A Decision Theoretic Meta-Reasoner for Constraint Optimization

Jingfang Zheng and Michael C. Horsch

Department of Computer Science
University of Saskatchewan
Saskatoon, Saskatchewan, Canada

Abstract. Solving constraint optimization problems is hard because it is not enough to find the best solution; an algorithm does not know a candidate is the best solution until it has proven that there are no better solutions. The proof can be long, compared to the time spent to find a good solution. In the cases where there are resource bounds, the proof of optimality may not be achievable and a tradeoff needs to be made between the solution quality and the cost due to the time delay. We propose a decision theoretic meta-reasoning-guided COP solver to address this issue. By choosing the action with the estimated maximal expected utility, the meta-reasoner finds a stopping point with a good tradeoff between the solution quality and the time cost.

1. Introduction

Constraint optimization problems (COPs) can be very much harder than solving constraint satisfaction problems (CSPs), because CSP solving algorithms can stop once a solution is obtained; but for COP solving, unless the optimal cost is known before hand, an algorithm that optimizes COPs cannot stop until it has proven that a solution is optimal. Even problems of modest size can be costly in terms of time. For some applications, the time cost may be as important as the solution quality.

A simple approach to this tradeoff is to spend as much time as is allowed by the application. In effect, this approach amounts to spending the user's entire budget for computation. When time is cheap, this approach may be effective, but when time is costly, a user may prefer a good solution sooner than a better solution later. Another approach is to search for a solution whose quality is no less than a given quality. This approach may result in solutions whose quality could be improved with a little more computation, or as before, solutions whose quality is not justified by the expense of the computation. By explicitly considering the costs and benefits of computation, a system may be able to optimize the comprehensive value of a solution, namely the quality net of computational costs.

In this paper we present the design of a practical COP solver that uses decision theoretic meta-reasoning to control computation. In this approach, computational actions are associated with utilities [Horvitz 1989, Russell & Wefald 1991]. The value of the computational action is the solution quality that results, and the cost is associated with resources used in the solving (e.g., time). We apply this approach by monitoring the status of the solver and deciding to halt when the solver seems trapped

in a proof of optimality. Since different users have different requirements for time and solution quality, we allow for different user preference models. Our results show that the meta-reasoning solver obtains a good trade-off when resource costs are high.

Our work differs from that of Horvitz et al. [2001], in that the decision problem is different. For a satisfiability problem, every solution is equally valuable, and the decision problem faced by a stochastic local search method is to choose whether to restart the search, or carry on from the current location. In a COP, solutions vary in quality, and the tradeoff is more flexible since the value of the solution is part of the decision.

Many of the issues addressed in the domain of planning under uncertainty (eg, [Boddy and Dean, 1994] and [Dean et al., 1995]) arise in constraint optimization and soft constraint propagation. The main difference lies in the information available during deliberation, and the extent to which the representation provides structure to the meta-reasoner. Our approach does not exploit the structure of the COP as much as is done in the planning domain.

To evaluate our approach, we focus on the problem of finding an assignment that violates the fewest number of constraints, i.e., Max-CSP, when all constraints are binary. However, our approach generalizes to any COP that can be expressed as a Valued-CSP (VCSP), or equivalently, a Semiring-based CSP (SCSP) [Bistarelli et al. 1996]. The details of these representations are not important for the purposes of this paper. However, we refer the reader to [Zheng and Horsch 2003] for details concerning the COP solver used in this study.

1.3 Decision theoretic meta-reasoning

In problems like CSPs and COPs, there is always uncertainty in the solving process. Horvitz [1988] summarizes the sources of uncertainty during computation: the value of alternative computed results in a particular situation, the difficulty of generating results from a problem instance, and the costs and availability of resources (such as time) required for reasoning. Meta-reasoning refers to the deliberation concerning possible changes to the computational state of an agent [Russell & Welfald 1991]. More concisely, it is the reasoning *about* computation.

