Constraint Programming and Graph Algorithms

Kurt Mehlhorn

Max-Planck-Institut für Informatik
Im Stadtwald
66123 Saarbrücken
Germany
http://www.mpi-sb.mpg.de/~mehlhorn

In the spring of '99 my colleague Gert Smolka gave me a short introduction to constraint programming. During our discussion Gert emphasized that the search for efficient propagation algorithms leads to hard and well motivated questions in algorithmics. He pointed me to the papers [Reg94] by J.-C. Regin on "A filtering algorithm for constraints of difference in CSPs" and [GC97,GC00] by N. B. Guernalec and A. Colmerauer on "Narrowing a 2n-block of sortings in $O(n \log n)$". I soon learned that Gert had pointed me to an extremely rich source of algorithmic problems which I am now exploring in cooperation with E. Althaus and S. Thiel from my research group, D. Duchier and J. Niehren from the programming systems lab, A. Koller from the computer linguistics department at the Universität des Saarlandes, and with Nicolas Beldiceanu at SICS. Some papers [MT00,ADK+00,KMN00] and implementations of propagators for the Oz system (http://www.ps.uni-sb.de/oz2/) have come out of the cooperation so far.

The purpose of this talk is to get more researchers from the algorithms community interested in the subject. I want to make the case that constraint programming offers a lot of very challenging algorithmic problems and that cooperations between the constraint programming and the algorithms community could be very beneficial to both communities.

1 Constraint Programming

I give a brief account of constraint programming based on [Smo96]. For more thorough discussions we refer the reader to [Smo96,MS98]. Examples of constraint programming systems are ILOG, Chip, Eclipse, and Oz.

Constraint programming is a powerful programming paradigm, that allows one to formulate computational problems at a very high level. A computational problem is formulated as a *constraint*. A constraint is simply a conjunction of formulae of first-order logic. A solution is an assignment of values to the (free) variables which satisfies the constraint. It is the task of the constraint programming system to find a (all, a best) solution. The programmer only needs to specify the problem[1]. Readers entirely unfamiliar with constraint programming should move on to the next section and have a look at the specification of the N-queens problem given there.

[1] In most constraint programming systems, the programmer can also specify the search strategy, but this is not important at the current level of discussion.

Two types of constraints are distinguished: *basic* and *non-basic* constraints. This distinction is more a pragmatic distinction than a mathematical distinction. Basic constraints are constraints that can be handled efficiently and non-basic constraints are difficult constraints.

The basic constraints must have two properties:

- There is an efficient procedure which, given a collection of basic constraints, decides their satisfiability, or exhibits a solution, or exhibits all solutions.
- Fixing the value of a variable turns a set of basic constraints into a set of basic constraints.

We come to non-basic constraints. Non-basic constraints are difficult. They are simplified by means of iterated branching and propagation steps. A branching step splits the current state into two: the first is obtained by adding some constraint and the second is obtained by adding its negation. Branching generates a tree of states. Propagation operates on the current leaves of this tree and tries to simplify the states associated with them.

A *propagator* is associated with every non-basic constraint C. Let B denote the conjunction of all basic constraints. The invocation of a propagator has one of the following effects.

- The propagator may declare C obsolete. If C is declared obsolete, B must entail C, i.e., any assignment satisfying B must also satisfy C.
- The propagator may declare inconsistency. In this case $B \wedge C$ must be unsatisfiable.
- The propagator *advances to* D. Here D is a basic constraint that is entailed by $B \wedge C$ and that is strictly stronger than B, i.e., D is not entailed by B.
- The propagator cannot make any progress[2].

The computation in a leaf can stop when there is no further propagator or when a propagator declares inconsistency. In the former situation B defines the set of solutions. In the latter case, the leaf contributes no solution.

When the computation has not stopped yet and no propagator can make any progress, a distribution step (branching step) is taken. We invent a constraint C and proceed from the unsolved space to two new spaces, the first obtained by adding a propagator for C and the second obtained by adding a propagator for the negation $\neg C$.

