Generating Implied Constraints via Proof Planning

Alan M. Frisch, Ian Miguel and Toby Walsh

University of York, York, England. frisch,ianm,tw@cs.york.ac.uk

Abstract. Implied constraints are those that logically follow from the initial model of a constraint satisfaction problem. The addition of implied constraints can greatly improve efficiency and is an important step in solving difficult problems. Unfortunately, the addition of implied constraints is currently done by hand in an ad-hoc manner. We describe a new application for automated reasoning in general, and proof planning in particular: generating those implied constraints automatically which will help to reduce search. The utility of this approach is demonstrated on two small examples.

1 Introduction

Constraint satisfaction is a highly successful technology for tackling a wide variety of search problems including resource allocation, transportation and scheduling. Recent studies have shown that implied constraints added by hand to a problem representation can lead to significant reductions in search (e.g. [SSW99]). There has, however, been little research on how to generate such implied constraints automatically. Implied constraints are simply logical consequences of the initial problem specification. Therefore, we are attempting to use theorem proving techniques, in particular Bundy's "proof planning" approach [Bun91], to generate implied constraints automatically.

Once we have good methods for generating implied constraints, our next objective will be to develop heuristics for identifying which of the generated constraints to retain and which to discard. Such methods and their associated heuristics could then be integrated into the next generation of constraint satisfaction toolkits.

2 Proof Planning

Proof planning is a technique used for guiding the search for a proof in automated theorem proving. Common patterns in proofs are iden-

tified and encapsulated in *methods* which are made available to the planner. It appears to be one of the most promising approaches for dealing with the combinatorially explosive problem of generating useful logical consequences of an initial problem specification. Proof planning has often been associated with "rippling" [BSvH+93], a powerful heuristic for guiding search in inductive proof. However, proof planning can easily be adapted to other mathematical tasks like finding closed form sums to series [WNB92] or, as here, generating logical consequences which may make useful implied constraints. We will identify common patterns in generating implied constraints to create appropriate methods for our application.

Proof planning offers several potential advantages over other theorem proving techniques for this task. First, methods can be given very strong preconditions which limit the generation of logical consequences to those that are likely to make useful implied constraints. Second, methods can act at a very high level. For example, we can perform complex rewriting, simplifications, and transformations. Such steps might require very long and complex proofs to justify at the level of individual inference rules. And third, the search control in proof planning is cleanly separated from the inference steps. We can therefore easily try out a variety of different search strategies like best-first search or limited discrepancy search.

3 The Fractions Puzzle

Consider the following example, taken from the Oz finite domain constraint programming tutorial[1], in order to demonstrate some example methods. The constraint satisfaction problem below has 9 distinct non-zero digits, A to I, which satisfy the constraint:

$$\frac{A}{BC} + \frac{D}{EF} + \frac{G}{HI} = 1 \tag{1}$$

Note that BC is shorthand for $10 * B + C$, EF for $10 * E + F$ and HI for $10 * H + I$.

Our first method, `symmetry` removes a symmetry (in this case, that the order of the three fractions can be permuted) from the

[1] `http://www.mozart-oz.org/documentation/fdt/node31.html#section.propagators.fractions`

problem by adding the constraints:

$$\frac{A}{BC} \leq \frac{D}{EF} \tag{2}$$

$$\frac{D}{EF} \leq \frac{G}{HI} \tag{3}$$

Such symmetry breaking constraints are not logical consequences of the initial problem specification but appear to be very useful for generating implied constraints. Note that the `symmetry` method preserves satisfiability: all other solutions can be generated by permutation.

Our next method, `eliminate`, is a generalisation of Gaussian elimination. Its preconditions identify a non-linear constraint and find a sub-term to eliminate. `Eliminate` can be used to simplify equation (1) by eliminating first $\frac{D}{EF}$ and then $\frac{G}{HI}$ in favour of $\frac{A}{BC}$ as follows:

$$\frac{A}{BC} + \frac{A}{BC} + \frac{G}{HI} \leq 1 \text{ using (2)} \tag{4}$$

$$\frac{A}{BC} + \frac{A}{BC} + \frac{A}{BC} \leq 1 \text{ by transitivity, using (2), (3)} \tag{5}$$

In many examples, we have found that we may have to do some algebraic manipulations in order to eliminate some sub-term. Therefore, we intend to adapt methods from PRESS [BW81] like `isolate`, `collect` and `attract` to perform algebraic manipulations. In this case, `collect` can be used to simplify equation (5), providing a ternary constraint:

$$\frac{3A}{BC} \leq 1 \tag{6}$$

Our next method, `linearise`, converts this into a linear form that will be useful to constraint solvers, giving the implied constraint:

$$3A \leq 10B + C \tag{7}$$

Through a similar process, $\frac{A}{BC}$ and $\frac{D}{EF}$ can be eliminated in favour of $\frac{G}{HI}$. The `eliminate` and `linearise` methods then give the implied constraint:

$$3G \geq 10H + I \tag{8}$$

Both of these implied constraints can help reduce search. Note that existing constraint solvers like ECLIPSE or CLP(FD) will simply delay a non-linear constraint like (1) until *all* 9 variables have been instantiated. They therefore do no better than generate and test. By comparison, our methods have identified two implied ternary constraints with which bounds consistency will start pruning values before we instantiate a *single* variable.

