DIMACS Series in Discrete Mathematics
and Theoretical Computer Science

Using Global Constraints for Local Search

Alexander Nareyek

Abstract. Conventional ways of using local search are difficult to generalize. Increased efficiency is the only goal, generality often being disregarded. This is manifested in the highly monolithic encodings of complex problems and the application of highly specific satisfaction methods.

Other approaches take the general constraint programming framework as a starting point and try to introduce local search methods for constraint satisfaction. These methods frequently fail because they have only a very limited view of the unknown search-space structure.

The present paper attempts to overcome the drawbacks of these two approaches by using global constraints. The use of global constraints for local search allows us to revise a current state on a more global level with domain-specific knowledge, while preserving features like reusability and maintenance. The proposed strategy is demonstrated on a dynamic job-shop scheduling problem.

1. Introduction

The use of local search has become very popular in recent years as applications have begun to tackle complex real-world optimization problems for which complete (refinement) search methods are still not powerful enough. Problem domains with nontrivial optimization functions, large problems, short computation time limits, partial constraint satisfaction, dynamic constraint satisfaction and only limited demands on optimality are promising candidates for local search.

Local search approaches perform a search by iteratively changing an initial assignment of variables. In each iteration, a *neighborhood* of potential successor states is generated. The quality of the neighborhood states can be computed by a *cost function*. This information is used by the *successor choice criterion* to determine the successor state.

Conventional ways of using local search are difficult to generalize. Increased efficiency is the only goal, generality often being disregarded. It is quite common to define highly sophisticated and problem-tailored representations with specialized neighborhoods and successor selection methods (see [**Aarts97**] for examples).

2000 *Mathematics Subject Classification.* Primary 90C27; Secondary 68T20.

This work is supported by the German Research Foundation (DFG), NICOSIO, Cross Platform Research Germany (CPR) and Conitec Datensysteme GmbH.

From a software engineering point of view, this is not a good idea. Integrating complex, heterogeneous problems in a monolithic system hinders the system's reuse, extension and modification.

Other approaches take the general constraint programming framework as starting point and try to introduce local search methods for constraint satisfaction. Problems are formulated in a framework of variables, domains and constraints. A constraint satisfaction problem (CSP) consists of

- a set of **variables** $x = \{x_1, \ldots, x_n\}$,
- where each variable is associated with a **domain** $d_1, ..., d_n$
- and a set of **constraints** $c = \{c_1, ..., c_m\}$ over these variables.

The domains can be symbols as well as numbers, continuous or discrete (e.g., "door", "13", "6.5"). Constraints are relations between variables (e.g., "x_a is a friend of x_b", "$x_a < x_b \times x_c$") that restrict the possible value assignments. Constraint satisfaction is the search for a variable assignment that satisfies the given constraints. Constraint optimization requires an additional function that assigns a quality value to a solution and tries to find a solution that maximizes this value. Through the use of CSP formulations, local search acquires a general application-independent framework.

CSP formulations used with local search approaches typically involve only very basic constraint types, e.g., linear inequalities or binary constraints. The problem with this kind of formulation is that the inherent problem structure is mostly lost by the necessary translation of the problem to this low-level formulation. Domain-specific knowledge about appropriate representations and search control is only available at a higher level and cannot be used. Consequently, these methods frequently fail because they have only a very limited view of the unknown search-space structure.

This paper attempts to overcome the drawbacks of these two local search approaches — monolithic problem-tailored and general low-level — by using global constraints. The use of global constraints for local search allows us to revise a current state on a more global level with domain-specific knowledge, while preserving features like reusability and maintenance. The proposed strategy is demonstrated on a dynamic job-shop scheduling problem.

Local search techniques provide a solution at any time, the quality of the solution being subject to constant improvement. This anytime feature makes it worthwhile evaluating the whole solution process, not just the final result. To have a measure for the utility of intermediate states as well, we can extend the CSP formulation to weighted CSPs (WCSPs). In WCSPs, constraints have associated costs, which are dependent on the constraint variables' assignments. The goal is to minimize the total sum of costs[1], which provides us with a useful cost function for local search. A value of zero for the costs means total satisfaction. The graded consistency measure makes it possible to extend the applicability of the concept to over-constrained systems and real-time reasoning.

2. Global Constraints

Constraint satisfaction has traditionally been tackled by refinement search (domain-reduction techniques), which faced similar problems of exploiting high-level knowledge. The success of commercial tools like the ILOG Scheduler [**LeP94**] can

[1]The term *inconsistency* is used below as a synonym for *costs*.

be ascribed mainly to their use of so-called *global constraints*. Régin [**Rég98**] defines a global constraint as a substitute for a set of lower-level constraints, such that a more powerful domain-reduction algorithm can be applied. Using global constraints, constraint solvers can attain an enormous speedup and real-world application requirements can be satisfied.

For example, the `alldifferent(V)` constraint is a global constraint that forces all included variables of the set `V` to take different values. A formulation by lower-level constraints would include inequality constraints for all possible variable pairs of the set `V`. But the application of the global constraint with a specific data representation and satisfaction methods can yield much better performance (e.g., by a demon-observed array representation [**PL95**]).

The notion of global constraints can support local search approaches as well. It is transferred to a local search context in the following subsections.

2.1. Global Constraints from a Local Search Perspective. The central issue in local search is the transition from one state to the successor state. As it is uncertain what kind of change improves a current state, lots of neighbor states are usually analyzed. There are no general rules here, the kind of neighborhood and successor selection being a heuristic matter.

The term *heuristic* already implies the existence of some *domain knowledge* for guidance. The heuristics of conventional local search mechanisms for CSPs, like GSAT [**Gu92, SLM92**], can only exploit knowledge about their representation's special low-level structure, thus having to cope with self-produced complications instead of being able to incorporate higher-level domain knowledge. It is well known that there is a strong relation between a problem's representation and its computability, and it remains unclear to what kind of problems these low-level standardized representation approaches are suited. Consequently, there is often not enough information locally available to direct the search [**MJPL92**]. Larger variable domains than the binary ones normally used exacerbate the problem because information about qualitative differences is not directly available.

The concept of global constraints can help remedy this situation. To this end, we have to extend the notion of a global constraint:

> A global constraint is a substitute for a set of lower-level constraints, additional domain knowledge allowing the application of specialized data representations and algorithms to guide and accelerate search.

A global constraint must feature a start function to construct its initial structures and inconsistency, an improvement procedure for determining a promising successor state, and an update function for its internal structures and the constraint's cost contribution in case of variable changes.

Heuristics for building neighbors and for selecting a successor can be directly encapsulated in the global constraints, where the appropriate domain knowledge is available. This is illustrated in the following sections using an example of a global constraint PERMUTATION. The global constraint PERMUTATION(A, X) has to ensure that the variables of set X are a permutation of the values of bag A. The costs of the constraint are to be determined by the minimal sum of the distances of the variables' current values to a valid assignment.

2.2. Internal Structures. Many successful applications of local search gain their power from sophisticated updates of the inconsistency information rather than from recomputations from scratch. For this purpose, *additional structures* often have to be maintained. In the case of the PERMUTATION constraint, three lists can be used as additional support structures: list l_a with the ordered values of bag A, list l_x with the variables of X ordered by their current assignments, and list l_c with the ith element being the distance between the ith value of l_a and the value of the ith variable of l_x. The costs of the PERMUTATION constraint can be computed by summing the values of l_c.

