*. A repair step can consist of several simple schedule modification steps. The Boolean function “acceptable” in algorithm 1 is the main difference between the different techniques that will be explained later. Genetic Algorithms have a further difference to the other approaches because they maintain several schedules concurrently.

```
iterative_improvement;
  S := initialSchedule();
  repeat {iterative improvement}
    Sbest := S;
    repeat {improvement}
      S' := modify(S);
      if acceptable(S', S)
        then S := S';
    until better(S, Sbest);
  until stopping_criteria
```

Algorithm 1: General iterative improvement algorithm

In each of the presented techniques a lot of details can be adjusted to change their performance. For example, the procedure how a schedule is evaluated may have influence on the performance and furthermore, one evaluation technique may fit better to one search technique and another better to another search technique. Our approach to compare the methods relies on the invariance of these possibilities. To facilitate a comparison we have applied all algorithms to the same initial schedules with 35 jobs, used the same evaluation function, the same modification operators and the same stopping criteria.

We stop every algorithm after 300 seconds run time (The run times cited in this paper were measured on a 386 PC). This is possible because these 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). Furthermore, “good” schedules are usually found relatively early. Experiments with very long search times have brought only very small improvements.

The applied modifications are similar in all approaches. We assume that the schedule is a sequence of jobs whose order is modified only. Hard constraints (compatibility constraints) that are violated by such a modification are automatically repaired through the insertion of necessary quality separations, tundish changes or set-ups. To satisfy due dates 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	123**4**56789 → 1235678**4**9
two jobs are exchanged	12**3**4567**8**9 → 12**8**4567**3**9
a group of jobs is shifted	1**234**56789 → 1567**234**89
two groups are exchanged	1**23**45**678**9 → 1**678**45**23**9

By means of our evaluation function that is defined on single constraint satisfaction 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 most of the described algorithms some decisions are generated randomly. Therefore, we have performed for each variant a series of 10 measurements. In all experiments we use the same data set, but we start with different schedules. We have one preliminary 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 sequencing the groups with a domain heuristic, and due dates are satisfied, too.

4.1 Iterative Deepening

We have designed a new deterministic search method that was inspired by the *iterative deepening* approach of Korf (1987). We have deveveloped this method to evaluate whether stochastic techniques are necessary to find good solutions. Iterative Deepening seems to be the best approach to search deterministically for a repair. It is a combination of depth-first backtracking search and breadth-first search and has the advantage that it has only linear space requirements and guarantees to find the shortest path to an improvement.

In iterative deepening, one searches in a first attempt to depth 1, this means we try out all modifications and decide whether an improvement was achieved. If no improvement is found, a depth-first backtracking search to depth 2 is performed in a second attempt. This means, we select one modification operator, apply it to the schedule and test then all operators that can be applied to the new schedule. The depth limit is incremented iteratively until an improvement is found. On each depth level all generated schedules are evaluated. Only modifications that are directly related with the greatest conflict of the actual schedule are tried thus reducing the neighborhood. The following figure illustrates the search tree for an improvement. Here an improvement is found after two calls to the backtrack procedure sketched next.

