1-1-2007

Conditional constraint satisfaction and configuration: A win-win proposition

Esther Gelle

Mihaela C. Sabin
University of New Hampshire, Manchester, mihaela.sabin@unh.edu

Conditional Constraint Satisfaction and Configuration – A Win-Win Proposition

Esther Gelle, Power Generation, ABB Switzerland Ltd, Bruggerstrasse 72, CH-5400 Baden, Switzerland
Mihaela Sabin, Department of Mathematics and Computer Science, Rivier College, 420 Main Street, Nashua, NH 03060

Submitted and accepted by IEEE Intelligent Systems (Revised Submission, November 2006)

Product configuration, like design, planning, or hardware test generation, is a real-world, dynamic task. It supports on-the-fly selection of parts to assemble a large assortment of possible product variants to satisfy functional specifications and customer choices. The conditional constraint-satisfaction formalism offers a natural way of representing these constraints, some of which model how parts selection changes during the problem-solving process. Conditional constraint-satisfaction problems (CondCSPs) extend standard CSPs with a special type of constraints that can capture changes at solving time based on predefined conditions.

Recently, researchers have been increasingly interested in modeling dynamic application problems as CondCSPs. Consequently, they've developed specialized algorithms that depart from standard CSP solving. However, we know little about these algorithms' relative performance because they've been experimentally analyzed separately using different test suites. We present a CondCSP solver that includes two algorithms for direct and reformulated problem-solving of CondCSPs. Our analysis uses randomly generated CondCSPs and shows new correlations between problem topology and algorithm performance. These results shed more light on the CondCSP problem class while expanding on the configuration problem's solution.

Why Conditional Constraint Satisfaction?

Sanjay Mittal and Brian Falkenhainer introduced the dynamic CSP formalism and its original application domain, product configuration.[1] Sabin and Eugene C. Freuder further developed this formalism and renamed it "conditional CSP" to point out that the particular dynamic behavior this formalism captures consists of conditionally selecting only those variables and constraints that are relevant to final solutions.[2] Figure 1 shows a configuration problem's general characteristics in a simple example for configuring an industrial mixer.

Required parts and their values:

- mixer's vessel type is either mixer or reactor
- vessel's volume is either small or large
- mixing process is either dispersion or blending

Optional parts and their values:

- cooler's type is either cool1 or cool2
- condenser's type is either cond1 or cond2

Configuration requirements of parts compatibility:

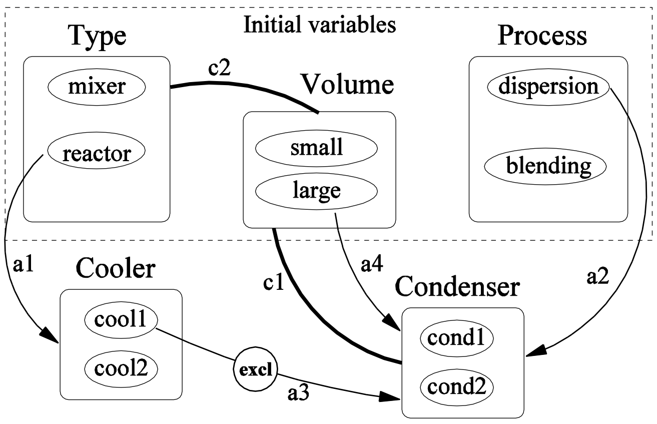
- small volume is incompatible with condenser's cond1
- mixer and reactor type is only compatible with small volume

Configuration requirements for selecting optional parts:

- reactor type includes the cooler option
- dispersion process includes the condenser option
- cool1 cooler excludes the condenser option
- large volume includes the condenser option

Figure 1 The characteristics of a configuration problem for an industrial mixer example.

In CondCSP terms (see figure 2), the mixer configuration problem has variables, corresponding to all the catalog parts, and constraints, corresponding to parts' compatibility rules and optional functional and user selections. A formal description of CondCSPs is available elsewhere.[3]



Initial Variables:
Type: {mixer,reactor}
Process: {dispersion,blending}
Volume: {small,large}

Optional Variables:
Cooler: {cool1,cool2}
Condenser: {cond1,cond2}

Compatibility Constraints:
c1: {(large,cond1)(large,cond2)
(small, cond2)}
c2: {(mixer,small)(reactor,small)}

Activity Constraints:
a1: Type=reactor →includes Cooler
a2: Process=dispersion →includes Condenser
a3: Cooler=cool1 →includes Condenser
a4: Volume= large →excludes Condenser

Solution Set:

Type	Process	Volume	Cooler	Condenser
reactor	dispersion	small	cool2	cond2
reactor	blending	small	cool1	EXCLUDED
reactor	blending	small	cool2	UNDETERMINED
mixer	dispersion	small	UNDETERMINED	cond2
mixer	blending	small	UNDETERMINED	UNDETERMINED

Figure 2. Variables and constraints of the industrial mixer example.