For PERMUTATION(<1, 7, 4, 3>, {A, B, C, D}) and a current assignment of A=2, B=6, C=2 and D=7, the structures are constructed in the following way:

$$\begin{aligned} l_a &= [1,3,4,7] \\ l_x &= [A_{(2)}, C_{(2)}, B_{(6)}, D_{(7)}] \\ l_c &= [|1-2|=1, |3-2|=1, |4-6|=2, |7-7|=0] \\ Permutation_{costs} &= 1+1+2+0=4 \end{aligned}$$

Note that by changing a variable all global constraints involving this variable must be called to update their internal structures. For the PERMUTATION constraint, there must be a reordering of the variable in l_x with a corresponding update of l_c and the resulting cost sum. For example, if B changes from 6 to 9,

$$\begin{aligned} l_a &= [1,3,4,7] \\ l_x &= [A_{(2)}, C_{(2)}, \underline{D_{(7)}}, \underline{B_{(9)}}] \\ l_c &= [1,1,\underline{|4-7|=3}, \underline{|7-9|=2}] \\ Permutation_{costs} &= \underline{4+(-2+3)+(-0+2) \;=\; 7} \end{aligned}$$

2.3. Improvement Heuristics. What is a good heuristic for building the successor state may depend on the other constraints involved as well as on the current state of the search. Thus, a global constraint should provide various heuristics, e.g., one with a complete revision of the violated variables, one with a minimal change of just one variable, one with a randomized successor selection, and one with random walks.

For the PERMUTATION constraint, the list l_c gives us quantified information about the variables' violation of the constraint. A heuristic with a cautious strategy can reset only one variable — according to the highest element in l_c — to the corresponding value of l_a. Following the last assignment of A=2, B=9, C=2 and D=7, the variable D would be changed to 4:

$$\begin{aligned} l_c &= [1,1,\underline{3},2] \\ l_a &= [1,3,\underline{4},7] \\ l_x &= [A_{(2)}, C_{(2)}, \underline{D_{(4)}}, B_{(9)}] \\ Permutation_{costs} &= \underline{7+(-3+0) \;=\; 4} \end{aligned}$$

The range of heuristics to be applied to a specific problem should be customizable by the user. During search, a constraint-internal function has to decide on the concrete heuristic to be applied. This decision can be taken at random, be dependent on the current search state, or be subject to learning mechanisms.

3. Granularity

If the PERMUTATION constraint is to be modeled by low-level constraints, linear inequalities can be used. A particular problem, e.g., PERMUTATION(<1, 7, 4, 3>, {A, B, C, D}), can be expressed by the following CSP:

$$\begin{aligned} A, B, C, D &\in \{1, 3, 4, 7\} \\ \forall n \in \{1, 3, 4, 7\}: \; A_n, B_n, C_n, D_n &\in \{0, 1\} \end{aligned}$$

$$\begin{aligned} A &= A_1 + 3 \times A_3 + 4 \times A_4 + 7 \times A_7 & \qquad C &= C_1 + 3 \times C_3 + 4 \times C_4 + 7 \times C_7 \\ B &= B_1 + 3 \times B_3 + 4 \times B_4 + 7 \times B_7 & \qquad D &= D_1 + 3 \times D_3 + 4 \times D_4 + 7 \times D_7 \end{aligned}$$

$$\begin{aligned} A_1 + A_3 + A_4 + A_7 &= 1 & \qquad C_1 + C_3 + C_4 + C_7 &= 1 \\ B_1 + B_3 + B_4 + B_7 &= 1 & \qquad D_1 + D_3 + D_4 + D_7 &= 1 \end{aligned}$$

$$\begin{aligned} A_1 + B_1 + C_1 + D_1 &= 1 & \qquad A_4 + B_4 + C_4 + D_4 &= 1 \\ A_3 + B_3 + C_3 + D_3 &= 1 & \qquad A_7 + B_7 + C_7 + D_7 &= 1 \end{aligned}$$

The permutation problem is very hard to recognize now. And not even the global constraint's distance measure is included in the upper solution. In contrast to the low-level formulation, the statement of a global PERMUTATION constraint is highly declarative and easy to understand.

A local search method that is based on a low-level problem representation cannot make use of domain knowledge and is not able to recognize the low-level constraints' interactions, e.g., that it is inadvisable to compensate a change from $A_7 = 1$ to $A_7 = 0$ with respect to the constraint $A = ...$ by an activation of $A_3 = 1$ and $A_4 = 1$ in order to keep the constraint satisfied. The lack of a general overview and of heuristic knowledge makes the low-level approach less efficient and susceptible to cycling and getting stuck in local minima.

A low-level representation also has advantages, though. A wide variety of problems can be modeled using the general low-level representation, whereas modeling using global constraints presupposes the availability of suitable global constraints. For example, if the problem is to find an assignment such that the variables of set X are a permutation *of a subset* of the values of bag A, the low-level representation can easily be adapted. But a solution based on the global PERMUTATION constraint would require a considerable effort to adapt the constraint's internal structures and heuristics, if not a redesign from scratch[2].

A comparison of global constraints and monolithic solutions is straightforward, regarding the global constraints as a low-level representation and the monolithic solutions as a single highly global constraint. Again, the higher-level (monolithic) system is faster as it makes better improvement decisions because of its superior overview, but it is difficult to reuse because of its specialization.

Global constraints are a compromise between the generality of low-level CSP-based local search and the efficiency of monolithic problem-tailored local search encodings (see Fig. 1). Finding the *right* granularity for global constraints is up to

[2]However, all low-level constraints could be realized by global constraints as well, enabling the user to model the problem by these constraints if no high-level constraint is available

the designer. Only very general rules apply, which are comparable to the problems of object-oriented design (see [**GHJV95**] for a discussion and general guidelines).

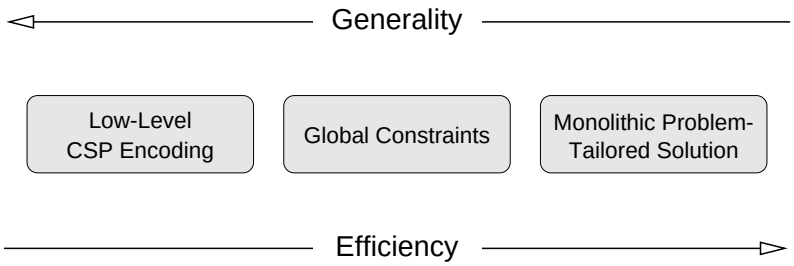

FIGURE 1. Global Constraints as a Compromise

4. Global Search Control

As described above, global constraints have integrated heuristics to enable them to choose a successor state on their own. On top of the constraints, there must be a mechanism that combines the constraints' cost contributions to the overall cost function and a regulation determining which constraint is allowed to select the successor state for a current iteration. This is the job of the *global search control*. The components' interplay is outlined in Figure 2.