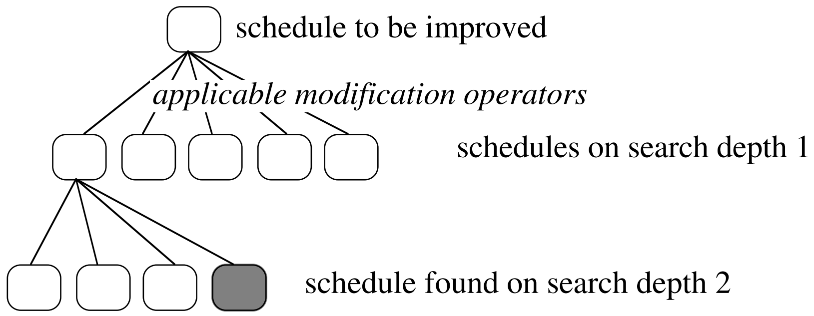

Figure 1: Iterative Deepening Algorithm

```
backtrack(Depth, MaxDepth, Limit, S, Sbest)
  Ops = applicableOperators(Schedule);
  if Depth=MaxDepth
    then Sbest = bestNeighbor(Ops);
    else for i=1 to NumOps do
    {
      S' = apply(Ops[i], S);
      backtrack(Depth+1, MaxDepth, Limit, S', Sbest);
      if better(Sbest, Limit) return;
    }
```

Algorithm 2: Backtrack Algorithm to be called to improve Schedule

In each iteration, first the operators that move single jobs involved in the greatest constraint violation are applied. 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 jobs between two tundish changes or jobs between a quality separation and a tundish change.

We distinguish two versions of the iterative deepening algorithm. The first version tries to repair the greatest constraint violation by means of operators that move single jobs 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 another version that tries 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 variants whereby the deterministic version is labeled with 0. However, we found that this incorporation of randomness brings no improvement in the solution quality or in the required time, as shown in Fig.2.

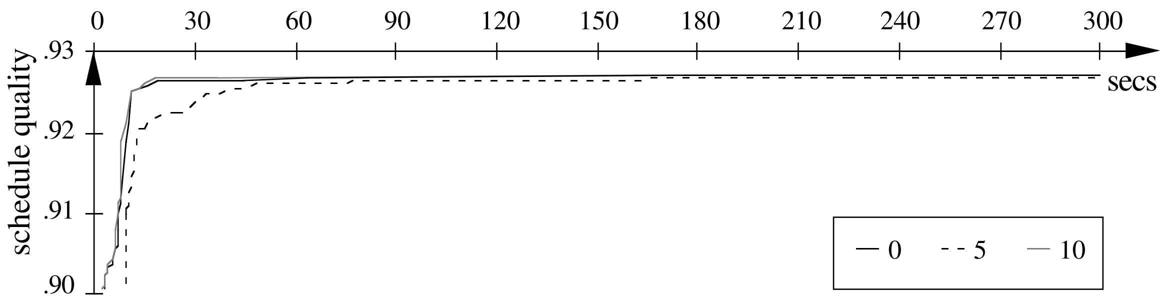

Figure 2: Performance of three variations of iterative deepening

The following diagram contains the performances for the three preliminary schedules with the deterministic version of the iterative deepening approach. For all schedules a very good evaluation was found after about 30 seconds and this solution is in each case better than the solution of the original expert system. The value of .906 that was achieved by the original expert system was reached for the preliminary schedule 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 was found with schedule B in 72 seconds with a value of .92725. We see that the goodness of the initial schedule has a relatively small influence on the solution if there is enough time to search.

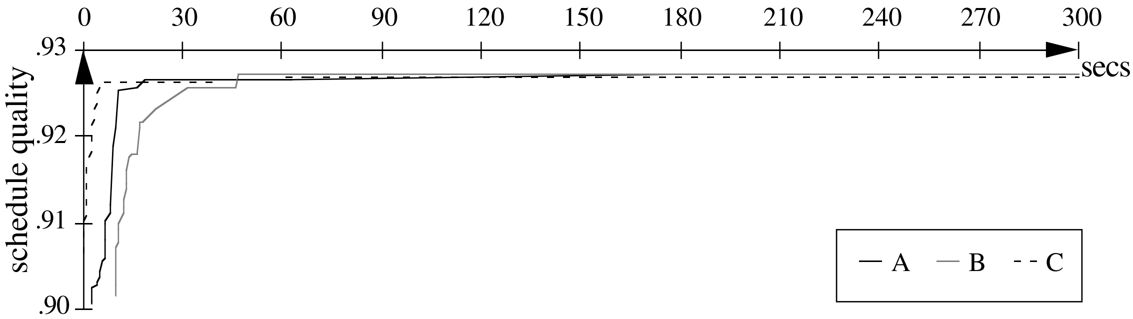

Figure 3: Performance of iterative deepening

Another remark that can be drawn by observing Fig. 2 and 3 is that “good” solutions are found relatively early (after 40 seconds), and even “better” solutions differ only minimally from these, while taking a lot more time to find them. However, the question remains whether a “much better” solution, having for example a score over .94, exists but represents the proverbial needle in a haystack, which would imply that the presented algorithm remain stuck in local maxima.

