Randomized Filtering Algorithms

Irit Katriel and Pascal Van Hentenryck

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-06-09
June 2006

Randomized Filtering Algorithms*

Irit Katriel[1]** and Pascal Van Hentenryck[2]

[1] BRICS***, University of Aarhus, Århus, Denmark.
irit@daimi.au.dk
[2] Brown University, Box 1910, Providence, RI 02912
pvh@cs.brown.edu

Abstract. Filtering every global constraint of a CSP to arc consistency at every search step can be costly and solvers often compromise on either the level of consistency or the frequency at which arc consistency is enforced.
In this paper we propose two randomized filtering schemes for dense instances of *AllDifferent* and its generalization, the *Global Cardinality Constraint*. The first *delayed filtering* scheme is a Monte Carlo algorithm: its running time is superior, in the worst case, to that of enforcing arc consistency after every domain event, while its filtering effectiveness is analyzed in the expected sense. The second scheme is a Las Vegas algorithm using *filtering triggers*: Its effectiveness is the same as enforcing arc consistency after every domain event, while its running time improves in the expected sense.

1 Introduction

At the heart of the propagation-search technique in constraint programming is a trade-off between the amount of resources (space, time) spent on propagation and the amount of pruning that it achieves. One type of propagation is filtering, i.e., the task of identifying useless values in variable domains and removing them. Global constraints play an important role in this game by enabling a larger number of useless values to be identified compared to the decomposition into a semantically equivalent set of simpler constraints. A "good" global constraint is a constraint that allows a significant increase in the amount of filtering with a low computational overhead.

An important question is that of *incremental* filtering, i.e., the development of fast algorithms that re-filter a constraint after a small change has occurred, such as the removal of a value from a variable domain due to filtering with respect to another constraint. For *AllDifferent* and its generalization, the *Global Cardinality Constraint* (*GCC*), incremental arc consistency can be performed faster than computing arc consistency from scratch. The arc consistency algorithm first finds a solution, i.e., a flow in the variable-value graph of the constraint (which is augmented into a flow network by adding a source and a sink), and then computes the strongly connected components

* A preliminary version appeared in the proceedings of CP'AI'OR 2006 under the title *Expected-Case Analysis for Delayed Fitlering.*

** Supported by the Danish Research Agency (grant # 272-05-0081).
*** Basic Research in Computer Science, funded by the Danish National Research Foundation.

(SCCs) of the residual graph w.r.t. this flow [5, 6]. Let n and m be, respectively, the number of nodes and edges in the variable-value graph. Then the running time is dominated by the time to find a single solution. Hopcroft and Karp's algorithm [1] for maximum bipartite matching in $O(m\sqrt{n})$ time applies directly to *AllDifferent*, and as recently shown by Quimper et al. [4], can also be used for *GCC*.

An incremental arc consistency algorithm for *AllDifferent* and *GCC* receives an arc consistent flow network, a flow in it, and an edge that should be removed. If the removed edge is a flow edge, a new flow can be constructed after finding an augmenting path between the edge's endpoints, which takes linear time. Recomputing SCCs also takes time which is linear in the size of the graph. Although linear-time is much better than $O(m\sqrt{n})$, it is still a pity when it is wasted, i.e., when the algorithm does not discover any inconsistent edges.

We say that an edge in an arc consistent constraint's flow network is *important* if its removal would render at least one other edge "inconsistent". It is an interesting question whether there are fast methods to determine whether a given edge is important, or even to estimate *how* important it is, i.e., how many edges become inconsistent once it is removed?

In this paper we address a different aspect of this issue. We show that the number of important edges cannot be very large for the flow-based constraints mentioned above. More precisely, it is at most linear in the number of nodes in the variable-value graph. Furthermore, we show that it is possible, in linear time, to identify a set of this magnitude that contains all important edges (and perhaps other edges as well), even if edges outside of this set are removed from the graph. Using these results, we derive two randomized filtering schemes for dense instances of *AllDifferent* and *GCC*, i.e., instances in which the number of edges in the variable-value graph (that is, the sum of the cardinalities of the variable domains) is large compared to the number of variables.