Recently, researchers have adapted constraint satisfaction and its conditional variant to the planning domain[4,5] and hardware test generation.[6] These application domains have produced either specialized algorithms for solving CondCSPs or reformulations of CondCSPs into standard CSPs using the classical CSP-solving arsenal. However, the CondCSP class lacks systematic findings with regard to relative algorithm performance and how that performance correlates with problem topology, such as density, satisfiability, and conditionality. CondCSP benchmark problems are almost nonexistent, compounding this challenge.

Our Contributions

To address these challenges, we've developed a CondCSP solver that includes two algorithms, which we initially introduced and evaluated separately. One algorithm uses direct solving methods that adapt standard consistency checking, such as forward checking and maintaining arc consistency, to the special constraints that enforce conditionality in a CondCSP.[7,8] The second algorithm reformulates the original problem into intermediate CondCSPs with incrementally lesser conditionality as they're ultimately transformed into standard CSPs. Standard consistency checking is interleaved with problem reformulation to eliminate inconsistent subproblems and solve the resulting standard CSPs.[9,10]

To overcome the lack of publicly available benchmark problems, we've used random CondCSPs and designed test suites for both direct- and reformulation-solving algorithms. Our initial work is available elsewhere.[3]

A more extensive evaluation analysis shows that no one method is best.[11] Reformulation solving in conjunction with forward checking has the advantage of pruning whole subproblems, so it copes better with larger problems (with solution spaces in the tens of thousands) than direct solving methods. Direct solving in conjunction with maintaining arc consistency is always preferable to direct solving using forward checking and is more efficient on large, overconstrained problems.

Solving Methods for CondCSPs

The large collection of thoroughly tested algorithms for solving standard CSPs contrasts starkly with the few existing algorithms for solving CondCSPs. Esther Gelle provides the first complete description of CondCSP backtrack search, which solves a partially reformulated CondCSP that rewrites exclusion activity constraints as compatibility constraints.[9] Sabin proposed CondCSP analogs to CSP backtrack (CondBt), forward checking (CondFc), and maintaining arc consistency (CondMac) search algorithms.[7] CondMac interleaves backtrack search with maintaining arc consistency, which is adapted to propagate consistency checking on both compatibility and activity constraints in the original CondCSP. Experimental evaluation on random CondCSPs shows up to two orders of magnitude improved efficiency over plain backtrack search.

A different approach is to successively process activation conditions of inclusion into an equivalent reformulation.[9,10] The reformulation algorithm, Gt, generates a tree whose internal nodes are CondCSPs and whose leaves are standard CSPs. Gt reformulates inclusion activity constraints into compatibility constraints by creating intermediate CondCSPs with lesser conditionality until it reaches the leaves level. Standard consistency checking is interleaved with tree generation to eliminate inconsistent subproblems and to eventually solve the resulting standard CSPs. The two algorithms implemented are Gt with forward checking (GtFc) and Gt with maintaining arc consistency (GtMac) algorithms. Experimental evaluation results of GtFc were reported for solving a bridge design problem [9]. **//by you? Do you have a reference?//**

To examine these algorithms' relative performance, we've integrated both implementations within the same solver framework (written in C++).

Experimental Evaluation

Our experimental analysis used randomly generated CondCSPs. The random CondCSP generator extends problem topology parameters, such as density and satisfiability of standard constraints, with similar parameters that describe problem conditionality. R.J. Wallace, currently at the Cork Constraint Computation Centre, University College Cork, Ireland, developed the random CondCSP generator in 1996 when he was affiliated with the University of New Hampshire, U.S.A. . We designed several test suites for which we varied the number of variables and the density and satisfiability of both standard and condition-based constraints. We wanted to run experiments for solving problems of a solution space that was large enough to expose measurable differences in execution time, but still manageable. Our evaluation's main goal was to compare performance of both types of algorithm encoded in the same C++ framework and executed on the same type of machine (1.86 GHz Intel Pentium M processor).