4.2 Random Search

The iterative deepening approach tests deterministically all nodes of the search tree and is theoretically guaranteed to be exhaustive if time allows. However, this can be very time consuming if many levels of the search tree must be examined. Also the number of neighbors may be very large. In our experiments the number of neighbors 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, another modification is generated randomly. The advantage in contrast to the iterative deepening algorithm is that not all successors of a schedule must be evaluated. In our series of ten runs starting with schedule C we have found one schedule that has a better value (.92895) than those schedules that were found by iterative deepening. The average measurements for the three initial schedules are shown in the following figure. We see that the initial schedule has a greater influence on the solution than for the iterative deepening approach.

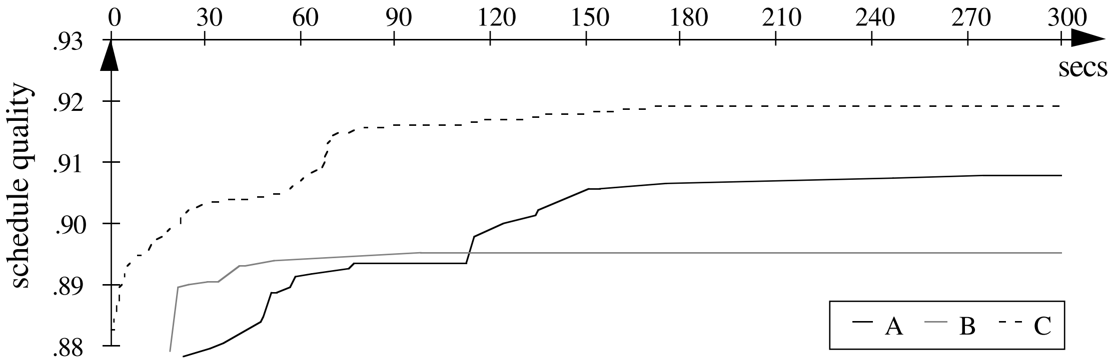

Figure 4: Performance of random search

However, the problem with this approach is its large 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.