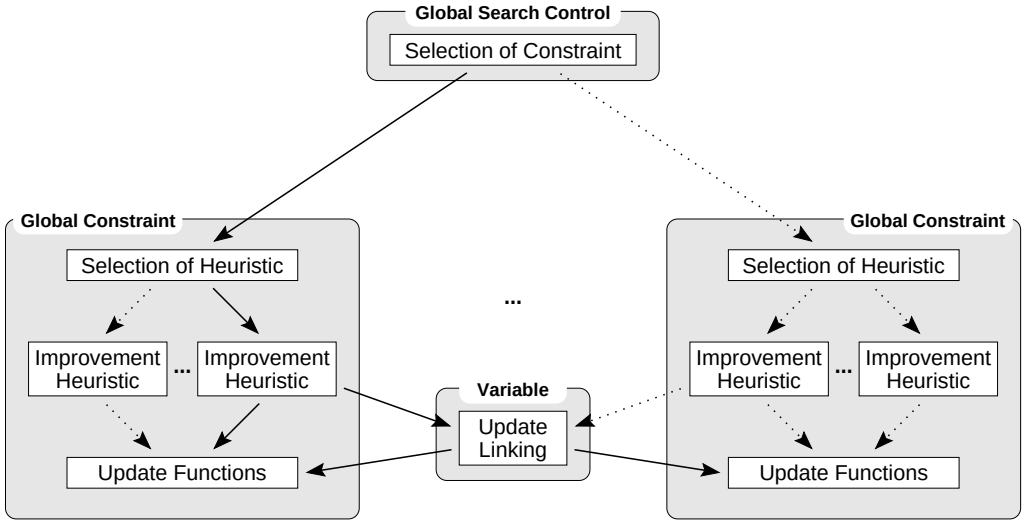

FIGURE 2. Local Search with Global Constraints

The global search control serves as a general manager, where variables and constraints can dock on and off in a plug-and-play manner. This provides a simple mechanism for tackling problems with dynamic changes. For an addition of a new variable, the variable to be added must already be instantiated. For an addition of a constraint, the constraint's inclusion of a variable must be announced to the variable to allow the constraint's updating in the case of value changes.

For the overall cost function, which is handled by the global search control, problem-specific coefficients can be introduced to weight the constraints' subjective costs, as a constraint's importance may be different in different contexts (see also Section 5.3).

Conventional local search methods make a successor decision on the basis of a calculation of the neighborhood states' costs. These calculations might be quite costly to compute within the global constraint framework, but the current constraints' costs can serve as an excellent compensation for this. As the goal is to minimize the constraints' costs and as the global constraints should know how to resolve the conflicts by themselves, it is a straightforward matter to select constraints according to their current inconsistency.

The selection of the global constraint that is to improve the current state can be enhanced by various meta-heuristics to avoid local minima and plateaus, ranging from a simple random choice of unsatisfied constraints to more elaborate techniques including learning, tabu lists, etc. Some of these are demonstrated in the following section's case study. The global search control module should support a variety of methods that can be applied in a user-specified way (as in the flexible blackbox system [**KS98**]). By integrating global search-state parameters for the constraints' improvement procedures, e.g., a simulated-annealing-like temperature, an anytime improvement depending on the current search situation can be achieved. In addition, it is possible to introduce higher-level coordination mechanisms between constraints to minimize violations of multi-constraint variables. If knowledge is available about these interface areas, it may be sufficient to introduce redundant global constraints to cover these areas. Nevertheless, the overhead of coordination mechanisms may pay off in some cases.

5. A Case Study: Dynamic Job-Shop Scheduling

As an example of the application of local search with global constraints, we look at the dynamic job-shop scheduling problem.

The problem of solving standard job-shop scheduling by local search has been addressed in numerous research papers (see [**VAL94**] for a survey), but there are few experiments dealing with a partial satisfaction/infeasible state neighborhood. Experiments including dynamic aspects of job addition/removal are also rare. This is a pity since dynamics and partial satisfaction are very realistic problem features and local search approaches can bring their advantages to bear in these domains particularly. Unlike refinement/global search, local search is not normally affected by modifications of the search space because of its local focus, and it supports integration of partial satisfaction because it has an inherent cost function that is iteratively improved toward full satisfaction/optimization.

As in traditional $n \times m$ job-shop scheduling, there are n jobs in a schedule, each of them having m tasks with fixed durations. All tasks of a job are connected via $m - 1$ linear distance inequalities involving start or end time points of two tasks. For compliance with the scheduling horizon h, there is in addition a linear distance inequality $task_{end} \leq h$ for each task. Each task has to be processed on a specific machine, and each of the o machines can process only one task at a time. Every p microseconds of computation time, one job is removed from the schedule and a new job must be added. The tasks of the new job are randomly distributed within the scheduling horizon.

The goal is to find a concrete begin/end for all currently active tasks, such that the inconsistency of the constraints is as low as possible. To compare different algorithms, the inconsistency can be displayed over time (see Fig. 3; peaks occur on job removal/addition) and the average inconsistency can be computed.

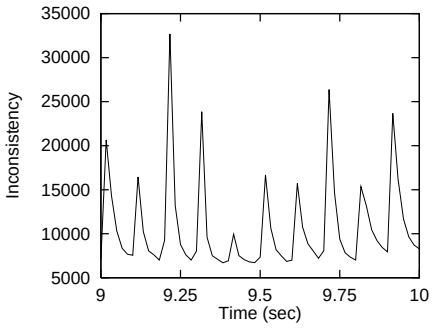

FIGURE 3. A Test Run Example

The measurement of inconsistency is done in the following way:

- For each discrete point of time from 0 to h on each machine, the *number of assigned tasks* -1 is added to the inconsistency if there is an assignment of more than one task.
- For each linear distance inequality that is unsatisfied, the *minimal shift distance* for one of the inequality variables required to satisfy the inequality is added to the inconsistency.

The initial state for the test runs is computed by the iterative addition of n jobs, with p microseconds of computation time between each addition, which is used for improvement iterations.

The following parameters were used for the test runs throughout our experiments: The tasks' duration is 100 plus a random value of -99 to +100. 40 % of a job's tasks require the same machine as another of the job's tasks. The jobs' inequalities consist of 20 % equations and 80 % real inequalities. The inequalities involve a shift constant, which is 100 plus a random value of -99 to +100.

5.1. Realization. The dynamic job-shop scheduling problem was encoded according to the global constraint concept[3]. There are two types of global constraints:

- ACTION RESOURCE CONSTRAINTS (ARCs) for the nonoverlap of tasks that require the same machine
- TASK CONSTRAINTS (TCs) for the temporal ordering relations between tasks within a job

Decomposition into these types of constraints is done because of the strong dependencies among the variables for each constraint and the manifold reuse possibilities. Resource constraints that hinder a temporal overlap of activities/assignments are needed in many applications, e.g., to model the availability of a person, a room or a machine, and the TC's temporal relations of tasks/activities are needed in nearly every system that involves temporal reasoning. Another application of the ARCs and TCs can be found in the EXCALIBUR agent's planning system [**Nar00**].

[3]The implementation was done in ObjectC.

5.1.1. *Global Action Resource Constraints.* An ARC is connected to a set of pairs P and two variables c and h. A pair of the set P, (s_i, e_i), represents a task i that uses the machine/resource. The starting time of a task i is represented by the variable s_i and the end of the task by the variable e_i. The ARC's role is to prevent overlapping tasks. For each discrete point of time from c to h, the *number of assigned tasks* -1 is added to the constraint's inconsistency if there is an assignment of more than one task at this time point. The constraint can affect the tasks' temporal distribution by changing the variables s_i/e_i.