We also plan to develop some special purpose methods targeted at specialised constraints that occur frequently in constraint satisfaction problems. For example, the Fractions puzzle also has the constraint that that all the variables take different values. An `all-different` method will then deduce that $10H + I \geq 12$ since H and I must take different values. This is a tighter lower bound than could be derived by bounds consistency alone (which will give a bound of 11 for this sum). Using this tighter bound, we get that $3G \geq 12$. That is, $G \geq 4$.

In general, the `all-different` method identifies a sub-term consisting of variables that must take distinct values. Bounds for the sub-term can be found by examining the (all-different) bounds of the variables involved, subject to the constraint represented by the sub-term.

4 Professor Smart's Safe

This example problem is also taken from the Oz finite domain constraint programming tutorial[2]. In this example, the `all-different` method is used as a precursor to the `eliminate` method, whereas in the Fractions puzzle the opposite order was used. This demonstrates the need for the flexible search strategy enabled by the use of proof planning, as opposed to a more simplistic waterfall method such as that employed by PRESS.

The code of Professor Smart's safe is a sequence of 9 non-zero digits $x_1, ..., x_9$ such that the following constraints are satisfied:

$$x_4 - x_6 = x_7 \tag{9}$$

$$x_1 x_2 x_3 = x_8 + x_9 \tag{10}$$

$$x_2 + x_3 + x_6 < x_8 \tag{11}$$

[2] `http://www.mozart-oz.org/documentation/fdt/node19.html#section.problem.safe`

$$x_9 < x_8 \tag{12}$$
$$x_i \neq i \tag{13}$$
$$\text{all-different}(x_1, ..., x_9) \tag{14}$$

The `all-different` method can be used effectively on a sub-term of equation (11):

$$x_2 + x_3 + x_6 \geq 6 \tag{15}$$

As part of this process, `eliminate` can also be called as a sub-method on equation (11) to establish a lower bound for x_8:

$$6 < x_8$$

Node consistency and $x_i \neq i$ (equation 13) can then reduce the domain of x_8 to just two elements: $\{7, 9\}$.

Bounds consistency can make use of the upper bound of x_8, again in conjunction with equation (11):

$$x_2 + x_3 + x_6 < 9 \tag{16}$$

The `all-different` method can now establish upper bounds on the variables in equation (16):

$$x_2 \leq 5$$
$$x_3 \leq 5$$
$$x_6 \leq 5$$

This follows from the fact that if any digit takes the value 6 or greater, the smallest values that the remaining two can take are 1 and 2 (by all-different), and the sum of the three digits is not less than 9. Again using node consistency and equation (13), the domains of x_2 and x_3 can be reduced to $\{1, 3, 4, 5\}$ and $\{1, 2, 4, 5\}$ respectively.

5 Related Work

Proof planning has been implemented in a number of systems. The CLAM proof planner developed in Edinburgh controls search in the

Oyster proof checker [BvHHS90]. CLAM has also been linked to the HOL theorem prover [BSBG98]. Whilst much of the development of CLAM has been for inductive proof [BSvH+93], several other domains have been explored including finding closed form sums to series [WNB92]. Prior to CLAM, the PRESS system used a meta-level representation of proof methods to solve algebraic equations [BW81]. Despite the lack of a planner to put its methods together, PRESS was competitive with computer algebra systems of its era. The Ω system developed in Saarbrücken also implements proof planning, but in this case for a a higher order natural deduction style logic [HKK+94]. Recently, the third author has built a simple proof planning shell, CLAM-Lite, on top of the Maple computer algebra system [Wal00]. This system allows us to explore how proof and computation can be mixed together. It can, for example, find a closed form sum to a series, and prove by induction that the answer is correct.

As exemplified by [SSW99], several recent studies show that implied constraints added by hand to a problem representation can lead to significant reductions in search. However, outside a highly focused domain like planning (see, for example, [EMW97]), there has been little research on how to generate such implied constraints automatically. One exception is [Fri99], which generalises resolution to multi-valued clauses in which variables can take more than just the two values *True* and *False*, and proves that implied constraints generated by the closure of this operation will eliminate search.