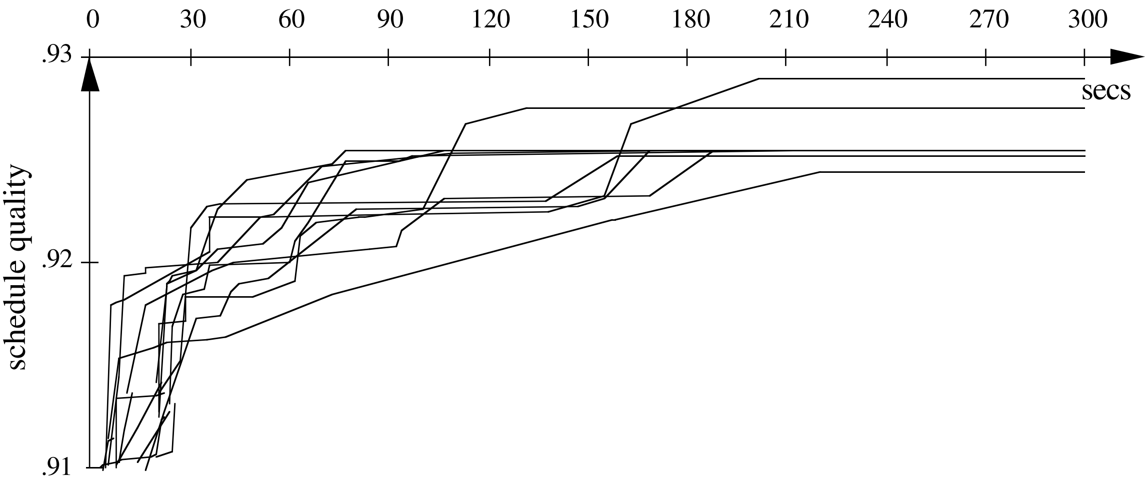

Figure 5: Variance of random search with preliminary schedule C

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 because this length has brought us the best results.

We first try to "repair" the greatest constraint conflict and stop the search after 50 iterations. If no improvement is found in 50 steps, the search is started again with an arbitrary constraint violation. 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.

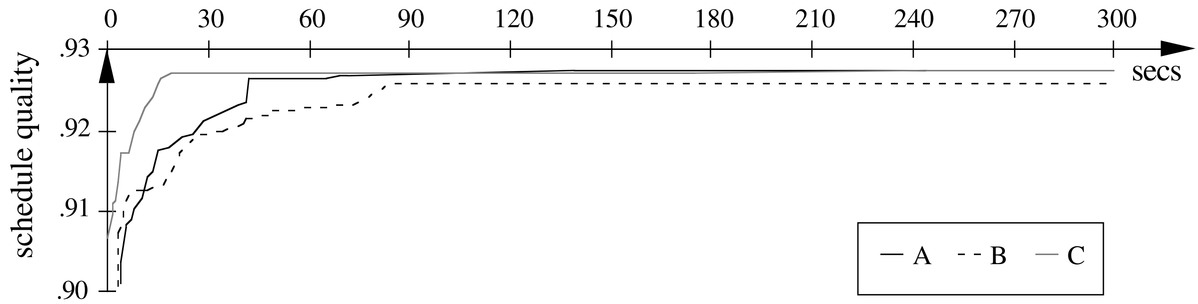

Figure 6: Performance of tabu search

The following figure shows the performance of 10 runs with schedule A. The achieved score varies between .926 and .928. From this variance we see that sometimes tabu search has not found an improvement after 50 modifications. A stronger diversification by a random selection of a modification was necessary.

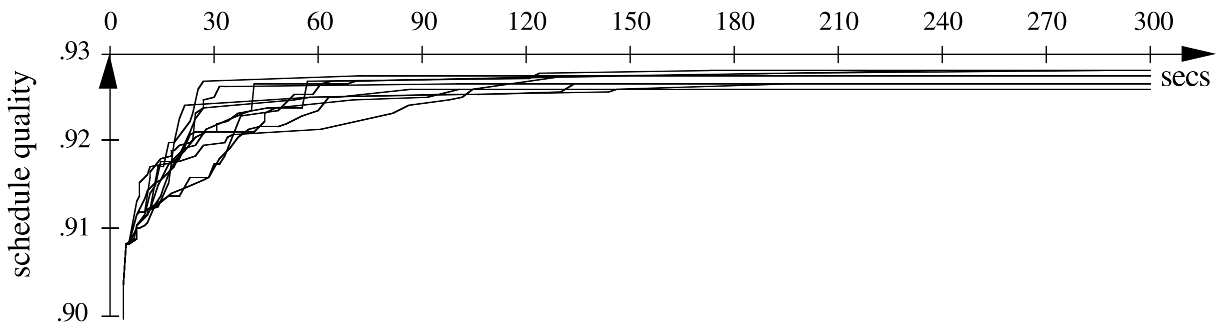

Figure 7: Variance on the performance for Tabu Search

We have performed a second set of experiments with tabu search whereby the best modification is not always 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.