2 An Example: The Alldiff Constraint

We use the alldiff constraint to illustrate the connection between propagation and graph algorithms.

Consider the following situation. We have a set of variables $x_1, \ldots, x_n$. The basic constraints define for each variable x_i a finite set V_i of conceivable values. We also want

[2] Propagators must have some minimal capabilities. In particular, propagators must make progress if B determines a single assignment. In this case a propagator for C must either declare C obsolete or must declare inconsistency.

the values of the variables to be pairwise distinct (alldiff constraint). The well known N-queens problem is a toy example which shows the usefulness of the alldiff constraint.

The goal is to place n queens on a $n \times n$ checker board. There can be at most one queen in each row, column, diagonal, and anti-diagonal. Assume that we place the i-th queen in position (i, x_i) (this definition makes sense since we have exactly one queen per row), $1 \leq i \leq n$. Every queen excludes a diagonal (45° line) and a anti-diagonal (-45° line) If we identify a diagonal and anti-diagonal with the square which it uses in row 0, then a queen in position (i, x_i) uses the diagonal $y_i = i + x_i$ and the anti-diagonal $z_i = i - x_i$. Thus we have the constraints:

$$\begin{gathered} x_i \in [1 \,..\, n], \quad y_i \in [2 \,..\, 2n], \quad z_i \in [-n+1 \,..\, n-1], \\ y_i = x_i + i, \quad z_i = i - x_i, \\ \mathit{Alldifferent}(x_1, \ldots, x_n), \quad \mathit{Alldifferent}(y_1, \ldots, y_n), \quad \mathit{Alldifferent}(z_1, \ldots, z_n) \end{gathered}$$

The first line defines the conceivable values for our variables, the second line defines relations between pairs of variables, and the third line defines three all-different constraints. Propagators for the equalities in the second line are easily designed. If a value v becomes impossible for x_i, the value $v + i$ becomes impossible for y_i, and vice versa.

Regin [Reg94] observed that matching theory leads to efficient propagators for the alldiff constraint. He suggested to view an alldiff constraint as a matching problem in a bipartite graph. On the left side of the bipartite graph there is a node for each variable and on the right side of the graph there is a node for each conceivable value. A variable x is connected to a value v if v is a possible for x.

An alldiff constraint is satisfiable iff the graph G just defined has a matching in which all variables are matched (a variable-perfect matching). This is a well studied problem in algorithmics. There is a solution to the alldiff constraint in which variable x has value v iff there is variable-perfect matching containing the edge (x, v). This is also a question which has been discussed in the literature on matching. Puget gave the following answer.

Let M be any matching involving all variables. We orient G as follows: Direct all edges in M from right to left and all edges not in M from left to right. An edge $(x, v) \notin M$ is part of a perfect matching if there is an alternating cycle containing it or an alternating path starting in a free node on the right side and ending in a matched node on the right hand side. In the directed version of G, the first kind of edge is an edge in any strongly connected component and the second kind of edge is any edge that is reachable from a free node on the right hand side. Thus given a matching, narrowing takes linear time. If no matching is available, narrowing takes time $O(\sqrt{n}m)$. Isn't this a nice application of matching theory?

In the two preceding sentences we talked about narrowing instead of propagation because propagation for the alldiff constraint amounts to narrowing the sets of potential values of the variables.

We can go further. Suppose we have just narrowed an alldiff constraint and then perform branching, e.g., we may partition the set of values of some variable into two. This will define two new bipartite graphs. Of course, we would like to work on these graphs

without explicitly constructing them. That's a typical problem in incremental graph algorithms.