1. The first *delayed filtering* scheme is a Monte Carlo algorithm. Assume that filtering is random, i.e., the probability of removing an edge from the variable-value graph is uniformly distributed. Since most edges are not important, most domain events do not remove important edges and therefore do not require re-filtering. The method computes a filtering frequency that reduces the amount of computation. Moreover, important domain events are propagated early in the expected sense.
2. The second *filtering triggers* scheme is a Las Vegas algorithm. Whenever we filter the constraint, we also identify a small subset S of the edges which has the following property: As long as we do not remove any edge of S from the variable-value graph, it is guaranteed that no filtering is necessary. It is only when one of these edges is filtered that a re-filtering is triggered. Since the set of triggers can be identified in linear time, the asymptotic running time of the Las Vegas algorithm is the same as that of recomputing arc consistency after every domain event. However, the expected time spent on filtering is reduced by a factor of m/n.

2 The Important Edges Of A Flow Network

The constraints we address can be modeled by a flow network such that there is a one-to-one correspondence between the integral feasible flows in the network and the solutions

to the constraint. Computing arc consistency then amounts to identifying edges that do not belong to any integral feasible flow, and flow theory tells us that given a flow f, the inconsistent edges are exactly the edges that do not belong to f and which connect two distinct SCCs in the residual graph corresponding to f [5, 6].

In the following we abuse terminology and speak of an edge in a flow network as being consistent if it belongs to at least one integral feasible flow. Furthermore, we say that the network is arc consistent if all of its edges are consistent and that an edge is important if its removal from the network would render at least one other edge inconsistent.

Definition 1. *Given a strongly connected graph G, a* directed ear decomposition *is a sequence $P_0, P_1, \ldots, P_k$ such that P_0 is a simple directed cycle of G and for every $i > 0$, P_i is a simple directed path in G such that the first and last node of P_i are in $\cup_{j<i} P_j$ and the internal nodes of P_i do not belong to $\cup_{j<i} P_j$. Each P_i is referred to as an* ear *of G.*

Clearly, a directed ear decomposition is a strongly connected spanning subgraph (SCSS) of G. Our key observation is expressed by the following theorem.

Theorem 1. *Let $G = (V, E)$ be a flow network and let F be an integral feasible flow in G. The number of important edges in G is bounded from above by $|F| + 2|V|$, where $|F|$ is the maximum number of edges participating in a feasible flow (which can be bounded from above by the value of the flow because it is an integral flow).*

Proof. We construct a set of $|F| + 2|V|$ edges and show that it contains all important edges. First, we include all $|F|$ flow edges. A non-flow edge between two SCCs of the residual graph cannot be important. We proceed by analyzing each SCC of G separately.

Let C be an SCC of G and consider an ear decomposition of S from which we remove every ear that consist of a single edge. Let $P_0, P_1, \ldots, P_k$ be the remaining ears and let $P = \cup_{i=1}^{k} P_i$ be their union. A simple induction on k shows that P is a strongly connected subgraph of C. Since P spans G, we know that G remains strongly connected as long as P is its subgraph. In other words, only edges in P can be important. Finally, we can show by induction on k that $|P| < 2|V|$. □

3 Important Edges For *AllDifferent* and *GCC*

The flow network corresponding to the *AllDifferent* and *GCC* constraints contains edges that do not correspond to variable-value pairs. Therefore, a refined upper bound is due. The proof of the following theorem is similar to the proof of Theorem 1, while counting only edges between nodes of the network that represent variables of the constraint or values in their domains, and noticing that $|F| \leq n$.

Theorem 2. *Let C be a constraint defined on a set X of n domain variables and let G be a flow model for C. The number of important variable-value pairs in C is at most $3n$.*

Perhaps a more careful analysis could yield a tighter upper bound, but we wish to point out that the upper bound in Theorem 2 is within a constant factor of optimal. Figure 1 shows the variable-value graph of an *AllDifferent* constraint with $2n$ important

edges: There are two solutions, and each edge belongs to exactly one of them. Removing any edge would destroy one of the solutions and turn the other edges that belong to it into inconsistent edges.