The uncertain trade-off between the costs and benefits of a computation can be modeled with decision theory. A decision theoretic meta-reasoner tries to determine the object-level computation that maximizes the agent's expected utility, considering the trade-off explicitly. The comprehensive utility *uc* refers to the net value associated with the commitment to a computation. Comprehensive utility can be decomposed into two components: the object-level utility *uo* and the inference-related utility *ui*. The object level utility *uo* of a strategy is the utility of the outcome, omitting the costs associated with computation. The inference-related utility *ui* includes the costs that are be involved in the computation, such as time cost, memory cost and network cost, etc. The relation between these 3 utilities can be represented by: $uc = f(uo, ui)$. In many cases, f can be treated as additively separable: $f(uo,ui) = g(uo) + h(ui)$ for some functions g and h. We assume that f is separable in this way.

We make the further simplification of assuming that the on-line cost of meta-reasoning is negligible, by designing our meta-reasoning to have negligible costs,

compared to the object-level algorithm. In our approach, there are non-negligible costs off-line in compilation and analysis, which we assume can be amortized over the use of a meta-reasoning system, but we ensure that on-line costs are negligible.

A simple approach to meta-reasoning is due to [Russell & Wefald, 1991]. Suppose the system maintains a "current best answer" *a*, which will be returned if it is interrupted during inference. The work of the future computation is to refine *a* for a higher utility. The algorithm is given as follows:

Step 1. Keep performing the object-level computation with highest expected net value (*uc*), until none has positive expected net value.

Step 2. Return answer *a* that is preferred according to Step 1.

More assumptions can be made to simplify the estimated computations [Russell & Wefald, 1991]. Meta-greedy algorithms consider only single computational steps, estimate their ultimate effect, and then choose the step that appears to have the highest benefit. The single-step assumption assumes the value of a partial computation as a complete computation, as if the system only had time for one more complete computation step. This assumption can cause underestimation of the value of some computations. The expensive alternative is to search for an optimal sequence of steps.

2. A Meta-Reasoning COP Solver

The task of computing the expected utility *uc* for each action is hard because of the uncertainty in the results of computation. A meta-reasoning agent should select its current best action by making explicit numerical estimates of the utilities of action outcomes. Statistical knowledge of the probability distributions over the results of computation can be used for future utility estimates of actions. Thus, our meta-reasoning system consists of

- branch-and-bound search combined with consistency propagation in VCSPs
- a statistical model for the outcome of a computational step
- a user preference model of the costs of computation, and the value of a solution

2.1 The branch-and-bound search method

Recent research has been devoted to building COP frameworks extended from constraint satisfaction problem (CSP) frameworks, including valued-CSPs (VCSPs) [Schiex et al. 1995] and Semiring-based CSPs (SCSPs) [Bistarelli et al. 1996]. Early approaches to COP solving use partial consistency propagation, combined with branch and bound search, such as Russian Doll Search [Verfaillie et al. 1996] and partial consistency propagation [Schiex et al. 1995]. More recently, Schiex [2000] proposed a definition of node and arc consistency in a VCSP framework.

The constraint propagation we use is based on Larrosa's variation [2002] of Schiex's approach, as implemented in [Zheng and Horsch 2003]. It consists of the systematic repetition of *projections* of constraint costs from the (binary) constraints in *C* to unary constraints over variables involved in the constraint, and then to a special 0-ary constraint for the entire COP instance. This method achieves node and arc

consistency in a VCSP framework, and can also prune the inconsistent values from variables' domains. The 0-ary constraint gathers the projected costs from the binary constraints and the unary constraints, and becomes a good lower bound (*lb*) for use in branch-and-bound search. The upper bound in the search is the valuation of the current best solution. Furthermore, the unary constraints on each variable provide a value-ordering heuristic, which has been shown to be effective [Zheng & Horsch 2003]. This combination of soft arc consistency, with branch-and-bound search, using the value ordering heuristic (BB-SRFL-H) is the object-level solver.