Frequently, the conceivable values of a variable form an interval of natural numbers. The bipartite graph can then be represented in space $O(n)$ and we may be interested in narrowing algorithms which reduce the sizes of these intervals. The question arises whether this can be done without constructing the bipartite graph defined above. Puget [Pug98] answered this question positively and described an $O(n \log n)$ algorithm for obtaining bound consistency. The algorithm is again based on matching theory. In graph-theoretic terms the question amounts to computing perfect matchings and strongly components of so-called convex bipartite graph. In a convex bipartite graph, the nodes on the right side are linearly ordered and the neighbors of each node on the left side form an interval in the right side. The matching problem in bipartite convex graphs was first studied by [Glo67]. In [MT00] S. Thiel and I exploit the connection and obtain simplified and faster narrowing algorithms for the alldiff and the sortedness constraint.

3 Further Reading

In the previous section we tried to make the point that propagation rises interesting algorithmic questions. How can you learn more about these problems?

- The most effective method is probably to talk to a researcher in the constraint programming community.
- Read [VHS+97] on strategic directions in constraint programming.
- Check the proceedings of the conference on "Principles and Practice of Constraint Programming (CP)" [MP98,Jaf99].
- Check the journal "Constraints" (Kluwer Academic Publishers).
- Read the paper [Bel00] by Nicolas Beldiceanu. The paper classifies constraints using graph terminology.

References

[ADK+00] E. Althaus, D. Duchier, A. Koller, K. Mehlhorn, J. Niehren, and S. Thiel. Efficient algorithms for dominance problems. www.mpi-sb.mpg.de/ mehlhorn/ftp/dominance.ps, March 2000.

[Bel00] N. Beldiceanu. Global constraints a graph properties on structured networks of elementary constraints of the same type. Technical Report SICS T2000/01, SICS, Uppsala, Sweden, 2000.

[GC97] N. B. Guernalec and A. Colmerauer. Narrowing a 2n-block of sortings in O(n log n). *Lecture Notes in Computer Science*, 1330:2–16, 1997.

[GC00] N. B. Guernalec and A. Colmerauer. Optimal narrowing a block of sortings in optimal time. *Constraints*, 5:85–118, 2000.

[Glo67] F. Glover. Maximum matchings in a convex bipartite graph. *Naval Res. Logist. Quart.*, 14:313–316, 1967.

[Jaf99] Joxan Jaffar, editor. *Proceedings of the Fifth International Conference on Principles and Practice of Constraint Programming*, volume 1713 of *Lecture Notes in Computer Science*. Alexandra, VA, USA, October 1999.

[KMN00] A. Koller, K. Mehlhorn, and J. Niehren. A polynomial-time fragment of dominance constraints. www.coli.uni-sb.de/ koller/papers/poly-dom.ps.gz, April 2000.

[MP98] Michael Maher and Jean-Franois Puget, editors. *Proceedings of the Forth International Conference on Principles and Practice of Constraint Programming*, volume 1520 of *Lecture Notes in Computer Science*. Pisa, Italy, October 1998.

[MS98] Kim Marriott and Peter J. Stuckey. *Programming with Constraints: An Introduction*. MIT Press, Cambridge, MA, 1998.

[MT00] K. Mehlhorn and Sven Thiel. Faster algorithms for bound-consistency of the sortedness and the alldifferent constraint. www.mpi-sb.mpg.de/ sthiel/sortedness_paper.ps.gz, March 2000.

[Pug98] Jean-François Puget. A fast algorithm for the bound consistency of alldiff constraints. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98) and of the 10th Conference on Innovative Applications of Artificial Intelligence (IAAI-98)*, pages 359–366, Menlo Park, July 26–30 1998. AAAI Press.

[Reg94] J.-C. Regin. A filtering algorithm for constraints of difference in CSPs. In *Proc. 12th Conf. American Assoc. Artificial Intelligence*, volume 1, pages 362–367. Amer. Assoc. Artificial Intelligence, 1994.

[Smo96] Gert Smolka. Problem solving with constraints and programming. *ACM Computing Surveys*, 28(4es), December 1996. Electronic Section.

[VHS+97] Pascal Van Hentenryck, Vijay Saraswat, et al. Strategic directions in constraint programming. *Computing Surveys*, 28(4):701–726, December 1997. ACM 50th Anniversary Issue. Strategic Directions in Computing Research.