Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence

Selecting the Appropriate Consistency Algorithm for CSPs Using Machine Learning Classifiers

Daniel J. Geschwender, Shant Karakashian, Robert J. Woodward, Berthe Y. Choueiry, Stephen D. Scott
Computer Science and Engineering, University of Nebraska-Lincoln, USA
{dgeschwe|shantk|rwoodwar|choueiry|sscott}@cse.unl.edu

Abstract

Computing the minimal network of a Constraint Satisfaction Problem (CSP) is a useful and difficult task. Two algorithms, PerTuple and AllSol, were proposed to this end. The performances of these algorithms vary with the problem instance. We use Machine Learning techniques to build a classifier that predicts which of the two algorithms is likely to be more effective.

Introduction

Constraint Processing is a flexible paradigm for modeling and solving constrained combinatorial problems. A Constraint Satisfaction Problem (CSP) is defined by a set of variables, their respective domains, and a set of constraints over the variables. The constraints are relations, sets of tuples, over the domains of the variables, restricting the allowed combinations of values for variables. To solve a CSP, all variables must be assigned values from their respective domains such that all constraints are satisfied. A CSP can have one, several, or no solutions. Determining if a CSP has a solution is NP-complete in general. A significant portion of the research on CSPs is devoted to consistency properties and algorithms for enforcing them. *The long-term goal of our research is to design strategies for automatically choosing the right level of consistency to enforce in a given context and the appropriate algorithm for enforcing the chosen consistency level.* One such important consistency property is constraint minimality, which guarantees that every tuple in a constraint definition appears in a solution to the CSP (Montanari 1974). This property was shown to be important for knowledge compilation (Gottlob 2012) and achieving higher consistency levels (Karakashian, Woodward, and Choueiry 2013). Karakashian et al. proposed two algorithms, PerTuple and AllSol, for computing the minimal network (2012). The performances of the two algorithms widely vary depending on the problem instance. We propose to use a classifier to select the appropriate algorithm to use given a set of problem features (Xu et al. 2008). In this paper, we describe building classifiers using three different learning algorithms trained under different conditions and summarize our experimental results, establishing the usefulness of our approach.

Computing the Minimal Constraint Network

In a minimal CSP, every tuple must appear in a solution to the CSP. We consider two algorithms to this end: PerTuple and AllSol (Karakashian et al. 2010; 2012). PerTuple operates by iterating through every tuple of every relation to extend this tuple to a consistent solution using backtrack search. If a solution is found, all the tuples in the solution are kept; otherwise, the original tuple is removed. PerTuple terminates when every tuple has been checked. PerTuple may execute as many searches as there are tuples in the problem. In contrast, AllSol performs a single backtrack search, finding all solutions in the problem. For every solution found, all involved tuples are saved. AllSol terminates when the entire search tree has been explored. When solutions are plentiful, finding a first solution is easy and PerTuple quickly terminates. In contrast, AllSol gets bogged down enumerating all solutions. When solutions are sparse, both algorithms terminate quickly by pruning much of the search tree. However, around the phase transition (Cheeseman, Kanefsky, and Taylor 1991), there are many "almost" solutions and backtrack search is costly. Therefore, PerTuple's multiple searches put it at a disadvantage to AllSol's single search.

Building a Classifier

Our CSP instances were taken from benchmarks from the 2008 CSP Solver Competition. Each benchmark has a set of instances of similar structure. Each CSP from each benchmark is broken down into clusters using CSP tree decomposition. Each cluster is a subproblem that is treated as an independent CSP instance. This decomposition is performed because it corresponds to the intended use of the two algorithms (Karakashian, Woodward, and Choueiry 2013). We ran both AllSol and PerTuple on every subproblem and recorded their run times. For each CSP instance, we also measured the values of 12 features assessing various aspects of the instance pertinent to the task (Karakashian et al. 2012). Examples include: κ (predicts the phase transition (Gent et al. 1996)), relLinkage (measures how likely a tuple at the overlap of two relations is to appear in a solution), and relPerVar (measures constrainedness). We build the input to the learning algorithm from this data. We used three algorithms from the Weka machine learning suite:[1] J48, Multi-

[1] www.cs.waikato.ac.nz/ml/weka

layerPerceptron, and NaiveBayes, chosen for their diversity. We studied four setup strategies: *a*) considering the original dataset; *b*) ignoring instances where the runtime difference of AllSol and PerTuple is within 100 ms; *c*) weighing each instance in the dataset with the runtime difference; and *d*) using a cost matrix (Xu et al. 2011).

Experiments

Our data was collected from five sets of benchmark problems (aim50, aim100, aim200, modRenault, and warehouses) with a total of 3592 data instances. AllSol outperformed PerTuple on 1130 instances and PerTuple outperformed AllSol on 2462. The average AllSol execution time on all instances was 4928 ms with a standard deviation of 238499 ms. For PerTuple, the values were 699 ms and 1000 ms, respectively. Within the considered dataset, PerTuple is generally fast and consistent while AllSol's performance varied by several minutes between instances.

