Feedback Control for Real-Time Solving

Ying Lu [1], Lara S. Crawford [2*], Wheeler Ruml [2] and Markus P.J. Fromherz [2]

[1] Department of Computer Science, University of Virginia
Charlottesville, VA 22903
ying@cs.virginia.edu

[2] Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304
{lcrawford, ruml, fromherz}@parc.com

Abstract. Numerous solvers have been proposed to solve constraint satisfaction problems (CSPs) or constrained optimization problems (COPs). Research has demonstrated that solvers' performance is instance-dependent and that no single solver is the best for all problem instances. In this paper, we further demonstrate that solvers' relative performance is time-dependent and that, given a problem instance, the best solver varies for different solving time bounds. We investigate an on-line feedback control paradigm for solver or problem reconfiguration so that the solver can reach the best possible solution within a specified time bound. Our framework is unique in specifically considering the time constraint in the feedback control of solving. With this augmented time-adaptivity, our paradigm improves solver performance for real-time applications. As a case study, we apply the feedback control paradigm to real-time performance control of a multidimensional knapsack problem solver.

1 Introduction

Given a problem instance, some solvers or solver configurations perform vastly better than others. In the literature, much research has tried to provide guidance for matching the right solver to a problem instance. Minton [24] pointed out that the performance of solvers is instance-dependent, i.e., for a given problem class a solver can perform well for some instances, but poorly for others, which makes the matching very difficult. Many authors have used off-line analysis (based on statistics or problem characteristics available before actually beginning to solve an instance) or probing to optimize algorithms or heuristics for a particular class of problems or even a particular instance. Others have used information acquired during a solving run to iteratively tune the solving process in a type of feedback loop.

For many real-world problems, a hard or soft time constraint is imposed on the solving process. The solvers have to be terminated within certain time

* Corresponding author.

bounds in order to provide acceptable service. Although previous work has always attempted to improve solving efficiency, it has almost never explicitly taken the time bound into account when selecting solvers, heuristics, or parameter values. We demonstrate in this paper that the best solver choice is dependent on the time constraint and propose an on-line feedback control framework that will adapt the solving process to the problem instance as well as to the real-time application requirement.

2 Related Work

There is a large body of literature on off-line adaptive problem solving. A number of systems (see Minton [24], Gratch and DeJong [15, 14], and Caseau, Laburthe, and Silverstein [8]) have used off-line analysis to optimize algorithms or heuristics for a particular class of problems. This approach can be seen as analogous to designing an open-loop controller, in the sense that the selection and tuning of algorithms, heuristics, and problem transformations are done in advance of the solving and are not responsive to the on-line performance of the system. The same is true for approaches such as that in Flener, Hnich, and Kızıltan [12], in which a model is built off-line defining the relationship between the problem instance and the best set of heuristics to use. There are several similar approaches to on-line algorithm or heuristic selection (see Allen and Minton [3] and Lobjois and Lemaître[23]). Although these approaches probe the problem instance on-line to determine the best algorithm or heuristics to use, and thus take performance feedback into account during this stage of solving, once the selection is made, no further feedback is used. A similar approach can be applied to algorithm parameter selection: in their *Auto-WalkSAT* algorithm, Patterson and Kautz [27] use probing to identify the best noise parameter for a particular algorithm/solver pair. Finally, instance-based solver or parameter selection need not depend on probing. Nudelman and his co-authors [22, 26] use a statistical regression approach to learn which problem features can be used to predict the run time of different solvers. They then use this prediction to select the fastest solver in a portfolio for each instance.

There are a number of approaches that make more use of feedback-type information for algorithm or parameter selection or for search control. Borrett, Tsang, and Walsh [5] use on-line performance feedback to switch between algorithms. Ruan, Horvitz, Kautz, and their coauthors [20, 29, 30] use it as part of a dynamic restart policy. Hoos [19] uses stagnation monitoring to dynamically adjust the noise parameter in WalkSAT algorithms. In the evolutionary algorithms community, a variety of techniques have been used to adapt genetic operators and parameters based on various performance measures (Eiben, Hinterding, and Michalewicz [11]). Similar methods have been used with simulated annealing (Wah and Wang [35]). There are also a variety of approaches that dynamically build up estimates of value or cost functions to guide the search (see, for example, Baluja, *et al.* [4], Boyan and Moore [6], Narayek [25], Ruml [31], and Lagoudakis and Littman [21]. These functions are measurements of the

"goodness" of particular states or action choices, and are developed on-line using accumulated performance data.