An ARC's basic internal structures are a set of task objects and a multiple-linked list of temporal intervals. A task object contains references to the start variable, the end variable and its intervals. The intervals split the time from c to h into maximal parts such that each interval has the same task assignment for its duration (see Fig. 4). Task overlaps are mapped to the intervals, and the inconsistency of an interval is computed by the overlaps multiplied by the interval's length. Links to an interval's task objects are also stored. The ARC's total inconsistency is the sum of the intervals' inconsistencies.

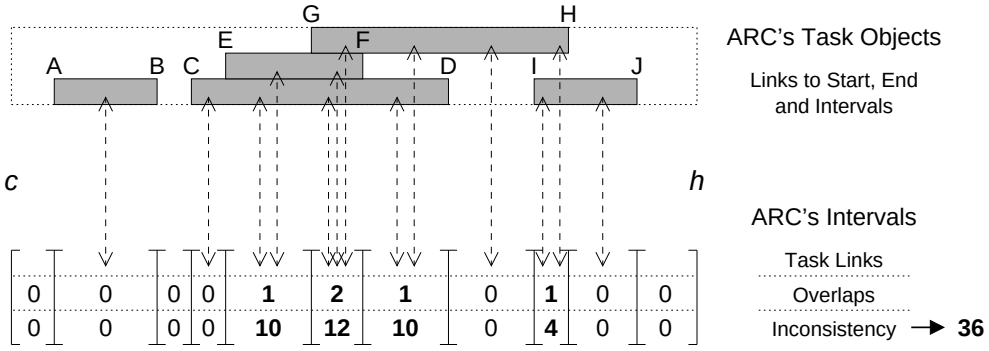

FIGURE 4. An ARC's Internal Structures

The ARC's basic heuristic (ARC-H1) selects an inconsistent interval (i_1; see selection a) of Figure 5) with a choice probability for an interval that is proportional to the interval's inconsistency. For example, if the intervals A to E have inconsistencies of $A_{costs} = 10$, $B_{costs} = 12$, $C_{costs} = 10$, $D_{costs} = 0$ and $E_{costs} = 4$, the chance of being chosen is 27.8 % for A, 33.3 % for B, 27.8 % for C, 0 % for D and 11.1 % for E.

Then, one of the interval's tasks (t_1) is chosen at random. The task's start variable is shifted to the beginning of an interval i_2, a choice probability for the interval being proportional to its length times its task-number improvement with respect to the interval i_1. Only intervals with fewer tasks than the i_1 interval's (without the task t_1) are considered for this decision, and the maximal length of intervals considered for the multiplication is that of the task t_1.

5.1.2. *Global Task Constraints.* A task constraint manages a set of temporal relations between a job's tasks (see Fig. 6). Each relation involves links to two decision variables V_1 and V_2, a constant c and a comparator $\bowtie \in \{<, =, >\}$, such that $V_1 \bowtie V_2 + c$. The inconsistency of a relation is given by the minimal shift distance for one of the variables required to satisfy the relation. The TC's total inconsistency is the sum of the relations' inconsistencies.

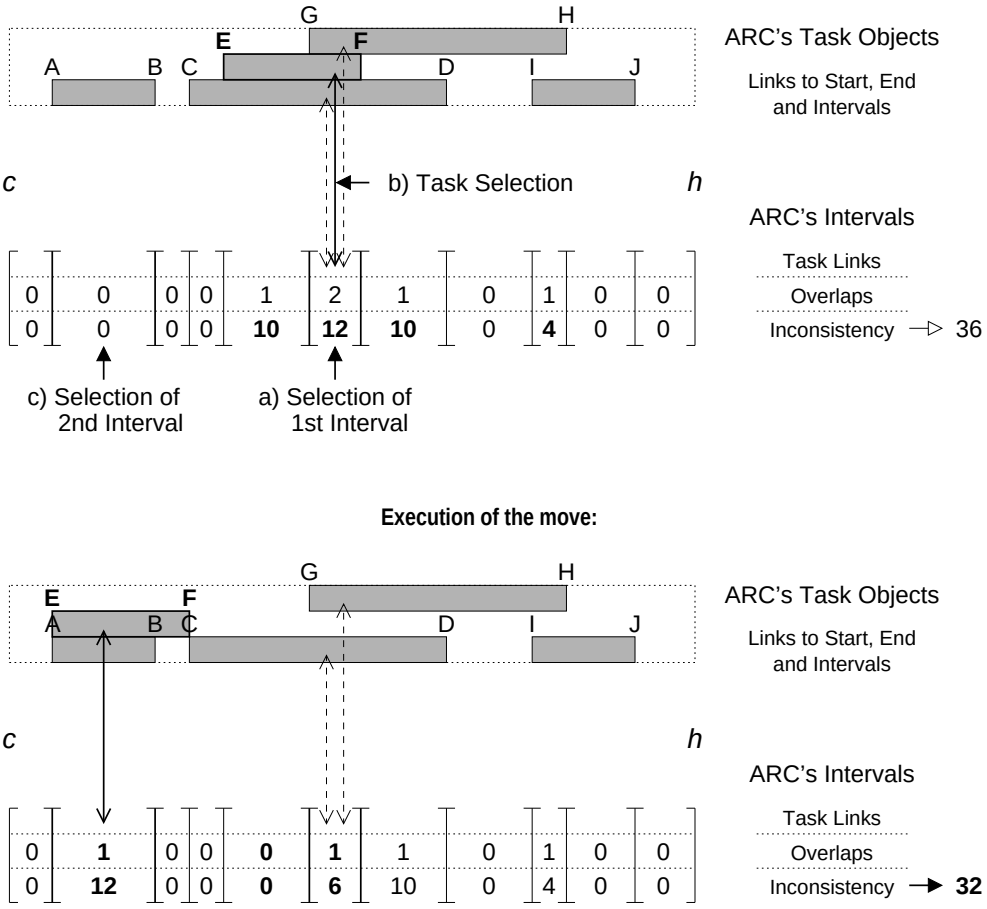

FIGURE 5. The ARC-H1 Heuristic

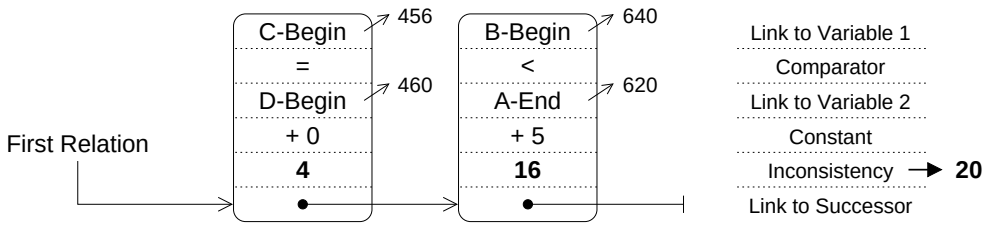

FIGURE 6. A TC's Internal Structures

The basic improvement heuristic of the task constraint (TC-H1) selects an inconsistent relation with a choice probability for a relation that is proportional to its inconsistency (see Fig. 7). One of the involved variables is selected randomly, and a minimal shift of this variable is performed such that the relation is fulfilled.

