A Framework for On-line Adaptive Control of Problem Solving*

Lara S. Crawford, Markus P.J. Fromherz, Christophe Guettier, and Yi Shang
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304, U.S.A.
{lcrawford, fromherz, guettier, yshang}@parc.xerox.com

Abstract

The design of a problem solver for a particular problem depends on the problem type, the system resources, and the application requirements, as well as the specific problem instance. The difficulty in matching a solver to a problem can be ameliorated through the use of on-line adaptive control of solving. In this approach, the solver or problem representation selection and parameters are defined appropriately to the problem structure, environment models, and dynamic performance information, and the rules or model underlying this decision are adapted dynamically. This paper presents a general framework for the adaptive control of solving and discusses the relationship of this framework both to adaptive techniques in control theory and to the existing adaptive solving literature. Experimental examples are presented to illustrate the possible uses of solver control.

1 Introduction

Real-world problem solving typically involves selecting both a problem representation and a solving algorithm. As is well known, both of these choices are critical to the success of the problem solution, and both choices may depend on specific aspects of the application problem, including operational and resource constraints. For example, some solvers may work better on under-constrained problems, while others are better choices for over-constrained problems. These issues exist for combinatorial problems, constraint programming problems, continuous problems, and mixed problems. They become particularly imposing for embedded, large-scale problems. Often, application information is only partially available at design time. For example, one of the most promising applications for embedded constraint solving is in model-predictive control of large-scale reconfigurable systems. Model-predictive control is important for reconfiguration, as it is explicitly model-based and models can be easily swapped in and out. This means, however, that the constraint problem to be solved may change over time.

One approach to this design issue is the use of adaptive control of problem solving. We define adaptive control of solving as modifying the solver or the problem representation in response to problem structure, resource limits, or on-line performance, and updating the basis for this set of decisions adaptively over time. Such adaptation allows a general solver to be used for a wide variety of problems with varying time and resource constraints; the solver can choose the best techniques for the circumstances and adjust its choices if progress is not satisfactory.

Adaptive solving control makes use of a model or set of rules concerning the relationship between various problem, system, and application parameters and best choices for solvers, heuristics, and problem transformations. These choices define the control parameters of the solver. This model or rule base also enables the prediction of the behavior of the solver defined by these control parameters. The predicted behavior can be used to monitor the actual on-line behavior of the solver and update the control parameters accordingly. If the predicted behavior is consistently at odds with the actual behavior, the accumulated performance feedback can be used to update the model or rule base itself. This approach is similar to some commonly used adaptive control techniques in control theory.

In this paper, we introduce a framework for the discussion of adaptive solving control systems. This framework is discussed in Section 2. The framework is motivated by a discussion of typical adaptive control techniques. This section also discusses some of the literature on adaptive solving in the context of the proposed framework. Section 3 gives some concrete, experimental examples for situations in which the proposed types of adaptive solving control would be useful. Conclusions are presented in Section 4.

*Published in CP'01 Workshop on On-line Combinatorial Problem Solving and Constraint Programming, Dec. 2001. This work was partially supported by DARPA under contract F33615-01-C-1904.

2 Adaptive Solving Strategies

In the field of control systems, the idea of on-line adaptation has already gone through many years of development (e.g., [3, 17]). In standard adaptive control, it is assumed that the controller has been designed off-line, but may contain some uncertain parameters. Two formulations are possible: *indirect adaptive control*, in which the uncertain parameters are actually parameters of an uncertain plant model that appear in the controller design, or *direct adaptive control*, in which the parameters are simply controller parameters (the best choice of which will still depend on the precise plant model). The purpose of *adaptive* control is to adapt the uncertain parameters on-line based on the system performance.

2.1 Adaptive Control

The elements of a typical self-tuning adaptive control system are shown in Figure 1. The system input is a reference signal, r, describing a goal for the system behavior (a desired trajectory or set point, for example). The controller supplies a control signal u to the plant, which produces output y. This plant output is fed back to the controller, closing the control loop. This closed-loop feedback control allows the controller to react to the actual output of the system, y, rather than giving inputs based solely on its knowledge of the plant model and of r. The portion of the diagram described so far, the controller and the plant connected in a feedback loop, is a complete control system in itself. The adaptation is performed in the outer loop of the system as follows. The control signal u and the plant output y are fed into the adaptation block. In a typical self-tuning regulator, this block performs system estimation to more accurately identify the plant model. The updated parameters in this model are either plugged directly into the controller (direct adaptive control) or are used in a plant model that in turn is used to define updated parameters for the controller (indirect adaptive control). The output of the update rule is the current estimate $\hat{a}$ of the best parameters. The adaptive control loop, which is typically much slower acting than the feedback loop, thus allows the system to improve its performance over time.