Adaptive techniques have also been used to modify problem representations. An "open-loop" off-line design approach for problem reformulation has been proposed by Hnich and Flener [17]. Feedback approaches have been used as well. For example, Pemberton and Zhang [28] have used (open-loop) phase transition information and on-line branching estimation to identify complex search problems and transform them into easier searches producing suboptimal solutions. Modification of penalty weights or chromosome representations in response to performance has also been explored in the evolutionary algorithms community (Eiben, Hinterding, and Michalewicz [11]).

In real time systems, though, time deadlines are a fact of life. None of the approaches described above explicitly takes this time bound into account when selecting solvers, heuristics, or parameter values. Some of these techniques represent anytime algorithms that can be stopped when a time bound is reached, but the time bound is not considered earlier. The solver thus does not take advantage of the known stopping time in order to make appropriate performance/speed trade-offs. Techniques also exist to monitor anytime algorithms and stop them when the solution improvement no longer justifies the additional time expenditure (Hansen and Zilberstein [16]), but again, this approach does not take a time bound explicitly into account in selecting or tuning the solver. Very recently, Carchrae and Beck [7] demonstrated a feedback-based system that switches among algorithms to get the best solution at a deadline. Their approach has a number of similarities to ours. Our goal in the example described here, however, is to use feedback to fine-tune algorithm or problem parameters rather than to select the algorithms themselves, though both applications fit within our general framework.

3 System Design

3.1 Solver Control

The generic framework for the feedback control of solving is shown in Figure 1 (see Crawford, *et al.* [10]). The control module is built upon a model or set of rules reflecting the relationship between problem solving dynamics (y), the real-time application requirement (i.e. the deadline T) and the choices for solvers, solver configurations, and problem transformations. These choices define the control parameters (u) of the solver. The model or rule base enables the prediction of the solver behavior defined by these control parameters. Based on the predicted behavior, the control module updates the control parameters (u), in order to achieve the best suboptimal solution at the specified time bound.

3.2 Control Parameters

Control parameters u that could be used to change the solving performance include the choice of solvers, solver configurations or problem representations. For

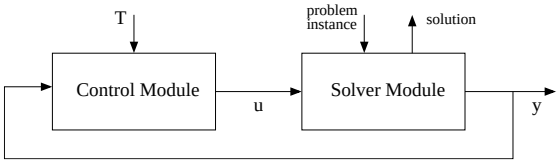

Fig. 1. Feedback control of solving framework.

example, some solvers may work better on under-constrained problems, while others are better choices for over-constrained problems (Shang and Fromherz [32]). In this case, the control parameter could be the choice of solvers. Another example is a solver with a restart policy, for which it is useful to adopt different restart cutoffs for problem instances with varied hardness levels. Previous research by Horvitz, Ruan, Kautz, and their co-authors [20, 29, 30] has proposed dynamic restart policies where the choice of cutoff is instance-based. In those mechanisms, the control parameter would be the restart cutoff configuration. Another possible control parameter is one defining a problem representation. For example, Pemberton and Zhang's ϵ-transformation [28] makes use of a parameter ϵ to define an off-line transformation of a tree search problem to one of lower complexity (with a loss of optimality, of course). The value of ϵ defines the severity of the reformulation.

In this paper, we investigate problem representation as a key control parameter for solving constrained optimization problems under a time bound. A linear constrained optimization problem is typically of the form:

$$\text{maximize} \quad c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

respecting the constraints:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n + b_1 &\leq 0 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n + b_2 &\leq 0 \\ &\cdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n + b_m &\leq 0 \end{aligned} \tag{1}$$

where x_i, $i \in 1, 2, \cdots, n$ are variables whose value lies in some permissible set $\{X\}$.

To simplify problem solving, sometimes it is desirable to reduce the problem scale by problem transformation. The new problem representation should satisfy the following two requirements. First, it is deduced from the original problem but has smaller number of variables or constraints. Second, a solution to the new problem can be transformed into a solution to the original problem.

Many approaches exist for such a problem transformation. Three simple possibilities are variable grouping, constraint grouping and variable removal. In variable grouping, two or more variables are grouped together and considered as one new variable. This type of problem transformation leads to the same assignments for the variables in one group. For example, assume that we group

variables x_1 and x_2 in Equation 1 together and consider the group as a new variable g_1. Then the original problem becomes

$$\text{maximize} \quad (c_1 + c_2)g_1 + c_3x_3 + \cdots + c_nx_n$$

such that

$$\begin{aligned}(a_{11} + a_{12})g_1 + a_{13}x_3 + \cdots + a_{1n}x_n + b_1 &\leq 0\\ (a_{21} + a_{22})g_1 + a_{23}x_3 + \cdots + a_{2n}x_n + b_2 &\leq 0\\ &\cdots\\ (a_{m1} + a_{m2})g_1 + a_{m3}x_3 + \cdots + a_{mn}x_n + b_m &\leq 0\end{aligned}$$