The solving algorithm is parameterized by a time bound that acts as a hard deadline: once the time bound is reached, the solving will stop, reporting the current best solution. The solving algorithm can also be interrupted by the meta-reasoner, which can halt the solver, and report the current best solution, even if the total time is not reached. The solver reports data to the meta-reasoner at regular intervals, as well as when the current best solution is updated. This data is outlined below.

2.2 The meta-reasoning problem

The objective of meta-reasoning, as mentioned above, is to maximize the expected comprehensive utility *uc*, which is a combination of object-level utility *uo* and inference related utility *ui*. In this system, the object-level utility *uo* is defined as the value of the solution quality, and the inference-related utility *ui* is the cost of achieving *uo*, mainly the required time. With the assumption that these two utilities can be separated additively, the relation is simply $uc = g(uo) + h(ui)$. This equation can be expressed in terms of cost. Suppose *cc*, *co* and *ci* are respectively the cost for *uc*, *uo* and *ui* ($cc = -uc$, $co = -uo$ and $ci = -ui$). The objective of maximizing *uc* is the objective of minimizing *cc*.

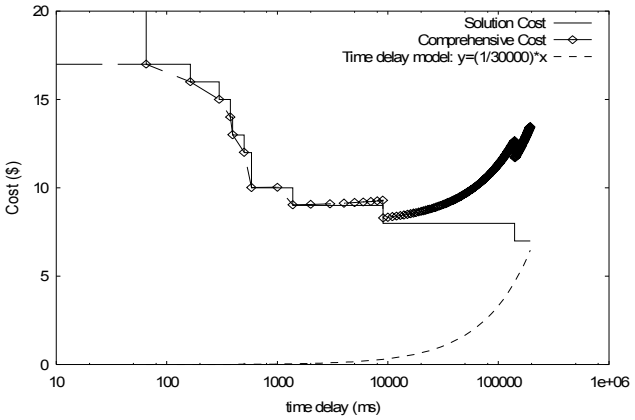

Fig. 1. An example graph of the combined cost *cc* from solution cost *co* and time cost *ci*. The optimal *cc* occurs just before the 10 second point.

To compute *cc*, we have to choose functions *g* and *h*. Here we use "dollar ($)" units to describe the costs, just for convenience. For the object-level cost *co*, (the solution cost), we suppose the violation of any constraint costs $1. The cost function

for *ci* will depend on the user. For example, every 30 seconds of time delay costs the user $1. These are arbitrary choices that do not affect the design of the meta-reasoner, and a later section will give a detailed discussion on user preference models.

The task for the meta-reasoner is to analyze data at short intervals, to predict or observe the point when the minimal *cc* (maximal *uc*) is achieved, at which point it should make the decision to halt computation. Figure 1 shows the graph of *cc* based on a solved problem. Initially, the solution improves quickly and the increase of the time cost was insignificant, so the combined cost cc decreases with the update of solutions. As time goes on, *cc* will increase if there is no update. But a new update would still reduce *cc*. The figure shows the ideal stopping point at the place of the minimal *cc*: the point after an update and before a long "no-update" period was about to start. This long interval would accumulate a high time cost, and even a new update to the solution does not pay off.

This ideal analysis was obtained in retrospect using a solved problem. However, for most problems, the solver cannot be certain if there will be another quick update after the update that it just found. The task of our meta-reasoner is to predict the stop point that achieves the expected maximal comprehensive utility *uc*. The meta-greedy assumption is used to simplify the situation: the system will consider only one step at every meta-reasoning point. Thus, our system takes the estimation of maximizing the next step's utility instead of directly maximizing the final decision's utility. At every short interval, say 1 second, the solver will pause and the meta-reasoner will consider, as we outline below, whether or not to let the object level solver continue.