One thing to note about the adaptive control framework is that there are three levels of increasingly sophisticated control in the system:

1. Open-loop control. In a basic control system, the controller could take the reference input and output a control signal to the plant without any feedback. For such a feed-forward, open-loop control system to produce the desired behavior, the controller must be designed and tuned off-line using a very accurate model of the plant. Any inaccuracies in the model or disturbances to the system (noise or environmental disturbances) will not be taken into account at run-time.

2. Feedback control. When the feedback loop is added to the control system, the controller gains the ability to react to the quality of the plant's output. The controller still must be designed and tuned off-line, but the system has a much better chance of performing as desired.

3. Adaptive control. In an adaptive system, the controller is able to adapt to compensate for systematic performance errors caused by inaccurate plant modeling. The controller is designed and tuned off-line, but can be formulated parametrically in terms of plant parameters that may be unknown or imprecisely known, for example. The model becomes more accurate over time and can also adapt to changes in the plant.

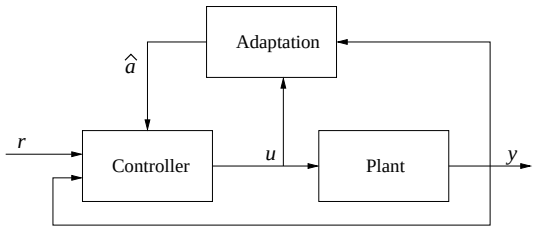


Figure 1: Typical Adaptive Control Scheme

2.2 On-line Adaptive Control of Solving

Based on this adaptive control viewpoint, we propose a generic framework for the adaptive control of solving, shown in Figure 2. This functional framework is useful for structuring a principled discussion of adaptive solving approaches. Examples of possible implementations and instantiations of its elements are provided in Section 2.4.

The Solver Module plays the role of the plant to be controlled. The Problem Transformer and the Constraint Solver both represent general systems, possibly containing multiple algorithms and heuristics among which to select, as well as tunable parameters. The algorithms and parameters are selected with the input u, which contains both u_S and u_T, provided by the Control Module. This module makes the solver and transformer algorithm selection and tuning decisions based on the environment specification, E, the problem to be solved, P, the on-line (during a run) behavior of the solver, y, and some internal tuning parameters, a.

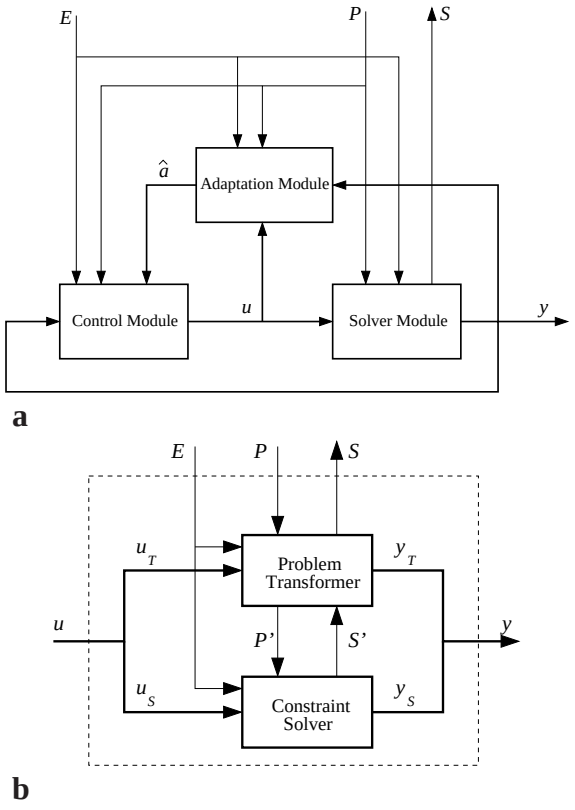


Figure 2: Adaptive Solving System Framework. The top diagram (a) shows an overview, and the bottom (b) shows an expanded view of the Solver Module.