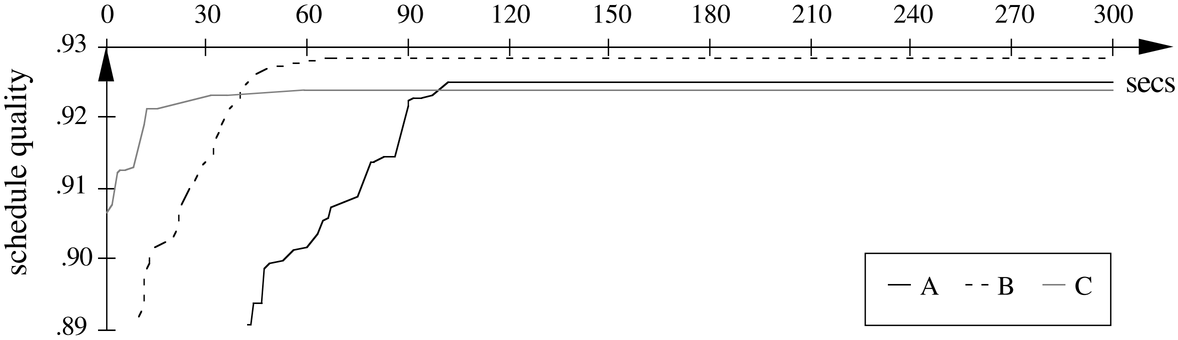

Figure 8: Performance of second version of tabu search

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 a selection mechanism. Genetic operators such as crossover 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. Although some Genetic Algorithms were designed for scheduling applications with such bit strings (e.g. Fox and McMahon 1991), 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 (Lawton 1992) and (Fox and McMahon 1991) several operators are presented.

We have experimented with some of them and found out that for our application a crossover operator that takes groups of jobs from one ancestor suits best. We apply a crossover operator that selects randomly two positions p_1 and p_2 in one chromosome and exchanges the genes between both positions with that of the second chromosome. For example, if we have the following ancestors 1,2,3,4,5,6,7,8 and 2,5,7,1,8,6,4,3 and we select the positions $p_1 = 2$ and $p_2 = 5$, then the resulting child will get the genes 5,7,1,8 of the second chromosome and is filled up with genes from the first chromosome resulting in a sequence 2,**5,7,1,8**,3,4,6. Our mutation operator simply selects two positions of a chromosome randomly and exchanges both genes.

In our experiments we start with a population of 30 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 10 chromosomes are deleted. 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. A mutation operator is afterwards applied to the children. The result of these operations is that we have again 30 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.

As we use the “repair”-heuristic to guide search in the other approaches, we have experimented 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 applied a probabilistic selection of positions in the crossover and mutation operator. The probability to mutate two genes is greater if one or both of them are involved in a bad constraint satisfaction. We have made three sets of measurements for schedule A. Their average performance is shown in the next figure:

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

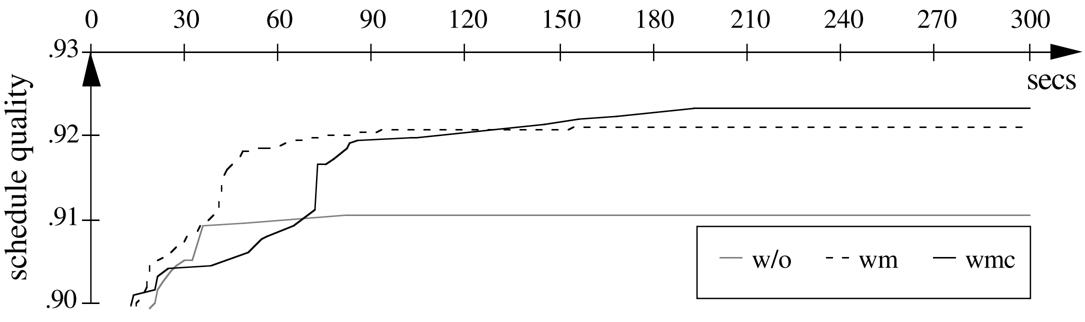

Figure 9: Performance of genetic algorithms

We see that the Genetic Algorithm enhanced by the probability model achieves a better performance. 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.