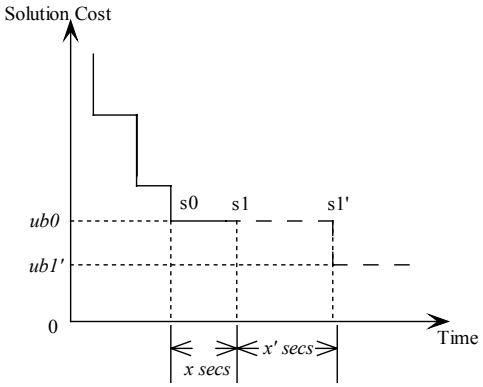

Fig. 2. A visualization for the notation that will be used throughout the method.

The following notation is used to derive the meta-reasoning process:

- *s0*: the previous state at which the solution was updated most recently;
- *ub0*: the cost of the best solution (upper bound) at *s0*;
- *s1*: the current meta-reasoning state; we assume that there is no update from *s0* (from the definition of *s0*), and the solution cost for *s1* is still *ub0*.
- *x*: the number of meta-reasoning decision intervals passed from *s0* to *s1*. If meta-reasoning is performed every second, the time between *s0* and *s1* is *x* seconds.
- *s1'*: the state that the meta-reasoner predicts to have the next update after *s1*.

- *ub1'*: the next update of solution cost, or the solution cost at *s1'*;
- *x'*: the time from the current meta-reasoning state s1 to the next update state *s1'*
- *Uc*: the utility function for *uc*
- *Uo*: the utility function for *uo*
- *Ui*: the utility function for *ui*

With the knowledge of how much the update will cost the user (from *ub0* to the predicted *ub1'*), the task for the meta-reasoner is to decide whether to continue from *s1* to *s1'*, based on the knowledge that it already has, and its prediction of how long *s1* to *s1'* will take, and whether it will be less than the cost the user is willing to accept.

The meta-reasoner will say "continue" if it believes in an increase of *uc*, and on the contrary "halt". The change of *uc* brought by the action of "continue" can be expressed in Equation 2 (from *s0* to *s1'*).

$$U_c(continue) = U_c(s1') \quad (1)$$
$$U_c(halt) = U_c(s0)$$
$$\Delta U_c(continue) = U_c(s1') - U_c(s0)$$

Since state *s1'* is unknown, its real utility is uncertain, which we will model in Equation 3 by the expected utility *EUc*(*s1'*).

$$\Delta EU_c(continue) = EU_c(s1') - U_c(s0) \quad (2)$$

With the knowledge of state *s1*, we can rewrite this as follows:

$$\Delta EU_c(continue) = (EU_c(s1') - U_c(s1)) + (U_c(s1) - U_c(s0)) \quad (3)$$

Equation 4 estimates the utility change in two periods: from *s0* to *s1* and from *s1* to *s1'*. Using the additive separation assumption, *Uc* can be replaced by *Uo* and *Ui*. Since there is no update from *s0* to *s1*, the change of object-level utility $U_o(s1) - U_o(s0)$ is 0; the change of the inference-related utility is the function of the time spent for this period (x seconds as known), which can be expressed as $U_i(x)$. From *s1* to *s1'*, the computation is based on prediction. The expected solution cost is *ub1'* as mentioned, and the expected time from *s1* to *s1'* is *x'*. Thus the change of utility can be expressed as $EU_o(|ub1' - ub0|) + EU_i(x')$.

$$\Delta EU_c(continue) = EU_o(|ub1' - ub0|) + EU_i(x') + 0 + U_i(x) \quad (4)$$

Equation 5 can be expressed using probabilities and utilities:

$$\Delta EU_c(continue) = \sum_{ub1'} P(ub1') \times U_o(|ub1' - ub0|) + \sum_{x'} P(x') \times U_i(x') + U_i(x) \quad (5)$$

Equation 6 is the equation used in the meta-reasoner. If the estimated expected change in computing one more step is positive, solving will continue. We assume that the user preference model for costs due to time produces a single, global optimal *uc*. Otherwise, the meta-reasoner may halt the system when it detects a local maximum. Notice that *ui* only included the cost of solving; recall we require that the meta-reasoning costs be negligible. If the probabilities P(*ub'*) and P(*x'*), and the utilities *Uo*

are available cheaply during on-line computation, the meta-reasoning costs will be negligible. The probabilities will come from a simple statistical model (Section 2.3), and the utility functions will be based on different user models (Section 2.4).