Including the on-line behavior of the solver as an input to the Control Module allows the solver control to have a feedback component. The Adaptation Module has access to E, P, u, and y. Its role is to monitor the behavior y of the Control Module/Problem Transformer/Constraint Solver system and adapt the parameters, a, of the Control Module to improve performance. Its outputs, $\hat{a}$, are the current estimates of the optimal Control Module parameters. The Adaptation Module would typically act at a much slower time scale than the feedback loop. As it actually modifies parameters of the Control Module, its actions would generally not be based on system behavior for a single problem or subproblem, as the feedback control would.

The environment, E, includes models of both the system specifications and the application requirements. Some of this environment information may also be available to the solver directly. The problem specification, P, is also supplied to the Problem Transformer. The resulting problem transformation, P', is supplied to the solver. The transformed solution to the problem, S', is then passed back through the transformer to recover the solution, S, to the original problem.

Both the Control Module and the Adaptation Module contain, either explicitly or implicitly, information about the solver and its expected behavior. An example of an implicit representation would be in the form of a rule base. On the other hand, one way of explicitly representing the solver information is as a solver model. A predictive model of expected solver behavior would provide a basis for the Control Module to make its solver and transformer parameter choices. The Adaptation Module would use the predicted behavior as a yardstick for measuring the true behavior, y. The controller parameters a adjusted by the Adaptation Module might then be parameters of the solver model.

2.3 Related Work on Adaptive Solving

Previous approaches to adaptive solving for different classes of problems have involved the first of the levels of control enumerated above, and sometimes the second, but not, as far as we are aware, the third. For example, a number of systems, such as Minton's MULTI-TAC [20], Gratch and DeJong's COMPOSER [13, 12], or Caseau *et al.*'s meta-heuristic factory [7], use 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 (defined by u_S and u_T in Figure 2) are not responsive to the on-line performance of the system. The same is true for approaches such as that in [10], in which a model is built off-line defining the relationship between parameters describing the problem instance and the best set of heuristics to use. There are several similar approaches to on-line algorithm or heuristic selection [2, 18]. 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, once the selection is made, no further feedback is used.

There are a number of approaches that make more use of feedback-type information. Borrett *et al.* [5] use on-line performance feedback to switch between algorithms. Horvitz *et al.* [15] use it as part of a dynamic restart policy. There are also a variety of approaches that dynamically build up estimates of value or cost functions to guide the search [4, 6, 21, 23]. These functions are measurements of the "goodness" of particular states or action choices, and are developed on-line using accumulated performance data. In the evolutionary algorithms community, a variety of techniques have been used to adapt genetic operators and parameters based on various performance measures [8]. Similar approaches have been used with other techniques, such

as simulated annealing [24].

Such techniques have also been used to modify the problem representation. An "open-loop" off-line design approach for problem reformulation has been proposed by Hnich and Flener [14]. Feedback approaches have been used as well. For example, Pemberton and Zhang [22] 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 [8].

Although some of these approaches can make use of a model of the relationship between the problem type and the best solver parameters, and some can change these parameters based on performance feedback, these techniques do not have a mechanism for "remembering" anything from experience, in terms of updating the predictive model. Fink's work [9], in which an open-loop statistical model predicting success or failure of a search is built up on-line over many search problems, is an exception. In this sense, most of the work on adaptive solving to date has in fact not been adaptive, at least not in the control sense. In other words, these approaches are not adaptive to the run-time environment of the solver, a crucial prerequisite for deploying solvers in embedded applications.

2.4 Enabling Adaptive Solver Control

The solver framework we have proposed incorporates these three levels of solver control or adaptivity. We suggest that including all three levels in a single system can improve performance over a range of problems significantly. Many strategies that fit this adaptive framework are possible. For example, an *open-loop strategy* could be based on the size and constraint-to-variable ratio (constraint ratio) of the problem. u_S could, in this case, indicate which solver (or solvers) to use for the problem, for how many iterations or to what depth to run it, which heuristics to use, and so on. u_T could define the granularity or a particular type of continuous-to-discrete or discrete-to-continuous transformation. This information could be encapsulated in a solver model and would represent the best solution for the average problem of a given size with a given constraint ratio. This model could be developed off-line through a system like that shown in Figure 3. There, an Analysis Module tests a Problem Ensemble on the Solver Module with a variety of parameters u and monitors the solver behavior y. The result of the analysis is a set of configuration parameters learned for average or important cases, which go into the Solver Model. The basis or input metrics for classifying problems in the model (constraint ratio, for example) can be determined through understanding and analysis or through a learning approach such as that in Horvitz *et al.* [15].