Constraint grouping combines two or more constraints together. That is, a new constraint will be used to replace the old constraints in such a way that the satisfaction of the new constraint leads to the satisfaction of all the old constraints. For instance, the grouping of the first two constraints in Equation 1 may result in the following new constraint (assume the permissible values of the variables are positive),

$$max(a_{11}, a_{21})x_1 + \cdots + max(a_{1n}, a_{2n})x_n + max(b_1, b_2) \leq 0$$

Variable removal fixes the assignments of some variables so that the number of unknown variables is smaller after the problem transformation.

Other, more complex approaches to problem transformation along these lines can also be envisioned; for example, hierarchical approaches to variable grouping in which the problem is decomposed into a number of smaller subproblems. In all the grouping approaches, whether of variables or constraints, care must be taken when choosing how to define the groups, as this choice can affect solver performance significantly.

3.3 Control Module

In the feedback control framework, the control module leverages a model or set of rules to predict the solving behavior with different control parameter choices. Therefore, a good model that accurately reflects the solving dynamics is the key to good control parameter selection. In this paper, we propose a general modeling framework. Given a solver with a defined set of parameter choices $U = u_1, u_2, \ldots, u_N$, we will first apply it to a group of representative benchmark problem instances $P_1, P_2, \ldots, P_M$. The solving performance on those instances will then be used to generate the model (see Figure 2). This approach is similar to that taken by Nudelman *et al.* [26], but with the differences that we are aiming to predict solution quality at a time bound rather than solver run time (to the optimal solution), and that we use data acquired during the solving process in the predictions.

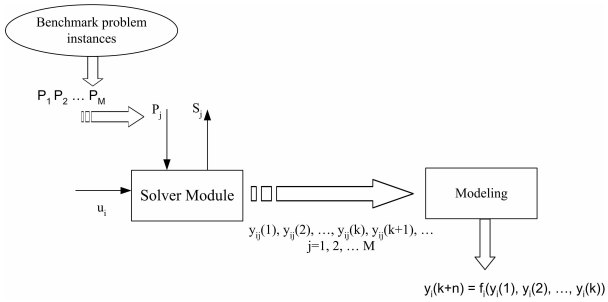

Fig. 2. Modeling framework.

As shown in Figure 2, a model (represented by f_i) is generated for each solver configuration (defined by control parameter u_i)[1]. The control module will then use the models ($f_1, f_2, \cdots, f_N$, assuming there are N different solver configuration choices) to predict the solving behavior on any given problem instance and try to choose the best parameters for solving the problem instance. As the control module will use the dynamic solving information $y(k)$, such as suboptimal solutions obtained, to make the best parameter choice, parameter changes may be required on-line. For example, the control module could choose some $u(1) \in U$ at the first sampling interval and later switch to other configurations $u(2), u(3), \cdots, u(k) \in U$ if switching is predicted to produce better solving performance.

When changing parameters from one configuration to another, one of two types of switch strategies can be applied. These are constructive and restart strategies. A constructive strategy uses the dynamic information obtained with one configuration to adapt or modify the starting point of the solving with the new configuration, while with a restart strategy, when the configuration switches from u_1 to u_2, say, the solving process for u_1 will pause and the process for u_2 will start from scratch or restart from where it was paused. An intelligent constructive strategy has the potential to improve the performance of the new configuration, but may be difficult to build, depending on the type of parameters u being used, and may require a solver-specific approach. With the simple restart strategy, on the other hand, the solving process of each configuration is independent, which leads to a simpler control system whose dynamics will be much easier to model. Hence, in our general feedback control framework, we have chosen the restart switching strategy.

For prediction and control in this framework to be feasible, two assumptions must be made. First, we assume that the solver performance on the modeling problem instances can be used to predict the performance on new instances.

[1] This paper only considers the case where there is a limited discrete number of control parameter choices. The modeling framework for control with a continuous choice of parameters requires a somewhat different modeling approach or a method for interpolating among models and is a topic for future work.

Second, if the solver behavior on two problem instances are the same from the beginning of the solving, then the solver is assumed to be very likely to continue performing similarly on the two for the rest of the solving process.

Based on these assumptions, a brute force prediction and control algorithm can be designed from a simple model. Let S be the set of M modeling problem instances. The modeling process solves them until the specified time bound (T) with each solver (defined by the parameter configuration $u_i \in U$). The suboptimal solutions reached by each solver at every sample interval are recorded. We use $y_{ij}(k)$ to represent the suboptimal solution reached by the i^{th} solver at the k^{th} sampling interval for modeling problem P_j. Note that the same parameter configuration is used throughout a solving run (there is no switching).