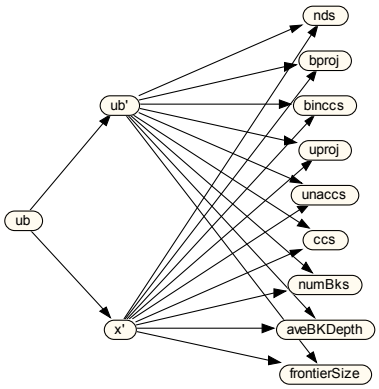

Fig. 3. The statistical model for predicting *ub0*' and *x*'.

2.3 A Statistical Model For Predicting Outcomes Of Computation

To obtain probabilities for *ub1*' and *x*' in Equation 6, we used a simple model based on a naïve Bayes classifier, extending it to 3 layers. The variables *ub1*' and *x*' are considered the classifications, and we used the following features which are easily available during the runtime of the solving mechanism:

1. *ub0* : the cost of the current best solution.
2. *nds* : the number of nodes in the search tree visited since last solution update.
3. *bproj* : the number of binary projections from last solution update.
4. *binccs* : the number of binary constraint checks since last solution update.
5. *uproj* : the number of unary projections since last solution update.
6. *unaccs* : the number of unary constraint checks since last solution update.
7. *ccs* : the number of checks since last solution update. This is a second level feature, which is the sum of features 3 to 6.
8. *numBks* : the number of backtracks since last solution update.
9. *aveBKDepth* : the average depth of search.
10. *frontierSize* : the length of the current unvisited node list for the search.

According to the naïve Bayes model, the features are assumed to be conditionally independent given the classifications. Our modified model puts *ub0*, the value of the current best assignment, as a parent to the classifications, acting as a kind of switch. For example, if the input of current solution cost *ub0* equals to 5, the probability of *nextub* being larger than 5 will be 0, because they can only be better than the current best solution. The model is shown in Figure 3.

To generate the training data, we solved randomly generated training problem instances using the same object level algorithm (BB-SRFL-H), reporting data at every point where an obvious change is observed. We tried three different strategies for

reporting runtime data. One alternative was to report at the point where a search node was visited; the second reported data when a backtrack occurred. Experiments showed that both of these two options produced too trivial information and very large data files, so we used a third option: whenever the solving algorithm found a solution which was better than the current best one, it reported the new solution cost, the time spent, the number of nodes, checks and all the input features mentioned in above.

The data are distributed over a wide range, and therefore were "discretized" into abstract states by visual inspection of the distribution of the data for each variable. This has two consequences. First, the summations in Equation 6 are feasible with discretized values, and second, the computed change in expected value is an approximation of the actual change.

The statistical model was constructed using the maximum a posterior hypothesis (MAP) learning rule, as is common in naïve Bayes models. Problem sets of 2, 5, 10, 20, 50, 100, 200, 400 and 800 COPs were used to generate training data and smaller numbers of testing data were used to test the models. Five statistical models were constructed from training data collected by solving COP instances. The average error rate for each set of 5 models was measured by counting correct predictions of *ub'* on a test set, as well as by computing the predicted error in the expectation of *ub'* for the test set. The error rates converged for trials greater than 200 COP instances. Specifically, the average prediction accuracy for the 200 instance models was 72% (standard deviation: 0.02), and the relative accuracy in the predicted *ub1'* for these models was 86% (st. dev. 0.004). Therefore, we used one of the five models constructed using 200 COP instances as the model to be used in our system.

2.4 User Preference Models

We have assumed that time is the main resource cost in this system. Future work can include other costs such as the memory cost. Focusing on the time cost, we introduce different user preference models in this section. Several classes of utility functions of *ui* (time) have been examined, including urgency, deadline, and urgent-deadline situations [Horvitz 1988]. Section 2.2 demonstrated meta-reasoning with a simple time model wherein 30 seconds costs $1. However, different users may have different requirements about urgency and deadlines. To measure how the time delay affects the solving and the decisions, utility functions are associated with time delay.