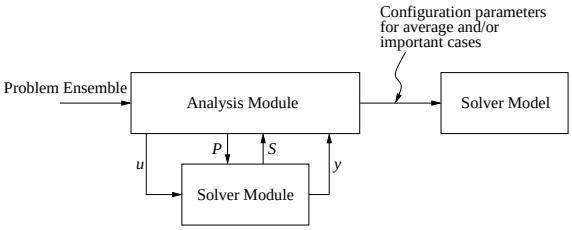


Figure 3: Analysis and Model Generation

The same approach can be used to design a system that includes the second level of control sophistication, *performance feedback*. The performance information y might be based on such measures as the on-line behavior of an objective function, the elapsed time, the number of constraints satisfied, or the degree of constraint violation, as well as solver state information such as the number of no-goods, the degree of thrashing, and so on. The off-line design of the Solver Model can incorporate this performance information into the decision process determining u_S and u_T. The model would also provide an expected behavior with which to compare the actual behavior, y. Then, as the solver is executed, the parameters u_S and u_T would be adjusted if the actual behavior deviates from expected behavior, e.g., by increasing search depth or iteration bounds.

Finally, such a system could also incorporate the third level of control sophistication described above, *adaptation*. The predictions of the Solver Model can be used to identify persistent deviations from expected behavior over the long term. This information can then be used by the Adaptation Module to modify the parameters of the model, making it more accurate. For example, the constraint ratio threshold for choosing one solver over another could be shifted if it emerged that the incorrect solver was being chosen repeatedly. In applications such as model-predictive control or dynamic, on-line scheduling, similar problems are solved over and over, so this type of on-line adaptation would be quite beneficial in refining the solver control scheme.

One caveat is that the system for adaptive control of solving must be continually exposed to the full range of problems and environmental conditions for which it is designed. In adaptive control systems, this requirement is referred to as the need for persistent excitation [3]. If the problem and environment specifications do not meet this criterion, the controller or model parameters a may get stuck at values that may appear satisfactory for the subset of problems and environments to

which the system has been exposed, but that may in fact be sub-optimal. The exposure to the complete range of E and P of interest must continue over time, as well, so that the system does not "forget" what it has learned earlier.

This section discussed the proposed framework for the adaptive control of solving and gave some examples of control parameters u and behavioral outputs y. The next section provides some specific examples portraying cases in which such adaptive control of solving might be useful.

3 Examples

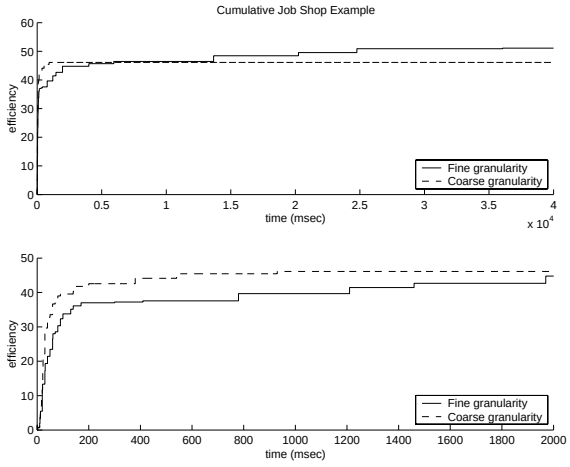


Figure 4: Scheduling Problem with Cumulative Resources Using Different Granularities. This figure shows the average solution efficiency over 9 runs as a function of time (the bottom plot is an expansion), using a CLP(FD) solver with an anytime search strategy. The problem involves 10 tasks and a resource limit of 40. Other parameters are randomly generated. The task length and horizon of the fine granularity are 5 times those in the coarse one.