When solving a new problem instance p, we will use a distance metric to evaluate how similar the performance on p is to that on each modeling instance P_j. The distance to P_j at time interval k, $D_j(p,k)$, is calculated based on the performance on p by the solvers (defined by $u_1, u_2 \cdots, u_N$) and their known performance on P_j. Assume that at time interval k, the algorithm has spent k_i sampling intervals on the solving process with configuration u_i, where $\sum_{i=1}^{N} k_i = k$, and generated outputs $y_i(1), y_i(2), \ldots, y_i(k_i)$. Then we calculate $D_{ij}(p,k_i)$, the distance between p and P_j with solver u_i at time interval k as follows:

$$D_{ij}(p,k_i) = \sum_{n=1}^{k_i} |y_{ij}(n) - y_i(n)| \tag{2}$$

All experience is weighted equally. From 2 the distance metric $D_j(p,k)$ is derived:

$$D_j(p,k) = \sum_{i=1}^{N} D_{ij}(p,k_i) \tag{3}$$

At each sampling interval, the algorithm calculates $D_j(p,k)$ for every $P_j \in S$ and finds the modeling problem that is the smallest distance from p. That is, it finds the problem P_{opt} that yields the minimum distance $D_{opt}(p,k) = min(\{D_j(p,k)|P_j \in S\})$. Then, the algorithm uses the solvers' performance on instance P_{opt} to predict how they would perform on p if allowed to run for the rest of the sampling intervals. Then, for the next sampling interval, the algorithm chooses the solver configuration $u(k+1)$ whose predicted final result at deadline T is best. The above process is repeated at each interval. At the beginning of a solving run, a minimum number of sampling intervals of each configuration will be run to ensure accurate prediction. For the sake of reducing switching in difficult cases, the feedback process continues until a specified time, at which point one solver configuration will be chosen for the rest of the solving.

4 Case Study

As a case study, we apply the proposed feedback control paradigm (Section 3) to the real-time performance control of a multidimensional knapsack problem

solver. The real-time solving performance of this type of problem is important because many practical real-time problems such as resource allocation in distributed systems, capital budgeting, and cargo loading can be formulated as multidimensional knapsack problems.

4.1 Multidimensional Knapsack Problem Solver

The 0-1 multidimensional knapsack problem (MKP01) can be stated as:

$$MKP01 = \begin{cases} \text{maximize } c \cdot x \text{ subject to} \\ Ax \leq b \text{ and } x \in \{0,1\}^n \end{cases}$$

where $c \in \mathbb{N}^n$, $A \in \mathbb{N}^{m \times n}$ and $b \in \mathbb{N}^m$.

There are a number of approaches to solving the MKP01 problem in the literature. A standard was set by Chu and Beasley [9], who obtained good results using a genetic-algorithm-based heuristic method. Their problem set was made publicly available in the OR-Library [2], and has been a benchmark problem set for testing other algorithms. Their paper also provides a useful survey of MKP01 solvers at that time. Many other heuristic solvers have been proposed since then (for example, Holte [18], Fortin and Tsevendorj [13], and Vasquez and Hao [33]), some placing emphasis on solution quality and some focusing on solving speed.

We will here apply the feedback control paradigm to a MKP01 solver developed by Vasquez, Hao, and Vimont [33, 34]. This solver takes a hybrid approach that combines linear programming with an efficient tabu search algorithm. It gave results on the OR-Library benchmarks [2] that the authors claim were the best known at the time. Other solvers could of course be used as targets for real-time performance control; we chose this one based on its final solution quality.

The main idea of the hybrid solver is to perform a search around a solution of the fractional relaxed MKP01 problem with additional constraints. Starting from the obvious statement that each solution of MKP01 satisfies the property: $1 \cdot x = \sum_1^n x_j = k$, where k is a positive integer, they add this constraint to the fractional relaxed MKP01 to obtain a series of problems like:

$$MKP[k] = \begin{cases} \text{maximize } c \cdot x \text{ s.t.} \\ Ax \leq b \text{ and } x \in [0,1]^n \text{ and} \\ 1 \cdot x = k \in \mathbb{N} \end{cases}$$

These $MKP[k]$ are solved and their fractional solutions, $\bar{x}_{[k]}$, are used as starting points for tabu searches. They are solved in order of most promising to least promising k. In Vasquez and Vimont [34], k_0, the first value used, is the rounded sum of the elements of the optimal solution $\bar{x}$ of the relaxed MKP01. That is, $k_0 = [1 \cdot \bar{x}]$, where

$\bar{x} = \arg\max c \cdot x$ s.t.