5.2. Results. Figure 8 shows experiments with different horizons. The global search control selects a constraint with a probability proportional to the constraint's costs. The schedule always contains 50 jobs ($\rightarrow$ 50 TCs), each of them with five

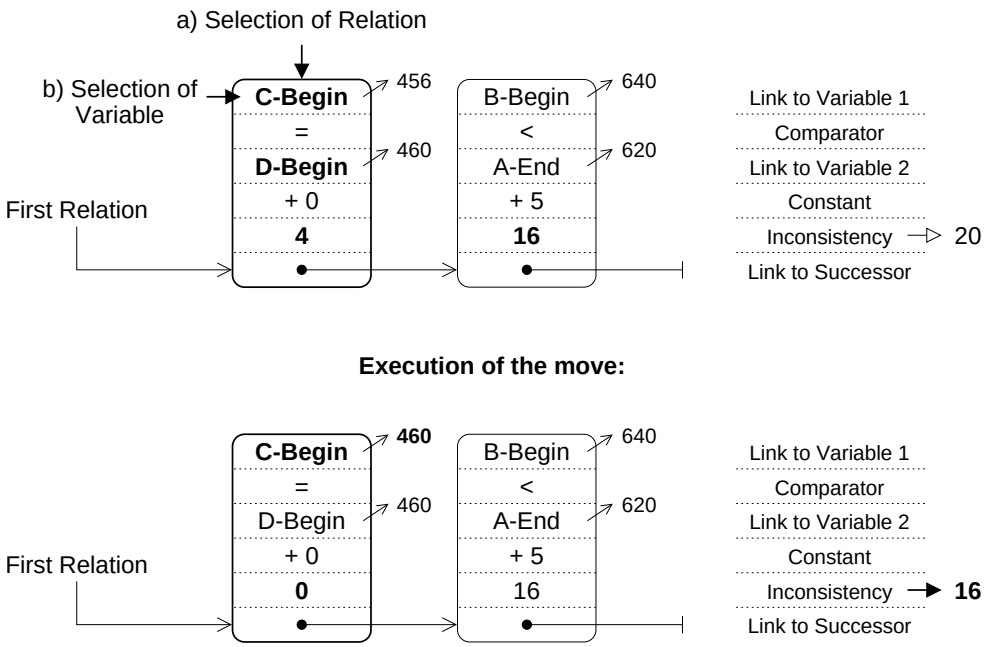

FIGURE 7. The TC-H1 Heuristic

tasks, and there are five machines ($\rightarrow$ 5 ARCs). Every tenth of a second there is a job removal/addition[4].

According to a computation using refinement/global search by the ConPlan system [**Gol97**], the minimal horizon for a maximal consistent solution varies around 6,000, depending on the currently active jobs[5]. With a horizon of 2,000, the topology of the search space is so flat that any improvement effect is close to pure noise. A more complete picture is given in Fig. 9. One point represents the inconsistency averaged over 10 seconds of runtime.

In order to study the search behavior until complete satisfaction was achieved, no job removal/addition was done for the rest of the experiments. There are 10 jobs ($\rightarrow$ 10 TCs) with 10 tasks, 10 machines ($\rightarrow$ 10 ARCs) and a horizon of 2,300. The start inconsistency is about 50,000.

5.3. Constraint Weights. The weights of the constraints' subjective costs for the overall cost function have so far been set to one. Looking at the individual constraints' cost development for a test run (Figure 10) may give the impression that this is not the best choice. Better weightings could restructure the search space and ensure that the inconsistency of possibly more critical constraints plays a more important role and that these constraints are chosen more often to execute improvement changes.

[4] Although there is an enormous variety of possible problem configurations, the results are presented for just one problem instance (given as the average of 1,000 test runs if not otherwise stated). Comparable results were obtained with other instances.

[5] A performance comparison with the ConPlan system involving the dynamics was not possible as a tenth of a second was not even enough to set up the constraints.

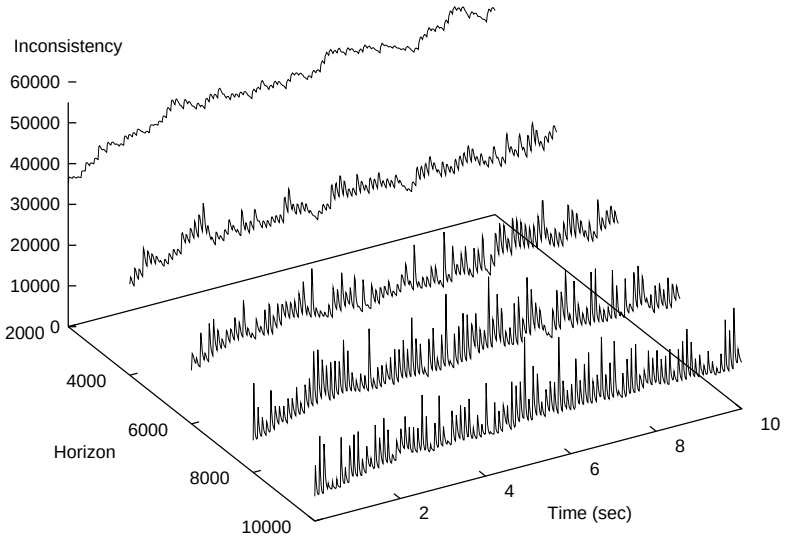

FIGURE 8. Horizon Variation — Single Runs

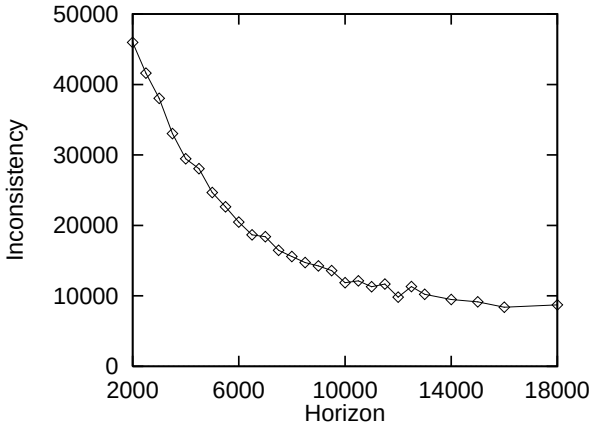

FIGURE 9. Horizon Variation — Averages

Figure 11 shows that this is not the case. The weights for the ARC and TC constraints' costs are given after the colons[6]. A stronger weighting of the ARCs has a negative effect, whereas higher weights for the TCs neither worsen nor improve the search behavior.

[6]To be able to better compare the differently weighted inconsistencies, the results are presented by displaying the inconsistencies with constraint weights of one. As the line patterns are sometimes difficult to recognize, the legends list the curves in the lines' order from top to bottom.