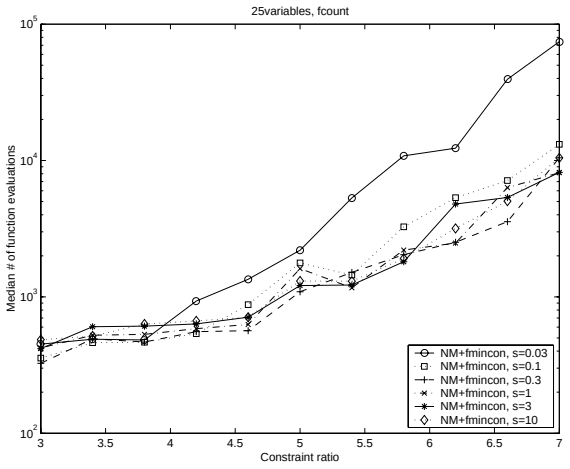
As discussed above, the literature is already in agreement as to why the open-loop level of solving control is useful, so here we will focus on some examples motivating the need for feedback and adaptive control of solving. These examples indicate that on-line solving might be improved by including feedback and adaptation (in the control sense), but do not yet constitute conclusive evidence.

One example in the arena of time-aware solving involves choosing a problem representation to maximize the solving efficiency. Figure 4 shows data on solving a standard scheduling problem [1] with two different granularities for the time representation. The figure displays the average efficiency obtained by the solver as a function of time. If enough time is available (as specified by the application requirements in E), the finer granularity is clearly the better choice, as it allows for a superior solution in the long run. If, however, less time is available, the coarser granularity may be better, as its solution quality has a faster initial rise. A time-aware solver could make use of such data as part of its solver model, and choose the granularity according to the time available. This type of decision would form part of an open-loop solver control scheme, according to the framework described above.

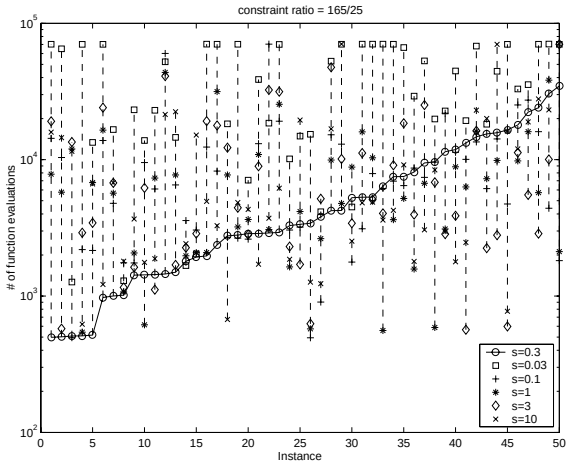
In addition, as the time limit (deadline) for solving approaches, the Control Module could monitor the improvement of the solution quality. If the quality is not reaching the expected levels predicted by the solver model, or if it is not improving fast enough, the Control Module might elect to switch to a problem representation with lower granularity, in hopes of a steep performance improvement, given the limited time remaining. Such a scheme, based on monitoring the output of the solver, falls in the category of feedback control of solving.

As a second example, consider a cooperative global/local solver for continuous constraint satisfaction problems [11] that uses the Nelder-Mead algorithm [16] for global search followed by sequential quadratic programming for local search (fminsearch and fmincon, respectively, in the Matlab Optimization Toolbox [19]). Such a solver has several tuning parameters that need to be set. One of these is the size of the initial simplex for the Nelder-Mead algorithm, which determines the search scale. As the constraint ratio seems to be an important indicator of complexity for this type of problem, the first step might be to analyze the performance of various initial simplex sizes for various constraint ratios, as is shown in Figure 5a for a 25-variable problem. Such an analysis provides the basis for a solver model, and as such the foundation for an open-loop control scheme. This scheme specifies that the simplex size should be chosen as the one yielding the lowest median complexity given the problem's constraint ratio.