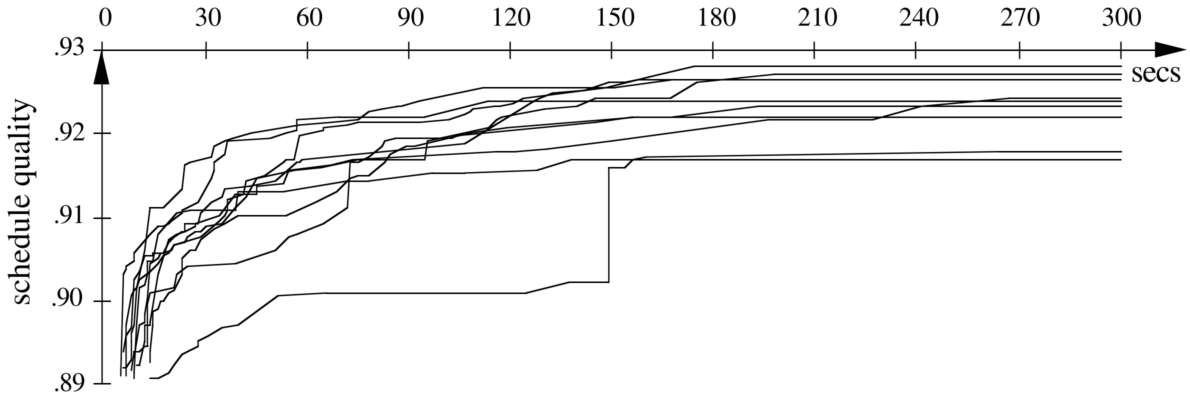

Figure 10: Variance for 10 runs on Schedule A with enhanced Operators

Additionally, we have performed experiments by initializing the population with schedules found by the domain heuristic. The experiments have shown that a better initialization has influence on the performance of the Genetic Algorithm. The best schedule found with the domain heuristic gets an evaluation of about .906. The curve labelled "C" shows the performance of the Genetic Algorithm when the initial population contains this schedule. Another schedule found with less effort obtains an evaluation of .88 and the initialization of the Genetic Algorithm with this schedule is shown by curve "B" in the following diagram. The curve labelled "A" shows the performance when all members of the initial population are generated randomly. The curve is the same as that labelled "wmc" in Fig. 9.

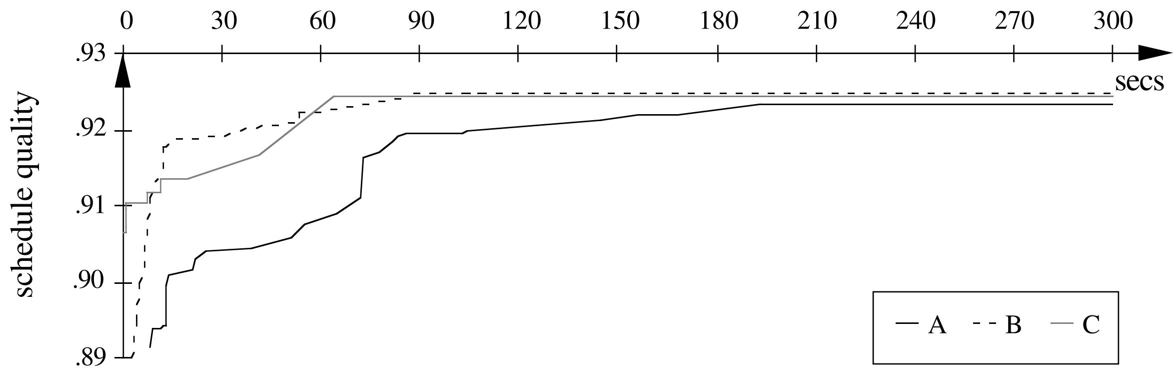

Figure 11: Comparison for different initial schedules

4.5 Comparisons of algorithms

The described experiments were part of two masters thesis (Skele 1994) and (Girsch 1994). They were implemented in C++ on a SPARC-station and on a PC. We have measured the time on a 386-PC because the measurement 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.