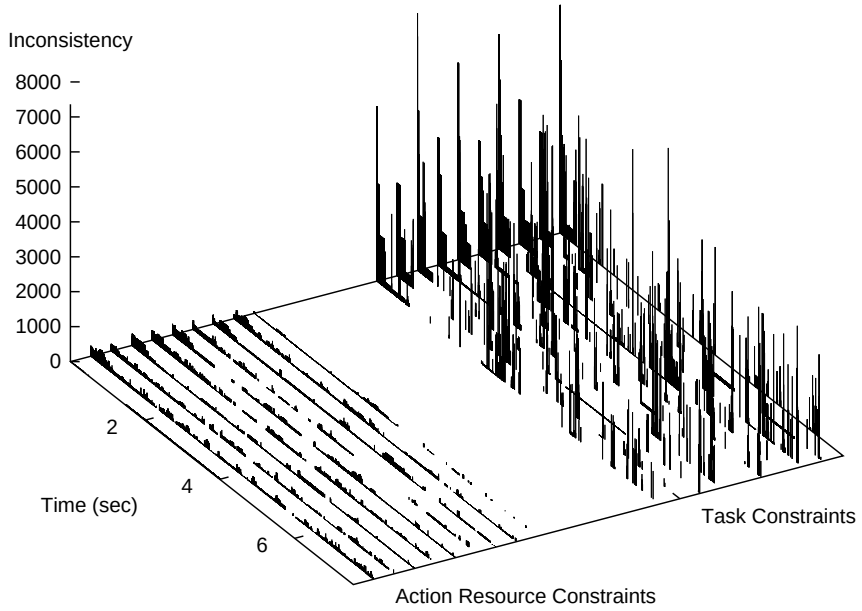

FIGURE 10. Cost Distribution for a Single Test Run

6. Susceptibility to Local Minima and Plateaus

Unlike low-level CSP-based representations, global constraints enable the search to be conducted in a more informed way. A measure for this is the susceptibility to getting caught in local minima and on plateaus. This is investigated in the following subsections.

6.1. Randomization. The previous test runs had randomization at all choice options. This is a common technique to leave local minima and plateaus. However, randomization need not always have a positive effect. Figure 12 shows choice variants, where N means choosing the subject with the highest inconsistency, and R means a choice with a probability of a subject's being chosen that is proportional to the subject's inconsistency. The letters g, m and t indicate the choice points: g the global search control's constraint selection, m the first interval choice of the ARCs' improvement heuristic (see Fig. 5), and t the relation choice of the TCs' improvement heuristic (see Fig. 7).

For the first phase (Fig. 12, top graph) of the search, the quality of the strategies can be more or less ordered according to their amount of randomization, the nonrandomized *NgNmNt* version clearly being the best. The superiority of nonrandomized strategies is not surprising, as local minima and plateaus are less probable in the early phase. The *Ng* component has the most important impact. Strategies with an *Rg* component are nearly always worse, even in the later phases of the search (middle and bottom graphs).

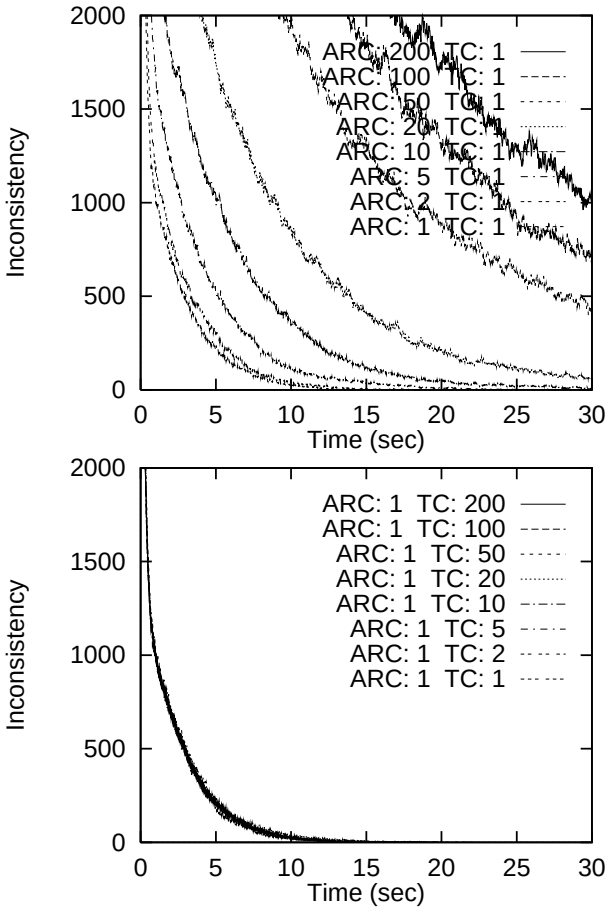

FIGURE 11. Variation of Weights

As the search proceeds, the Rm component becomes more important, indicating that the ARC's heuristic no longer always makes the right decisions. Shortly after, the Rt component acquires some influence as well, making $NgRmRt$ the first to converge to a complete satisfaction, followed by the nonrandomized $NgNmNt$. $RgNmNt$ is the third to achieve complete satisfaction, only a little before $NgRmNt$.

In general, nonrandomization seems to be best for the g decision, whereas randomization of m and t depends on the available computation time. The randomization of m is much more important than that of t, which indicates that the ARC's heuristic is not very powerful.

One should be careful with anytime switching between variants for different search phases based on the graphs of the individual variants. The switch to a variant with the steepest descent for an actual inconsistency does not necessarily represent the optimal behavior, because each variant has a different search history and may require structurally very different areas of the search space in order to advance. A prognosis of the behavior of switching strategies is further complicated by the dynamics of the dynamic job-shop scheduling problem.