If, for instance, for a problem with constraint ratio 6.6, the simplex size were chosen based on Figure 5a, a size of 0.3 would be selected (the lowest point in the second-to-last set of data). Since this is the best choice only on average for this particular constraint ratio, it will not be the best choice for every problem instance. Figure 5b shows the complexity for all 50 different instances for this constraint ratio. The instances have been ordered by the complexity for the open-loop simplex size, 0.3. Clearly, there are a number of instances where different choices of the simplex would yield much better results. Therefore, a feedback law



a



b

Figure 5: Continuous Constraint Satisfaction Problems with Different Initial Simplex Sizes. This figure shows data on sum-of-sines type continuous constraint satisfaction problems with 25 variables. The top diagram shows the median (over 50 instances) of the number of function evaluations required for problems with different constraint ratios, using initial simplex sizes s between 0.03 and 10. The bottom diagram shows, for a single constraint ratio, the variation among all instances and all simplex sizes.

would be useful here in order to fine tune the system for the particular instance. When monitoring the online solver behavior, if the number of function evaluations were growing too large, the simplex size could be changed. This type of feedback could improve the solving time in cases like those on the right-hand side of Figure 5b.

Finally, if, over the course of many problems, the initial open-loop estimate of the best simplex size is in error more often than not, the adaptation portion of the solver control comes into play. The Adaptation Module can monitor the behavior of the solver and compare it to that predicted by the solver model (here, the median complexity given the constraint ratio and the choice of simplex size). If this prediction appears to be incorrect, such as if the simplex size of 0.3 is yielding consistently worse results (and the feedback law described above has to act frequently), adaptation can adjust the solver model accordingly. Therefore, it may happen that, as more examples are gathered on-line, the estimate of the complexity for the 0.3 solver increases enough that 0.3 is no longer the preferred simplex size for problems with constraint ratio 6.6.

Thus, open-loop control implements parameter choices based on a solver model or a set of rules, either of which can be generated off-line based on analysis or statistical sampling. The feedback control component performs on-line corrections to compensate for individual instances deviating from the norm. Finally, the adaptive control component serves to correct persistent errors in the solver model or rule base, which can affect both the open-loop and feedback components.

4 Conclusion

The framework presented here is, as far as we are aware, the first principled classification of different techniques for adaptive control of problem solving. The motivation and discussion of this adaptive solving framework makes use of control theory as an analogical model. Most work on "adaptive solving" in the literature is not, in fact, adaptive in the control sense, but rather incorporates only the open-loop or feedback levels of solver control. We believe that the issues of problem representation and algorithm selection and tuning addressed here exist for a wide variety of applications and are particularly important for large-scale, dynamic, embedded problems.

References

[1] A. Aggoun and N. Beldiceanu. Extending CHIP in order to solve complex scheduling and placement problems. *Journal of Mathematical and Computer Modelling*, 17(7), 1993.

[2] 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.

[3] K. J. Åström and B. Wittenmark. *Adaptive Control*. Addison-Wesley, 1995.

[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] Y. Caseau, F. Laburthe, and G. Silverstein. A meta-heuristic factory for vehicle routing problems. *Constraint Programming*, 1999.

[8] A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE transactions on evolutionary computation*, 3:124–141, 1999.

[9] E. Fink. How to solve it automatically: selection among problem-solving methods. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, pages 128–136, Pittsburgh, PA, 1998.

[10] 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.

[11] M. P. J. Fromherz, T. Hogg, Y. Shang, and W. B. Jackson. Modular robot control and continuous constraint satisfaction. In *Proc. IJCAI-01 Workshop on Modelling and Solving Problems with Constraints*, pages 47–56, Seattle, WA, 2001.

[12] 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.

[13] J. Gratch and G. DeJong. A decision-theoretic approach to adaptive problem solving. *Artificial Intelligence*, 88(1-2):101–142, 1996.

[14] 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.

[15] 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.

[16] J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright. Convergence properties of the Nelder-Mead simplex algorithm in low dimensions. *SIAM Journal on Optimization*, 9:112–147, 1998.

[17] W. S. Levine, editor. *The Control Handbook*. CRC Press, 1996.

[18] 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.

[19] *Matlab Optimization Toolbox*. The MathWorks, Inc., 2001.

[20] S. Minton. Automatically configuring constraint satisfaction programs: a case study. *Constraints*, 1(1-2):7–43, 1996.

[21] A. Narayek. An empirical analysis of weight-adaptation strategies for neighborhoods of heuristics. In *Proceedings of the fourth metaheuristics international conference*, 2001. To appear.

[22] J. C. Pemberton and W. Zhang. ϵ-transformation: exploiting phase transitions to solve combinatorial optimization problems. *Artificial Intelligence*, 81(1-2):297–325, 1996.

[23] 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.

[24] 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.