We focus on urgency models, rather than deadlines. Where there are pure deadlines, the utility function *Ui* has two stages: before the deadline, Ui=0, and after the deadline Ui=-∞. Thus, meta-reasoning is not even useful for pure deadlines.

An urgency model is a general class of utility functions in which the cost increases monotonically as the time delay increases. We focus on the urgency model to convert time to utilities for the computation of expected value (see Equation 6). If the system is trapped in a long proof without any solution improvement, the cost will increase significantly. Our urgency models are linear with time as examples only; our approach is not limited to linear models.

3. Experimental Results

This section reports on the experiments of testing our solver for several different user models. The performance of our meta-reasoner solver is compared with the results from the original non-meta-reasoning solver.

For complete generality, a meta-reasoning system would be able to solve many different kinds of COP instances. However, in this system we focus on a very specific class of COP instance: randomly generated Max-CSPs with 17 variables, 8 values, a constraint density of 0.5, and an average constraint tightness of 0.5. Our implementation is limited to problems from this class, but our design can be extended to any class; we are pursuing the open research issue of developing an approach that can be used for many classes of COP problems. Our experiments use very small COP instances, because of the need to solve them completely to analyze the results. The details of our experimentation follow.

3.1 Testing the Meta-reasoning Solver

We tested the meta-reasoner on 50 random problems from the same class as above, using the user model from Section 2.3: each 30 seconds delay costs the user $1. Figures 4, 5 and 6 are the graphs showing the comparison between using the meta-reasoning solver and the same solver without the meta-reasoning. The graphs show the result for each of the problems in terms of the solution costs and the time spent.

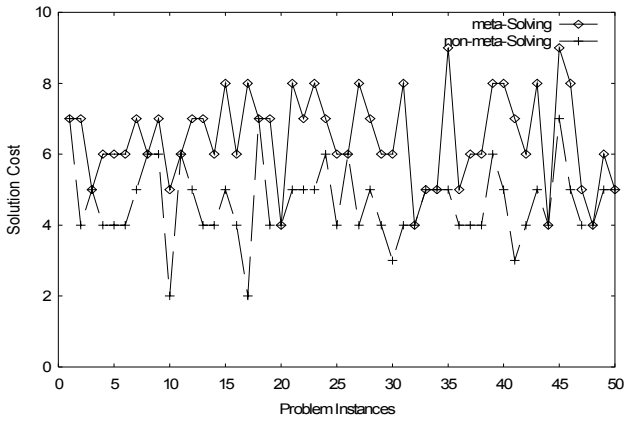

Fig. 4. Comparison on solution costs on 50 COPs. The average difference is 1.84 and the standard deviation is 1.40, in favour of the non-metareasoning solving.

In Figure 4, the x-axis shows the independent problem instances and the y-axis shows the solution costs from the two solvers. The solution cost for non-meta-solving is the cost of the solution at the end of the complete solving, and the solution cost for meta-solving is the cost of the current best solution at the point where the solver decided to stop. From Figure 4, we can see that using the meta-solver results in solutions that are about 2 constraint violations worse on average.

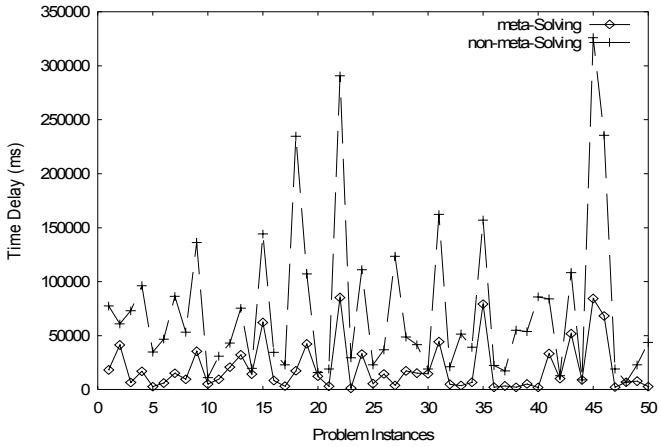

Fig. 5. Comparison on used time, on 50 problem instances as above. The average time difference is 53 seconds and the standard deviation is 55 seconds.