$Ax \leq b, x \in [0,1]^n$

The region around $\bar{x}_{[k_0]}$ is the first to be tabu searched. Next, regions around $\bar{x}_{[k]}, k = k_0 - 1, k_0 + 1, k_0 - 2, k_0 + 2, \cdots$ are sequentially tabu searched. According to Vasquez and Vimont [34], this search order ensures the exploration of the hyperplanes $1.x = k$ in the decreasing order of $\bar{z}_{[k]} = c.\bar{x}_{[k]}$, the optimal values of $MKP[k]$.

To reduce the search space, the algorithms impose a geometric constraint on the tabu search neighborhood, so that the local search is limited to a sphere of fixed radius around the fractional optimum $\bar{x}_{[k]}$. Therefore, each binary configuration x reached by the local tabu search satisfies the following two constraints:

1) $1 \cdot x = k$

2) $|x, \bar{x}_{[k]}| = \sum_{j=1}^{n} |x_j - \bar{x}_{[k]j}| \leq \delta_{max}$

4.2 Control Approach

To control the performance of the aforementioned hybrid solver in the context of a fixed time bound, several control parameters could be effective. First, the choice of the geometric constraint δ_{max} on the tabu search neighborhood will have an important impact on its performance. We believe that the best choice of δ_{max} is problem instance dependent and application requirement dependent. Therefore, on-line feedback tuning may be required to reach the optimal configuration. Allocating the time that the algorithm will run on each hyperplane is another place where the feedback control could be beneficial. Instead of strictly following the search order k_0, $k_0 - 1$, $k_0 + 1$, $k_0 - 2$, ..., it would be desirable to have some dynamic mechanism to identify unpromising hyperplanes and switch to possible better hyperplanes, so that a better suboptimal solution can be reached within the time bound.

In this paper, we consider the case where the time bounds imposed on solving are quite tight, and we have therefore chosen problem representation as the control parameter. To reduce the scale of the problem, we changed the problem representation by grouping several variables together. The solver performance on the new problem representation will be quite different with different grouping strategies. Although any grouping strategy might speed the solving process, some will cause big performance degradations in terms of the suboptimal solutions for the transformed problem. Instead of grouping variables together randomly, our algorithm groups pairs of variables with small values of the relative price heuristic:

$$\frac{c_j}{\sum_{i=1}^{m} \frac{a_{ij}}{b_i}}$$

Grouping with this heuristic allows for trading off between solution quality and solving speed.

To demonstrate that the optimal solver configuration is time-dependent, we describe experiments with the mk_gk11.dat benchmark (proposed by Glover and Kochenberger [1]). This benchmark problem instance includes 2500 variables and 100 constraints. Through variable grouping, we generated four new problem representations, in which the number of variables are 2000, 1500, 1000 and 500 respectively. For each representation, we used a reimplementation of the hybrid solver to solve the problem. Solution qualities (as fractions of the relaxed MKP01 solution) over time for one particular problem instance are shown in Figure 3. One can see that different time bounds lead to different optimal problem representation choices. For instance, if the solver is only allowed 50 seconds to solve the problem, the optimal problem representation should be the one with

1000 variables. For a 100-second time bound, the optimal choice becomes the representation with 2000 variables.

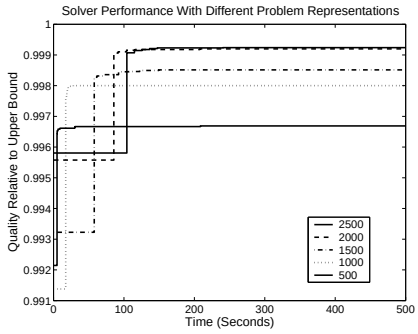

Fig. 3. The hybrid solver performance with different problem representations.

Previous work (Vasquez and Hao [33]) has demonstrated that the required solving time for a problem instance depends on its size, the number of variables and constraints in the instance. We further investigated whether other static features have an impact on the solver performance and can thereby provide a hint on selecting the control parameter (the problem representation, in this case). If offline analysis can be helpful in optimizing the solving process, the dynamic reconfiguration and switching overhead will be reduced. Several features of the knapsack problem were analyzed for their ability to predict how well the solver will perform (at the deadline) with each of the two problem representations. Example features included the tightness ratio $\frac{b_i}{\sum_{j=1}^{n} a_{ij}}$ where $i = 1, 2, ..., m$, and the relative price $\frac{c_j}{\sum_{i=1}^{m} \frac{a_{ij}}{b_i}}$. Linear regression results indicate that only the tightness ratio of the problem instance has an obvious effect on the solver performance, while other static features do not seem to yield any consistent indication of how the solver will perform.