First, we ran each individual benchmark and the combined dataset through the three Weka algorithms to generate classifiers, using 10-fold cross validation. Classifiers were evaluated based on their weighted average F-measure. For the combined data set, J48 achieved a .726 F-measure, MultilayerPerceptron was .726, and NaiveBayes was .728. Results varied between individual benchmarks, from a .463 F-measure on the aim50 benchmark to .993 on warehouses.

We ran the second experiment on the combined dataset, see Table 1. In an attempt to improve the classifier, we ignored all instances where the runtimes of the two algorithms differed by less than 100 ms. The F-measure of J48 increased to .917. Next, instead of ignoring the instances with a time difference of less than 100 ms, we put those instances into a third class (see 3 classes in Table 1). Here, J48 reached .936 F-measure. However, this value is misleading because of the large size of this new class. Further, it is easy to classify properly but is almost meaningless as it will result in no significant time savings. The classes we care about are AllSol and PerTuple, for which the F-measure was .501.

Table 1: Weighted avg. F-measure of the three algorithms.

	J48	MP	NB
All instances	.726	.726	.728
$\delta_t \geq$100ms	**.917**	.880	.900
3 classes	.936	.941	.871
PerTuple+AllSol	.501	.547	.433

Ignoring the data instances with small runtime differences helped to emphasize the more important data instances. To further emphasize the most meaningful data instances, we considered weighted datasets (third experiment) and cost-sensitive modifications (fourth experiment) to the J48 classifier. In both experiments, we considered the complete datasets classified into two classes (AllSol and PerTuple). In the third experiment, each instance was given a weight equal to the difference in execution times of the algorithms. Therefore, an instance with a difference of 100000 ms is given proportionally more importance than an instance with a difference of 10 ms. In the fourth experiment, we created a cost matrix for cost-sensitive classification. A correct classification costs 0. On our data set, classifying an AllSol instance as a PerTuple instance yielded an average time loss of 59 ms, whereas the converse yielded 6196 ms.

Table 2: Performance of J48 on four strategies.

Strategy	F-measure	Time saved		Time lost
		%	ms	ms
All instances	.727	99.87%	15,301,950	19,350
$\delta_t \geq$100ms	.729	99.90%	15,306,510	14,790
Weighted	**.743**	**99.96%**	**15,314,980**	**6,320**
Cost	.557	99.57%	15,255,190	66,110

The resulting F-measures for J48 are shown in Table 2. The accuracy on the weighted set gives the percent of potential time savings that was actually obtained. While not achieving perfect F-measures, all four classifiers saved over 99% of the time possible to save. All the significant instances were properly classified, only the more trivial instances were incorrect. The classifier trained on the weighted dataset marginally achieved the best F-measure and time savings. The classifier trained with the cost matrix had the worst performance. Indeed, the cost matrix takes into account the average cost of a misclassification, however, the standard deviation of the execution time is so high that the average is not particularly relevant. We conclude that J48 with the weighted dataset seems to be the most promising.

Conclusion and Future Work

Using the wrong algorithm to compute the minimal network of a CSP can be a costly error. We use machine learning techniques to predict the 'better' algorithm to apply, and empirically show that our approach is feasible and promising. Our goal is to use the classifier dynamically during search to select the appropriate algorithm, and, beyond minimality, the most advantageous consistency property to enforce.

Acknowledgments: Supported by NSF Grant No. RI-111795. Experiments conducted at UNL's Holland Computing Center.

References

Cheeseman, P.; Kanefsky, B.; and Taylor, W. 1991. Where the Really Hard Problems Are. In *Proc. IJCAI 91*, 331–337.

Gent, I.; MacIntyre, E.; Prosser, P.; and Walsh, T. 1996. The Constrainedness of Search. In *Proc. of AAAI 96*, 246–252.

Gottlob, G. 2012. On Minimal Constraint Networks. *Artificial Intelligence* 191-192(0):42 – 60.

Karakashian, S.; Woodward, R.; Reeson, C.; Choueiry, B. Y.; and Bessiere, C. 2010. A First Practical Algorithm for High Levels of Relational Consistency. In *Proc. of AAAI 10*, 101–107.

Karakashian, S.; Woodward, R. J.; Choueiry, B. Y.; and Scott, S. D. 2012. Algorithms for the Minimal Network of a CSP and a Classifier for Choosing Between Them. TR-UNL-CSE-2012-0007.

Karakashian, S.; Woodward, R.; and Choueiry, B. Y. 2013. Improving the Performance of Consistency Algorithms by Localizing and Bolstering Propagation in a Tree Decomposition. In *Proc. of AAAI 2013*, 8 pages.

Montanari, U. 1974. Networks of Constraints: Fundamental Properties and Applications to Picture Processing. *Information sciences* 7:95–132.

Xu, L.; Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2008. Satzilla: portfolio-based algorithm selection for sat. *JAIR* 32(1):565–606.

Xu, L.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2011. Hydra-MIP: Automated Algorithm Configuration and Selection for Mixed Integer Programming. In *RCRA Workshop 2011*, 16–30.