Figure 5 shows the comparison of the run time of the two methods. Here, the y-axis shows the amount of time used. The meta-solver almost always stops before the non-meta-solver does, and on average, about 53 seconds sooner. In a few cases, the two solvers require nearly exactly the same time.

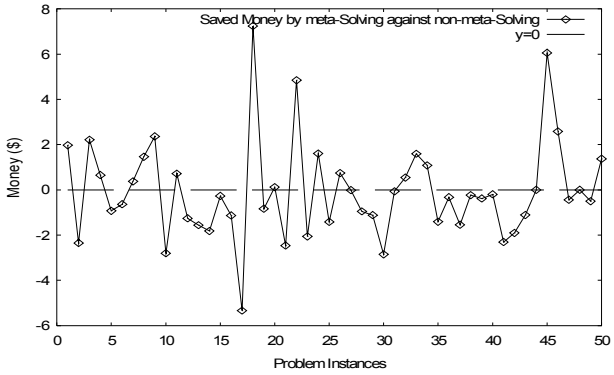

Fig. 6. Difference in comprehensive utility between the two solvers. The average difference in "dollars" is -0.05 and the standard deviation is 2.17, in favour of the non-metareasoning solver.

Figure 6 compares the comprehensive utility of the two solvers, using $1 per violation, and $1 per 30 seconds. The average *uc* is –$0.05, which is close to zero as one would expect, given that two violations equals one minute's computation.

3.2 Testing Different User Preference Models

To see how our meta-reasoning solver provides different results for different user models, 7 user models were tested with 50 COPs. These 7 models were just examples

that show the time cost from expensive to cheap: M1 (1s = $1), M2 (5s = $1), M3 (10s = $1), M4 (20s = $1), M5 (30s = $1), M6 (60s = $1), and M7 (120s = $1).

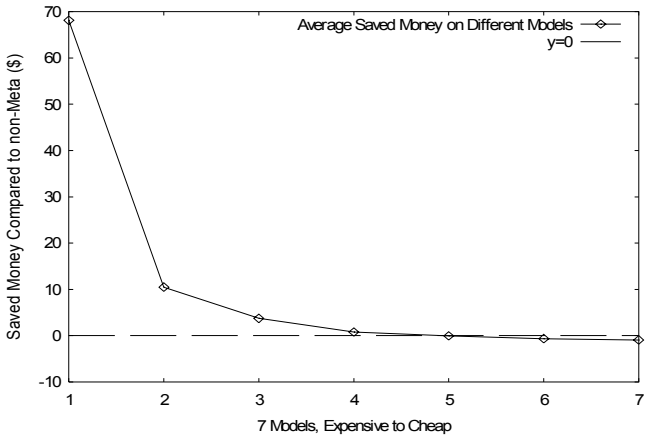

Fig. 7. Average difference in comprehensive utility using the metareasoning solving on 7 user preference models.

Figure 7 shows the average of the saving on these 7 models. The x-axis is the 7 models, from expensive to cheap, and the y-axis is the average saving for these 50 COP instances using these 7 models. This figure shows that our meta-reasoner pays off when time is expensive, but is comparable to the complete solver if time is cheap.

4. Summary and Future Work

We designed and implemented a decision theoretic meta-reasoning COP. The system is able to make run-time trade-offs based on a model of expected comprehensive utility. A traditional machine learning method was used to build a model of the way a COP solver improves solutions. Different user models were used to test the system's adaptability and its advantages or disadvantages according to different time urgency. Experiments suggest that this system is useful on expensive time models, but on average did not perform badly in cases when time was cheap.