This static feature analysis further demonstrates that static information alone is not enough for guiding the parameter choice. Experiments also show that for problem instances with the same tightness ratio the solver performance can still be quite different. Therefore, in this example our focus is on problem instances with the same tightness ratio and size (the number of variables and constraints in the instance). For such a group of problem instances, we build a dynamic model that can be used as a basis for making problem representation choices solely using solver runtime information.

We propose to use the solver runtime information (y) and the real-time application requirement (deadline T) to determine the right problem representation choice, using the feedback control paradigm. The best suboptimal solutions $z^*(k)$ reached at each sampling interval (i.e. the system performance outputs $y(k)$) are used for predicting the system performance and choosing the problem repre-

sentation [2]. We consider two different problem representations as our control parameter choices, $U = \{u_1, u_2\}$. One representation requires no problem transformation and presents the hybrid solver with the original problem, while the other representation transforms the problem through variable grouping.

4.3 Experimental Results

We implemented a benchmark generator according to the algorithm described in Chu and Beasley [9]. We then used it to generate problem instances that have $n = 500$ objects, $m = 30$ constraints and 0.25 tightness ratio. For these problem instances, we investigate two different problem representations. One representation will change the number of objects to $n = 420$ (chosen for demonstration purposes) by grouping 80 pairs of small relative price objects together. The other representation will retain the original problem formulation with $n = 500$ objects. The objective is to achieve the best suboptimal solution at a specified time bound with the Vasquez and Hao hybrid solver by solving the given problem instance with the feedback-chosen problem representation.

1100 problem instances were generated, with 1000 of them used as modeling instances to guide the feedback solving of the final 100 validation problem instances. The modeling and feedback control were done as described in Section 3. The sampling interval (feedback update period) was set to 10 iterations. The minimum number of intervals for each problem formulation was set to 40 (400 iterations) and the number of intervals after which the feedback was turned off and a single formulation was selected was 400 (4000 iterations). Two experiments were carried out, where different modeling and validation problem instances were chosen from the 1100 problem instance set. For reasons of simplicity, we chose 10000 solving iterations as the time bound, where a solving iteration is defined as the time interval between two successive moves of the local tabu search for the hybrid solver. Sample control traces, showing the controller's choice of problem representation at each iteration, are given in Figure 4.

To assess the the performance of the solver with and without feedback, we performed a paired comparison between the feedback-controlled solver and the no-feedback solver using each of the two problem formulations (420 and 500 variables). For this comparison, the two experiments (with different modeling and validation sets) were combined. Figure 5 shows the distribution of the differences in quality at the deadline between the feedback and no-feedback cases. Quality is measured as a fraction of the upper bound provided by the solution to the relaxed problem. The boxes indicate the central 50% of the data, while the whiskers denote the extent of the data. The gray bars show the 95% confidence interval around the mean.

The figure shows that with feedback control of the choice of problem representation, the solver performed better on average at the deadline than with

[2] Other possible dynamic information that could be used for feedback control includes the constraint violation $v_b = \sum_{i|a_i.x>b_i}(a_i.x - bi)$ and the time since the last improvement in the solution (stagnation).

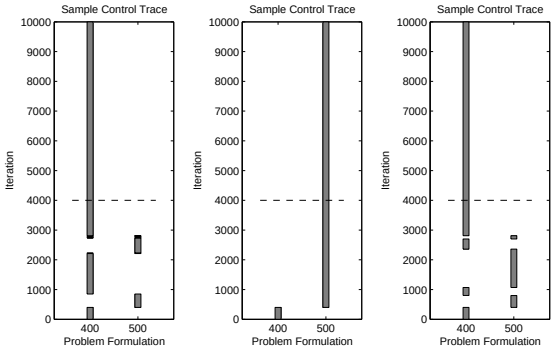

Fig. 4. Typical sample traces showing the feedback control switching between the two problem representations. The horizontal dashed line indicates the point beyond which no further switching was permitted.

either of the fixed problem representations. The improvement was small, but statistically highly significant ($p < 0.01$ in a binomial test for equal or better performance).

5 Conclusions and Future Work

In conclusion, we have here demonstrated an approach for using feedback control to improve the quality of the solution obtained when solving a problem under a strict time bound. The algorithm makes use of a model based on solution profiles to dynamically predict solver performance and choose solver control parameters accordingly. We have presented a case study application of this approach to solving the 0-1 multidimensional knapsack problem with the hybrid solver described by Vasquez, Hao, and Vimont [33, 34]. In this application, small solution quality improvements at the time bound were seen.