6 Future Work and Conclusions

We are in the process of implementing these methods and trying them out on a catalogue of examples. We expect to identify and implement a number of other methods in addition to those described here. We can often derive some useful constraints by summing a number of variables, or by introducing new variables equal to algebraic expressions that occur in multiple constraints. This tightens the constraint graph, a representation of the structure of a constraint satisfaction problem, increasing the amount of constraint propagation. For example, introducing a new variable (constrained via a summation of existing variables) to represent the distance between

two non-adjacent marks on a circular Golomb ruler produces a significant reduction in search [SSW2000].

The practical application of constraint satisfaction techniques is a multi-million dollar industry. Modelling is a major bottleneck in the uptake of these techniques. In particular, adding implied constraints is a key difficulty and is currently done by hand. This paper has identified an approach by which automated reasoning can effectively enhance existing constraint solvers by discovering useful implied constraints.

Acknowledgements

This project and the second author are supported by EPSRC Grant GR/N16129[3]. The third author is supported by an EPSRC advanced research fellowship. The authors wish to thank Brahim Hnich for discussions about generating implied constraints automatically. Julian Richardson's adaption of PRESS to manipulate inequalities and its success at generating a number of implied constraints automatically helped inspire the search for the methods presented here.

References

[BSBG98] R. Boulton, K. Slind, A. Bundy, and M. Gordon. An interface between Clam and HOL. In *Proceedings of the 11th International Conference on Theorem Proving in Higher Order Logics*. Springer Verlag, Lecture Notes in Computer Science, 1998.

[BSvH+93] A. Bundy, A. Stevens, F. van Harmelen, A. Ireland, and A. Smaill. Rippling: A heuristic for guiding inductive proofs. *Artificial Intelligence*, 62:185–253, 1993. Also available from Edinburgh as DAI Research Paper No. 567.

[Bun91] A. Bundy. A science of reasoning. In J-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 178–198. MIT Press, 1991. Also available from Edinburgh as DAI Research Paper 445.

[BvHHS90] A. Bundy, F. van Harmelen, C. Horn, and A. Smaill. The Oyster-Clam system. In M.E. Stickel, editor, *10th International Conference on Automated Deduction*, pages 647–648. Springer-Verlag, 1990. Lecture Notes in Artificial Intelligence No. 449.

[BW81] A. Bundy and B. Welham. Using meta-level inference for selective application of multiple rewrite rules in algebraic manipulation. *Artificial Intelligence*, 16(2):189–212, 1981. Also available as DAI Research Paper 121, Dept. Artificial Intelligence, Edinburgh.

[3] `http://www.cs.york.ac.uk/aig/projects/implied/index.html`

[EMW97] M. Ernst, T.D. Millstein, and D.S. Weld. Automatic SAT-compilation of Planning Problems. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1169–1177, 1987.

[Fri99] A.M. Frisch. Solving Constraint Satisfaction Problems with NB-resolution. *Electronic Transactions in Artificial Intelligence*, 3, Section B:105–120, 1999. Available from http://www.ep.liu.se/ej/etai/1999/006/

[HKK+94] Xiaorong Huang, Manfred Kerber, Michael Kohlhase, Erica Melis, Dan Nesmith, Jörn Richts, and Jörg Siekmann. Ω-MKRP: A proof development environment. In Alan Bundy, editor, *Automated Deduction — CADE-12*, Proceedings of the 12th International Conference on Automated Deduction, pages 788–792, Nancy, France, 1994. Springer-Verlag, Berlin, Germany. LNAI 814.

[SSW99] B.M. Smith, K. Stergiou, and T. Walsh. Modelling the Golomb ruler problem. In *Proceedings of the IJCAI-99 Workshop on Non-Binary Constraints*. International Joint Conference on Artificial Intelligence, 1999. Also available as APES report, APES-11-1999 from http://apes.cs.strath.ac.uk/reports/apes-11-1999.ps.gz.

[SSW2000] B.M. Smith, K. Stergiou, and T. Walsh. Using Auxiliary Variables and Implied Constraints to Model Non-binary Problems. In *Proceeding of the 17th National Conference on Artificial Intelligence*, pages 182-187, 2000.

[Wal00] T. Walsh. Proof planning in Maple. In *Proceedings of the Workshop on Automated Deduction in the Context of Mathematics*. 17th Conference on Automated Deduction, 2000.

[WNB92] T. Walsh, A. Nunes, and A. Bundy. The use of proof plans to sum series. In D. Kapur, editor, *11th Conference on Automated Deduction*, pages 325–339. Springer Verlag, 1992. Lecture Notes in Computer Science No. 607. Also available from Edinburgh as DAI Research Paper 563.