Currently the meta-reasoner only provides the solver with the decision to halt or to continue, providing no other help during the solving. We are developing a meta-reasoner to predict the next *ub* value. This could be used as an expected upper bound, which could be used to limit search. This is similar to the binary choice meta-reasoner of Carlsson et al [1996]; however, we expect the *ub* from the meta-reasoner will be more accurate than the binary choice algorithm, and thus we should have fewer wrong predictions and fewer backtracks.

The design of the system is not limited to any specific variety of VCSP. However, the implementation that was tested is specific to a class of very small VCSP. This limitation was imposed by the need to test the solver on problems for which the best objective solution (ignoring computational costs) is feasible to compute. We are

currently working on testing the design on larger problems. To extend the approach to a wider class of problem, the features used to estimate expectations and probabilities need to be made independent of the problem class. We do not claim that the features presented in our model are optimal in any sense. A different set of relationships or features may improve the accuracy of the predictions.

The Bayesian model was designed so that inference in the model would be easy, but because it is based on the "naïve Bayes" assumption, the manual construction of the network leaves significant space for improvement. Learning a model structure from the data may improve the predictive power of the meta-reasoner, but the choice of model has to take on-line meta-reasoning costs into account.

Acknowledgements

The second author acknowledges support by NSERC through RGPIN2387870-01.

References

Boddy, M. and Dean, T. 1994. Decision-Theoretic Deliberation Scheduling for Problem Solving in Time-Constrained Environments. Artificial Intelligence, Volume 67, Number 2, pp 245-286, 1994.

Bistarelli, S.; Fargier, H.; Montanari, U.; Rossi, F.; Schiex, T.; and Verfaille, G. 1996. Semiring-based CSPs and valued CSPs: Basic properties. In M. Jampel, E. C. Freuder, and M. Maher, editors, *Over-Constrained Systems*, Volume 1106 of Lecture Notes in Computer Science, pp111-150. Springer, Berlin, 1996.

Carlsson, M.; Ottosson, G. 1996. Anytime Frequency Allocation with Soft Constraints. CP96 Pre-Conference Workshop on Applications. 1996

Dean, T.; Kaelbling, L.; Kirman, J.; Nicholson, A. 1995. Planning Under Time Constraints in Stochastic Domains. Artificial Intelligence, Volume 76, Number 1-2, Pages 35-74, 1995.

Horvitz, E. J. 1988. Reasoning under Varying and Uncertain Resource Constraints. In Proceedings of the National Conference on AI (AAAI-88), pp 111-116. 1988.

Horvitz, E. J. 1989. Reasoning about Beliefs and Actions under Computational Resource Constraints. In Uncertainty in Artificial Intelligence 3. Elsevier Science Publishers, 1989.

Horvitz, E. J.; Ruan, Y.; Gomes, C.; Kautz, H.; Selman, B. and Chickering, D. M. 2001. A Bayesian Approach to Tackling Hard Computational Problems. Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence, pp235-244, 2001.

Larrosa, J. 2002. Node and Arc Consistency in Weighted CSP. In Proceedings of the 18th National Conference on Artificial Intelligence (AAAI-2002), pp48-53 2002 .

Russell, S.; Wefald, E. 1991. The principles of meta-reasoning. 1st International Conference on Knowledge Representation and Reasoning, pp406-411. Morgan Kaufmann. 1991

Schiex, T. 2000. Arc consistency for soft constraints. In CP-2000, pp411-424, 2000.

Schiex, T.; Fargier, H.; and Verfaillie, G. 1995. Valued Constraint Satisfaction Problems: Hard and Easy Problems. In Proc. of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95), pp 631--637. 1995.

Verfaillie, G.; Lemâıtre, M.; and Schiex, T. 1996. Russian doll search. In AAAI-96, pp181–187, 1996.

Zheng, J. and Horsch, M. C. 2003. A Comparison of Consistency Propagation Algorithms in Constraint Optimization. In Proceedings of the Sixteenth Canadian Conference on Artificial Intelligence, pp160-174, 2003.