There are several issues still to be addressed with the feedback method described here. The modeling and prediction technique used can incur a large overhead if the size M of the modeling set is big. Thus, in the future it will be essential to carry out research on how to reduce the size of the modeling problem set by, for example, removing redundant similar-performance problems. Another future research topic is how to generate the modeling problem set so that it includes "representative" problems. The modeling set should be able to self-evolve during the solving process when new performance unique problem instances are identified. Further, analyzing the underlying reasons why two problems have similar performance profiles is a very interesting research topic. It would also be useful to explore how the models used here could transfer to broader problem classes. Other modeling and prediction methods are of course possible, as well, and bear further investigation.

The choice of problem granularity as control variable should also be explored further. Though a performance improvement was observed with the knapsack

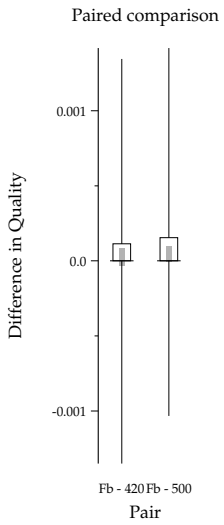

Fig. 5. Performance of the solver under feedback control, as compared to using each of the problem representations without control. The data plotted is the difference between the solution quality for the feedback case and the no-feedback case. The boxes indicate the region where 50% of the data points lie, and the whiskers show the extent of the data. The gray bars show the 95% confidence interval around the mean.

problem, it was not a very large one. It is possible that other classes of problems might lend themselves better to the approach of controlling the granularity by aggregating variables. Additionally, though we explored different ways to do the variable grouping, there may be better heuristics to use than relative price, and some of the other approaches to reducing granularity might be more fruitful than variable grouping.

Here we explored the use of on-line feedback control to choose solver and problem parameters. It would be interesting to integrate this technique with an off-line modeling and prediction approach based on, for example, problem instance features, such as that used by Nudelman and his co-authors [26, 22]. Such a combined approach might further improve performance at the deadline.

Feedback control of solving in response to a time deadline is a complex problem with many interacting variables, including choice of solver, choice of control parameters, choice of modeling and prediction techniques, and choice of control logic. If these obstacles can be overcome, however, the benefits of being able to provide high-quality solutions in real-time settings would be many.

6 Acknowledgments

The authors would like to thank Yi Shang, Hai Fang, Tarek Abdelzaher, John Stankovic, and Gang Tao for suggestions and helpful discussions. This work was partially supported by DARPA under contract F33615-01-C-1904.

References

1. Large benchmark, http://hces.bus.olemiss.edu/tools.html.
2. OR-library, http://mscmga.ms.ic.ac.uk/jeb/orlib/mknapinfo.html.
3. J. A. Allen and S. Minton. Selecting the right heuristic algoithm: runtime performance predictors. In *Advances in Artificial Intelligence. 11th Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, pages 41–53, Toronto, Ontario, May 1996.
4. S. Baluja, A.G. Barto, K.D. Boese, J. Boyan, W. Buntine, T. Carson, R. Caruana, D.J. Cook, S. Davies, T. Dean, T.G. Dietterich, P.J. Gmytrasiewicz, S. Hazlehurst, R. Impagliazzo, A.K. Jagota, K.E. Kim, A. McGovern, R. Moll, A.W. Moore, E. Moss, M. Mullin, A.R. Newton, B.S. Peters, T.J. Perkins, L. Sanchis, L. Su, C. Tseng, K. Tumer, X. Wang, and D.H. Wolpert. Statistical machine learning for large-scale optimization. *Neural Computing Surveys*, 3:1–58, 2000.
5. J. E. Borrett, E. P.K. Tsang, and N. R. Walsh. Adaptive constraint satisfaction: the quickest first principle. Technical Report CSM-256, University of Essex Department of Computer Science, 1995.
6. J. A. Boyan and A. W. Moore. Learning evaluation functions to improve optimization by local search. *Journal of Machine Learning Research*, 1:77–112, 2000.
7. Tom Carchrae and J. Christopher Beck. Low-knowledge algorithm control. In *Proceedings of AAAI-04*, pages 49–54. AAAI Press / The MIT Press, 2004.
8. Y. Caseau, F. Laburthe, and G. Silverstein. A meta-heuristic factory for vehicle routing problems. *Constraint Programming*, 1999.
9. P.C. Chu and J.E. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4:63–86, 1998.
10. Lara S. Crawford, Markus P.J. Fromherz, Christophe Guettier, and Yi Shang. A framework for on-line adaptive control of problem solving. In *CP'01 Workshop on On-line Combinatorial Problem Solving and Constraint Programming*, December 2001.
11. A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE transactions on evolutionary computation*, 3:124–141, 1999.
12. P. Flener, B. Hnich, and Z. Kızıltan. A meta-heuristic for subset problems. In *Practical Aspects of Declarative Languages. Third International Symposium, PADL 2001*, pages 274–287, Las Vegas, NV, March 2001.
13. Dominique Fortin and Ider Tsevendorj. Global optimization and multi knapsack: a percolation algorithm. Technical Report 3912, Institut National de Recherche en Informatique et en Automatique (INRIA), 2000.
14. J. Gratch and G. DeJong. COMPOSER: a probabilistic solution to the utility problem in speed-up learning. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 235–240, San Jose, CA, July 1992.
15. J. Gratch and G. DeJong. A decision-theoretic approach to adaptive problem solving. *Artificial Intelligence*, 88(1-2):101–142, 1996.
16. E. A. Hansen and S. Zilberstein. Monitoring the progress of anytime problem-solving. In *13th National Conference on Artificial Intelligence*, Portland, OR, August 1996.
17. B. Hnich and P. Flener. High-level reformulation of constraint programs. In *Proceedings of the Tenth International French Speaking Conference on Logic and Constraint Programming*, pages 75–89, 2001.
18. Robert C. Holte. Combinatorial auctions, knapsack problems, and hill-climbing search. In *Proceedings of AI'2001, the Fourteenth Canadian Conference on Artificial Intelligence*. Springer, 2001.