We have started these experiments to decide which algorithm shall be used in a project for the steel making plant in Linz. The algorithm shall be used for optimization, reactions on disturbances in the shop, and for interactive scheduling. In principle, all presented algorithms seem to be acceptable. However, the convergence to the found best solution is for iterative deepening and tabu search better than for the genetic algorithm and random search as can be seen in the following figure.

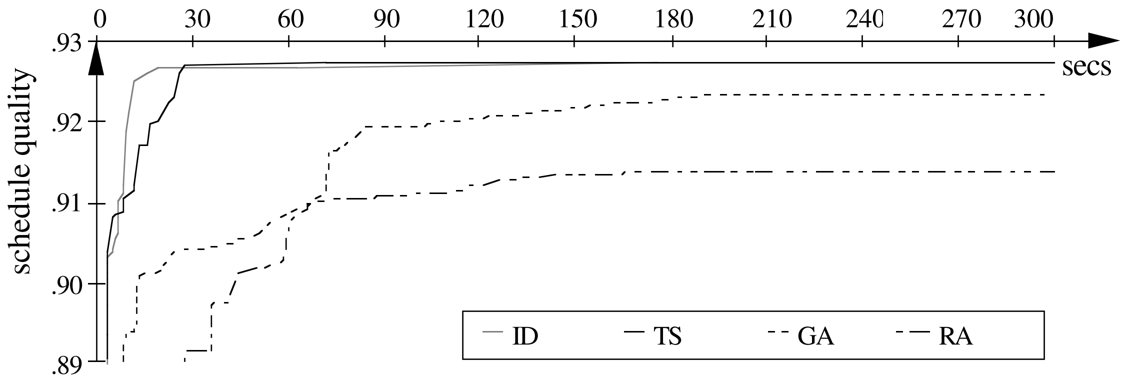

Figure 12: Comparison of four algorithms applied to schedule A

For both algorithms - iterative deepening and tabu search – one could decide to stop the search after 30 seconds. This is not so bad for interactive scheduling and deadlines will not be violated in reactive scheduling. The result of the other two search techniques is not so good, but a possible option would be to use first domain heuristics to construct a first schedule and to optimize this schedule later. The following figure shows the performance for the four algorithms with a good initialization. Nevertheless, tabu search and iterative deepening outperforms the two other algorithms as can be seen in the figure.

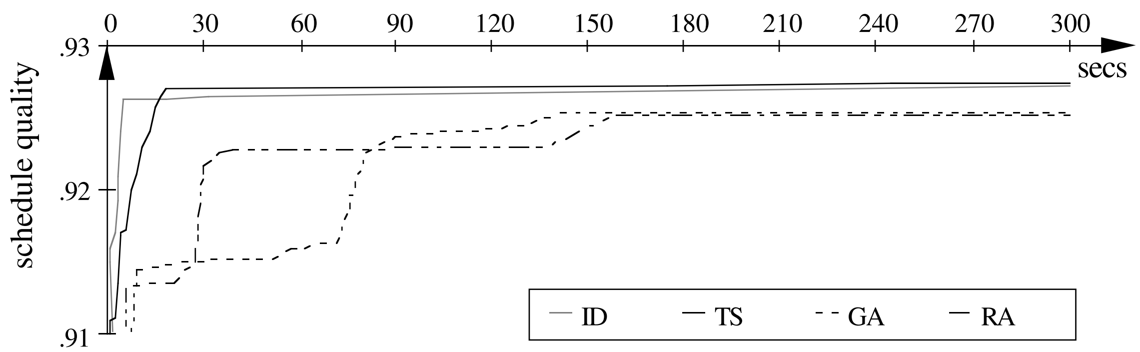

Figure 13: Comparison of the four algorithms applied to schedule C

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 contradictory constraints. This knowledge-intensive application demands a knowledge-based approach with explicit representation of constraints. To cope with the contradictory 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 partially satisfied.