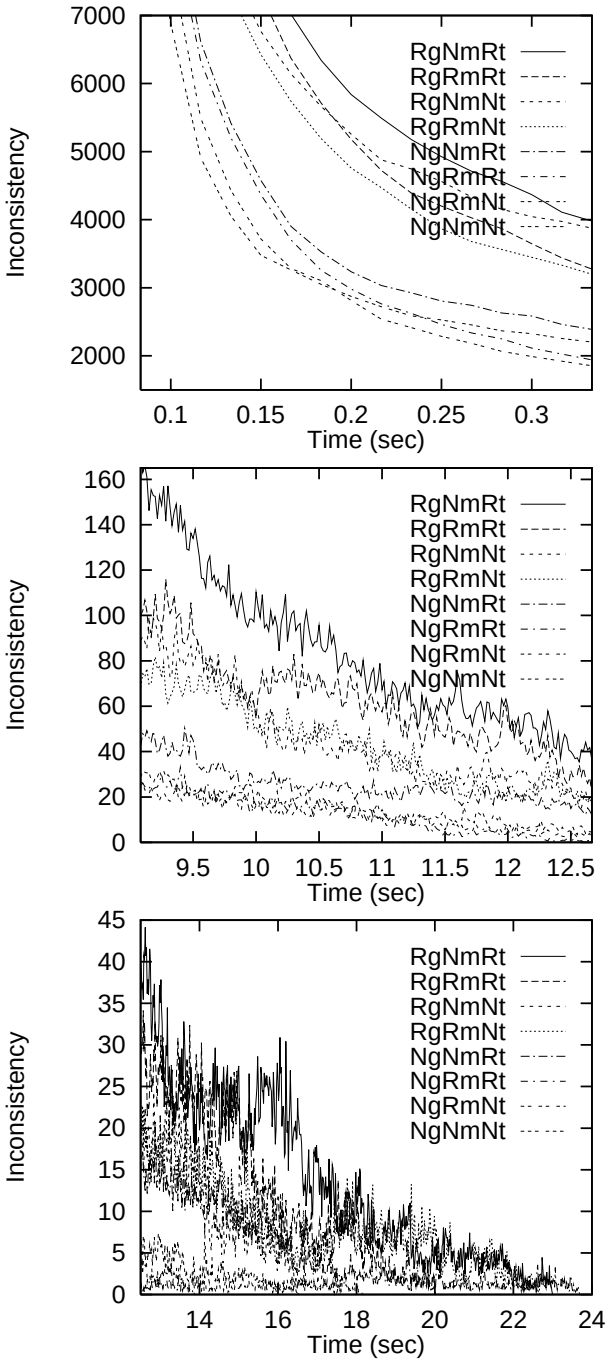

FIGURE 12. Randomization Variants

6.2. Random Walks. Random walks are random moves in the search space that disregard the change of the cost function value. The idea is that the search is retracted from hopeless situations (like local minima) from time to time. Unlike restarts, random walks remain within the area of the current state.

Random walks can be included by introducing a second improvement heuristic for each constraint that makes a random variation of a random variable. Figure 13 shows the results for different probabilities for the random variation heuristics to be chosen. It is obvious that the random walks generally cause a deterioration in the results. At no point is there a cross-over: the more random walks, the more inconsistency.

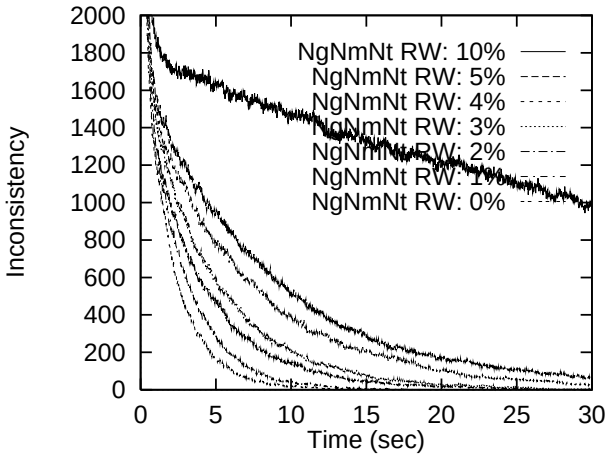

FIGURE 13. Random Walks

6.3. Tabu Lists. Figure 14 shows experiments using a tabu list for the global search control's constraint selection (based on tabu search [**Glo89**]). Each selected constraint is stored in a first-in-first-out list and blocked for another selection as long as it is a member of the list.

Applying tabu lists proves absolutely pointless. Even for the nonrandomized NgNmNt version, it makes no difference (results for other randomization variants are similar).

7. Extending the Constraints

The constraints' improvement heuristics can be created in various ways. The effect of heuristics with stronger domain knowledge and the inclusion of more aggressive heuristics are studied in the following subsections.

7.1. More Knowledge. The experiments in Section 6.1 showed that the ARC's heuristic is not very powerful. It would therefore seem advisable to consider other heuristics. For example, tasks should obviously be packed quite tightly on a machine. The following ARC-H2 heuristic supports this feature. ARC-H2 is very similar to ARC-H1, but it makes the selection of the second interval in a different way: for the new position of the task, only two shifts are possible, the task either beginning at the beginning of the task's predecessor interval or ending at the end of the task's successor interval. The interval with less tasks is chosen.

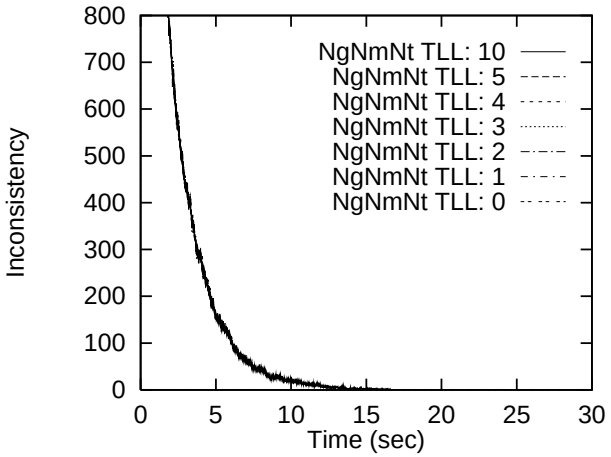

FIGURE 14. Results for Tabu-List Lengths of 0, 1, 2, 3, 4, 5, and 10

The impact of the new heuristic is impressive. The best results are obtained using a probability of about 90 % for the ARC-H2 heuristic to be chosen, and 10 % for the ARC-H1 heuristic (see Fig. 15). Choosing the ARC-H2 heuristic more often causes a deterioration in the results.

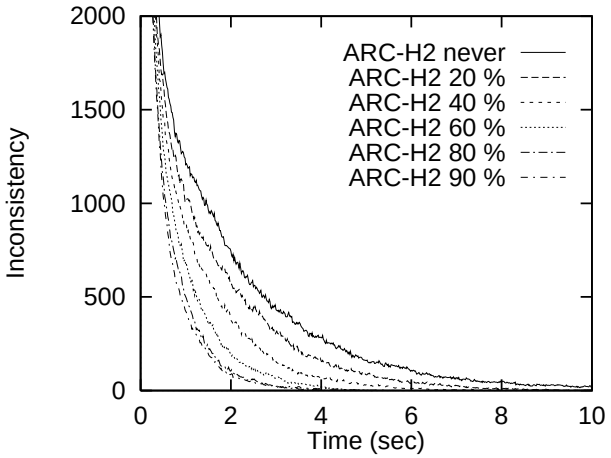

FIGURE 15. Introduction of the New ARC's Heuristic

Figure 16 shows the impact of randomization, indicating the first interval's selection of the ARC-H2 heuristic by the letter *s* (always with a probability of 90 % for the ARC-H2 heuristic to be chosen). The addition of domain knowledge by the new heuristic strongly reduced the effect of randomization compared to that in Section 6.1.

7.2. Aggressive Heuristics. The task constraint can also be extended. The current heuristic is very cautious, changing just one variable. After a couple of changes of task positions within a job, it may be useful to make a complete revision

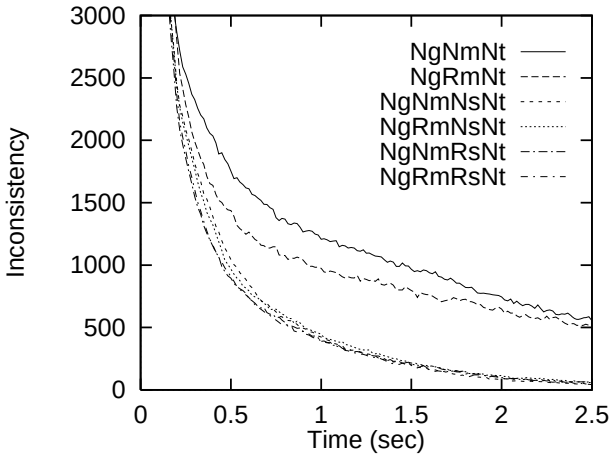

FIGURE 16. Randomization with the New ARC's Heuristic

of the internal distance relations instead of repairing only one relation. A more aggressive heuristic can add to the former heuristic a recursive repair of all relations of the constraint whose inconsistency has been changed by the improvement (considering only variables that have not already been changed within the improvement step).