19. Holger H. Hoos. An adaptive noise mechanism for WalkSAT. In *Proceedings of AAAI-02*, pages 655–660. AAAI Press / The MIT Press, 2002.
20. E. Horvitz, Y. Ruan, C. Gomes, H. Kautz, B. Selman, and M. Chickering. A Bayesian approach to tackling hard computational problems. In *Proceedings of the Seventeenth Conference on Uncertainty and Artificial Intelligence*, Seattle, WA, August 2001.
21. M. G. Lagoudakis and M. L. Littman. Learning to select branching rules in the DPLL procedure for satisfiability. In *LICS 2001 workshop on theory and appliations of satisfiability testing (SAT 2001)*, 2001.
22. Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. Learning the empirical hardness of optimization problems: the case of combinatorial auctions. In *Constraint Programming 2002*, 2002.
23. L. Lobjois and M. Lemaître. Branch and bound algorithm selection by performance prediction. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 353–358, Madison, WI, July 1998.
24. S. Minton. Automatically configuring constraint satisfaction programs: a case study. *Constraints*, 1(1-2):7–43, 1996.
25. A. Narayek. An empirical analysis of weight-adaptation strategies for neighborhoods of heuristics. In *Proceedings of the fourth metaheuristics international conference*, pages 211–216, 2001.
26. Eugene Nudelman, Alex Devkar, Yoav Shoham, and Kevin Leyton-Bown. Understanding random SAT: Beyond the clauses-to-variables ratio. In *Proceedings of Constraint Programming 2004*, 2004. to appear.
27. D. J. Patterson and H. Kautz. *Auto-walksat*: a self-tuning implementation of *walksat*. *Electronic Notes in Discrete Mathematics*, 9, 2001.
28. J. C. Pemberton and W. Zhang. ϵ-transformation: exploiting phase transitions to solve combinatorial optimization problems. *Artificial Intelligence*, 81(1-2):297–325, 1996.
29. Y. Ruan, E. Horvitz, and H. Kautz. Restart policies with dependence among runs: a dynamic programming approach. In *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP-2002)*. Springer-Verlag, 2002.
30. Y. Ruan, E. Horvitz, and H. Kautz. Hardness-aware restart policies. In *IJCAI-03 Workshop on Stochastic Search Algorithms*, Alcapulco, Mexico, 2003.
31. W. Ruml. Incomplete tree search using adaptive probing. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 235–241, Seattle, WA, August 2001.
32. Yi Shang and Markus P.J. Fromherz. Experimental complexity analysis of continuous constraint satisfaction problems. *Information Sciences*, 153:1–36, 2003.
33. Michel Vasquez and Jin-Kao Hao. A hybrid approach for the 0-1 multidimensional knapsack problem. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 328–333, Seattle, WA, August 2001.
34. Michel Vasquez and Yannick Vimont. Improved results on the 0-1 multi dimensional knapsack problem. In *Sixteenth Triennial Conference of the International Federation of Operational Research Societies (IFORS 2002)*, 2002.
35. B. W. Wah and T. Wang. Tuning strategies in constrained simulated annealing for nonlinear global optimization. *International Journal of Artificial Intelligence Tools*, 9(1), 2000.