All test suites showed that all methods' time performance improves with increasing conditionality. With increasing conditionality the problem gets overconstrained and the number of solutions decreases dramatically ,so finding all solutions is easier.

In all cases, CondMac outperformed CondFc, confirming Gelle and Sabin's results.[3] CondMac exploits activity constraints and the tension between inclusion and exclusion activations to prune variable domains. Because the experimental studies use binary constraints, the activity constraints with unary conditions yield a good pruning effect.

GtFc and GtMac's relative time performance depends on the solution-space size. In general, GtFc solves underconstrained problems faster. Gt transforms all exclusion constraints into compatibility constraints before generating the tree. We know that Mac's performance deteriorates with increasing problem density. In the case of Gt, the number of different compatibility constraints of the resulting standard CSPs is larger than the original CondCSP's because all activity constraints have been reformulated into compatibility constraints. Overall, the implementation overhead overcomes Mac's pruning power, which doesn't always pay off. Consequently, GtFc prunes more efficiently at a lower effort cost.

GtFc becomes faster than CondMac when the solution space is large (that is, in an underconstrained problem), and, within the same parameter setting problem, GtFc runs faster when the solution space is small (that is, in an overconstrained problem).

References

1. S. Mittal and B. Falkenhainer, "Dynamic Constraint Satisfaction Problems," *Proc. 8th Nat'l Conf. Artificial Intelligence*, MIT Press, 1990, pp. 25–32.
2. M. Sabin and E.C. Freuder, "Detecting and Resolving Inconsistency and Redundancy in Conditional Constraint Satisfaction Problems," *Proc. Constraint Programming* (CP 98), Workshop on Constraint Problem Reformulation, 1998. http://ic-www.arc.nasa.gov/people/frank/workshop.html#pub
3. E. Gelle and M. Sabin, "Solving Methods for Conditional Constraint Satisfaction," *Proceedings of the IJCAI*Workshop on Configuration affiliated with the Eighteenth International Joint Conference on Artificial Intelligence, 2003; www2.ilog.com/ijcai-03/Papers/IJCAI03-02.pdf, pp 7-12
4. M. Binh Do and S. Kambhampati, "Solving Planning-Graph by Compiling it into CSP," *Proc. 5th Int'l Conf. Artificial Intelligence Planning Systems*, AAAI Press, 2000, pp. 82–91.
5. I. Miguel, P. Jarvis, and Q. Shen, "Flexible Graphplan," *Proc. 14th European Conf. Artificial Intelligence*, IOS Press, 2000, pp. 506–510.
6. F. Geller and M. Veksler, "Assumption-Based Pruning in Conditional CSP," *Proc. 11th Int'l Conf. Principles and Practice of Constraint Programming*, LNCS 3709, Springer, 2005, pp. 241–255.
7. M. Sabin, "Towards Improving Solving of Conditional Constraint Satisfaction Problems," doctoral dissertation, Computer Science Department, University of New Hampshire, 2003.
8. M. Sabin, E.C. Freuder, and R.J. Wallace, "Greater Efficiency for Conditional Constraint Satisfaction," *Proc. 9th Int'l Conf. Principles and Practice of Constraint Programming*, LNCS 2833, Springer, 2003, pp. 649–663.
9. E. Gelle, "On the Generation of Locally Consistent Solution Spaces," doctoral thesis, Artificial Intelligence Laboratory, Ecole Polytechnique Fédérale de Lausanne, 1998.
10. E. Gelle and B. Faltings, "Solving Mixed and Conditional Constraint Satisfaction Problems," *Constraints*, vol. 8, no. 2, 2003, pp. 107–141.
11. E. Gelle and M. Sabin, "Solver Framework for Conditional Constraint Satisfaction Problems," Papers from the Proceedings of the ECAI 2006 Workshop on Configuration affiliated with the 17th European Conference on Artificial Intelligence, 2006, pp 14-19

Esther Gelle is head of R&D power generation at Asea Brown Boveri Switzerland. Contact her at esther.gelle@ch.abb.com.

Mihaela Sabin is associate professor of computer science in the Department of Mathematics and Computer Science at Rivier College. Contact her at msabin@rivier.edu.