We have defined a framework to compare several improvement techniques. Since these algorithms use the same knowledge representation and evaluation function, a comparison of the performance of the algorithms is more expressive than other comparisons in the literature. From our results one may conclude that for the given application and the described settings iterative deepening and tabu search is best. However, we must be very carefully to generalize these results. Our results rely heavily on the heuristic to repair the greatest conflict in a schedule. This reduces on one hand the number of neighbors and on the other hand focusses the search. Although this heuristic was also an improvement for the genetic algorithm, this heuristic seems to be more important for iterative deepening and tabu search. Since for iterative deepening the number of schedules grows exponential with depth of the search, we believe that the application of iterative deepening is restricted. Either the branching factor or the search depth must be limited. Since we have applied mighty modification operators for our application the maximal search depth was four. A further fact that supports relativ short paths in our case is the fine-grained evaluation function. First experiments have shown that with coarse-grained evaluation longer paths exists because more schedules with the same evaluation exist. Then a technique as simulated annealing or genetic algorithms may be advantageous. A more detailed theoretical investigation on such design decisions can be found in Dorn (1995). In this paper we have investigated further iterative improvement techniques like the shifting bottleneck procedure, threshold accepting and simulated annealing.

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 some conclusions can be drawn from the experiments, but it seems reasonable that some of the algorithms could be improved by a more specific selection of operators. Especially for the genetic algorithm a lot more of experiments to find a better performance seem to possible. However, this shows also a drawback of genetic algorithms. We have spend considerable time in finding the best control settings for the genetic algorithm. Although a genetic algorithm can be developed very straightforward with little effort, a lot of tuning and testing is necessary to achieve a satisfying performance.

For scheduling, Drummond (1993) has listed a handful of benchmark test sites where data can be obtained by ftp or email. However, these examples have only very simplistic structures, and usually only temporal constraints are regarded. Especially knowledge-based techniques were designed to solve complex representational and Fox (1983) asks for knowledge-based solutions 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 steel making application described in (Dorn and Slany 1994) 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. A test that pragmatically checks changes in the evaluation function against reference situations has been described in (Slany 1994). If the system generates a schedule, the expert can say whether this schedule is satisfying 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. Such a simulation environment should support furthermore the testing of interactive scheduling between a user and a system.

6 References

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

Dorn, J. (1995) Iterative Improvement Methods for Knowledge-based Scheduling, *AICOM* Journal, pp. 20–34, March.

Dorn, J., Kerr, R. M. and Thalhammer, G. "Reactive Scheduling – Improving the Robustness of Schedules and Restricting the Effects of Shop Floor Disturbances by Fuzzy Reasoning" To appear in *International Journal on Human-Computer Studies*, 1995.

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.

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, B.R and M.B. McMahon (1991) Genetic Operators for Sequencing Problems, in G.J.E. Rawlings (ed): Foundations of Genetic Algorithms.

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

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

Girsch, Mario (1994) *Optimierung von Schedules mit Genetischen Algorithmen und Iterativer Vertiefung*, Master Thesis, Vienna University of Technology, Institute for Information Systems.

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) *Adaptation 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, J.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.

Skele, Günter (1994) Repair-based Scheduling mit Tabu Search, Master Thesis, Vienna University of Technology, Institute for Information Systems.

Slany, W. (1994) Scheduling as a Fuzzy Multiple Criteria Optimization Problem, Technical Report Christian Doppler Laboratory for Expert Systems CD-TR 94/62.

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.