The effect is disappointing (Figure 17; using only the old ARC-H1 heuristic for the ARCs). Even a slight deterioration in the results is caused by activating the task constraint's new heuristic.

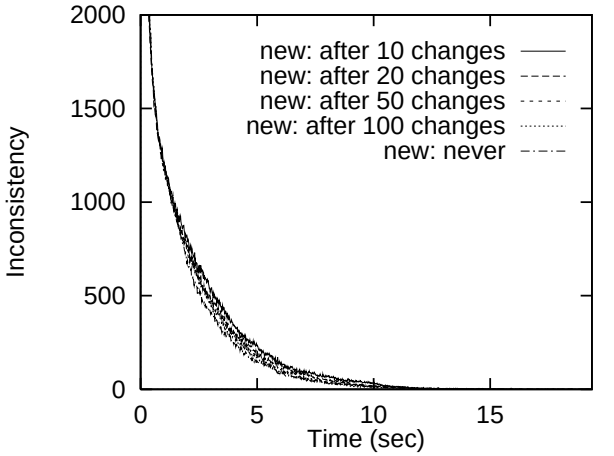

FIGURE 17. Introduction of the New TC's Heuristic

Cautious heuristics are often more appropriate than highly aggressive ones, the synthesis with other problem aspects being promoted by only slight changes of multi-constraint variables.

8. Related Work and Conclusion

Many publications focus on problem-specific local search solutions. Improved efficiency is the main goal, generality often being disregarded. This paper has demonstrated a way of combining local search with the constraint programming framework. The *modular structure* of the constraints makes it *easy to vary, reuse and extend problem descriptions.*

Other authors have tackled the problem of combining local search with search frameworks on a more general level, too. This includes work on Boolean satisfiability problems like GSAT [**Gu92, SLM92**] and Walksat [**SKC96**], the processing of linear pseudo-Boolean constraint problems [**Wal97**], and approaches for CSPs like coalition forming [**HT95**] and the well-known min-conflicts heuristic [**MJPL92**] with its extension and generalization by GENET [**DTWZ94**]. The most important difference between our work and these approaches is the ability of the global constraints to exploit domain-specific information by including constraint-specific search control and representation knowledge. In contrast to low-level constraint programming approaches, which correspond rather to SAT- or OR-based approaches, the use of higher-level constraints is more in keeping with the basic intentions of constraint programming. Fine-grained constraints allow a wide application range, but the low-level problem decomposition also deprives the search process of most of the domain-specific knowledge.

The results of the presented case study indicate that the global constraint approach's *revision of a current state on a more global level* with an *inclusion of domain-specific knowledge* makes the search quite resistant to getting caught in local optima or on plateaus. Techniques to escape from local optima and plateaus, like random walks or tabu lists, either worsened or failed to improve the search behavior. The only useful technique was randomization, though this was true for some decision points only, the advantage decreasing with the inclusion of stronger knowledge.

The concept of global constraints was originally used for refinement search (e.g., [**LeP94, PL95**]). Transferring it to a local search context makes it possible to get an efficient and declarative handle on local search, while preserving features like reusability and maintenance.

Further work will include concepts to temporally focus the search on specific problem aspects and methods to apply case-based reasoning within global constraints. More detailed information on the EXCALIBUR project is available at:

`http://www.ai-center.com/projects/excalibur/`

References

[Aarts97] Aarts, E. H. L., and Lenstra, J. K. eds. 1997. *Local search in Combinatorial Optimization.* Reading, Wiley-Interscience.

[DTWZ94] Davenport, A.; Tsang, E.; Wang, C. W.; and Zhu, K. 1994. GENET: A Connectionist Architecture for Solving Constraint Satisfaction Problems by Iterative Improvement. In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94), 325–330.

[GHJV95] Gamma, E.; Helm, R.; Johnson, R.; and Vlissides, J. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software.* Reading, Addison-Wesley Professional Computing Series.

[Glo89] Glover, F. 1989. Tabu Search — Part I. *ORSA Journal on Computing* 1(3): 190–206.

[Gol97] Goltz, H.-J. 1997. Redundante Constraints und Heuristiken zum effizienten Lösen von Problemen der Ablaufplanung mit CHIP. In Proceedings of the 12. Workshop on Logic Programming (WLP'97), Forschungsbericht PMS-FB-1997-10, LMU München, Germany.

[Gu92] Gu, J. 1992. Efficient Local Search for Very Large-Scale Satisfiability Problems. *SIGART Bulletin* 3(1): 8–12.

[HT95] Hirayama, K., and Toyoda, J. 1995. Forming Coalitions for Breaking Deadlocks. In Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), 155–162.

[KS98] Kautz, H., and Selman, B. 1998. BLACKBOX: A New Approach to the Application of Theorem Proving to Problem Solving. In Working Notes of the AIPS-98 Workshop on Planning as Combinatorial Search, 58–60.

[LeP94] Le Pape, C. 1994. Implementation of Resource Constraints in ILOG Schedule: A Library for the Development of Constraint-Based Scheduling Systems. *Intelligent Systems Engineering* 3(2): 55–66.

[MJPL92] Minton, S.; Johnston, M. D.; Philips, A. B.; and Laird, P. 1992 . Minimizing Conflicts: a Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. *Artificial Intelligence* 58: 161–205.

[Nar00] Nareyek, A. 2000. Open World Planning as SCSP. In Papers from the AAAI-2000 Workshop on Constraints and AI Planning, Technical Report, WS-00-02, 35–46. AAAI Press, Menlo Park, California.

[PL95] Puget, J.-F., and Leconte, M. 1995. Beyond the Glass Box: Constraints as Objects. In Proceedings of the 1995 International Logic Programming Symposium (ILPS'95), 513–527.

[Rég98] Régin, J.-C. 1998. Minimization of the Number of Breaks in Sports Scheduling Problems using Constraint Programming. In Proceedings of the DIMACS Workshop on Constraint Programming and Large Scale Discrete Optimization, P7: 1–23.

[SKC96] Selman, B.; Kautz, H.; and Cohen, B. 1996. Local Search Strategies for Satisfiability Testing. In Johnson, D. S., and Trick, M. A. (eds.), *Cliques, Coloring, and Satisfiability*, DIMACS Volume 26: 521–532.

[SLM92] Selman, B.; Levesque, H.; and Mitchell, D. 1992. A New Method for Solving Hard Satisfiability Problems. In Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92), 440–446.

[VAL94] Vaessens, R. J. M.; Aarts, E. H. L.; and Lenstra, J. K. 1994. Job Shop Scheduling by Local Search. Technical Report, COSOR Memorandum 94-05, Eindhoven University of Technology, Department of Mathematics and Computing Science.

[Wal97] Walser, J. P. 1997. Solving Linear Pseudo-Boolean Constraint Problems with Local Search. In Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97), 269–274.

GMD FIRST, Kekuléstr. 7, 12489 Berlin, Germany
E-mail address: alex@ai-center.com