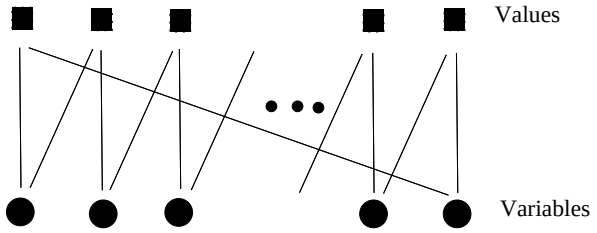

Fig. 1. An *AllDifferent* variable-value graph with $2n$ important edges.

4 Delayed Filtering: A Monte-Carlo Algorithm

Assume that when a value is removed from the domain of one of the variables of a constraint, it is selected at random among all possibilities. In other words, a random edge is removed from the variable-value graph. Selection without replacement corresponds to the *hypergeometric distribution*. If we have a bin with M balls, of which N are black and the remaining are white, and we randomly select x balls (without replacement), the expected number of selected black balls is xN/M. In other words, after selecting M/N balls, we are expected to have selected one black ball. Returning to our setting, we get that on expectation, an important edge is lost after $m/3n$ edges were removed from the variable-value graph of an *AllDifferent* or *GCC* constraint.

If we re-filter with respect to the constraint only after losing $m/3n$ edges, we need to recompute arc consistency from scratch, which takes $O(m\sqrt{n})$ time. On the other hand, if we re-filter every time an edge is lost, we spend a linear amount of time for each edge, for a total of $O(m^2/n)$. Whenever $m = \omega(n\sqrt{n})$, the delayed re-filtering is asymptotically faster.

Corollary 1. *For a dense instance of AllDifferent or GCC, re-filtering after $\Theta(m/n)$ edges are lost, is more efficient and its effectiveness, in the expected sense, is comparable to that of re-filtering every time an edge is lost.*

5 Filtering Triggers: A Las Vegas Algorithm

As mentioned in the introduction, we wish to identify a subset S of the edges of the variable-value graph whose removal should trigger a re-filtering. It may seem natural to try maintaining the set of important edges under edge removal (filtering). However, the removal of a non-important edge can cause an arbitrary number of non-important edges

```
procedure DomainEvent(G = (V,E))
    (* Input: The variable-value graph G. *)
    If EventCount= 0 Then
      Enforce arc consistency
      EventCount ← |E|/(3|V|)
    Else
      EventCount← EventCount−1
    Endif
End
```

Fig. 2. Delayed Filtering for *AllDifferent*.

to become important. Hence, it is not clear how to do this efficiently. In contrast, the set of filtering triggers that we identify has a stronger property: It is a set S of edges that contains all important edges and will contain all important edges under edge removal as long as no edge from S is filtered.

Observation 1 *Any strongly connected spanning subgraph (SCSS) has this property.*

Finding the minimum SCSS is NP-hard [2]. On the other hand, we have already described a linear-time algorithm that finds a 2-approximation: Find a directed ear decomposition by a depth-first search traversal of the graph and remove ears that consist of a single edge. A more sophisticated algorithm achieves a $5/3$-approximation in linear time [7].

With this in mind, it is possible to implement a Las Vegas algorithm which computes a perfect pruning but improves the running time in an expected sense. The algorithm is shown in Figure 3 and works as follows. After computing arc consistency, it computes an $O(n)$-size SCSS for the residual graph using one of the above techniques. It then re-filters the constraint only when a matching or SCSS edge is removed. The filtering achieved is perfect and equivalent to enforcing arc consistency at every search step. The time spent on filtering is, in the worst case, asymptotically the same. However, in the expected case, the Las Vegas algorithm re-filters only once every $\theta(m/n)$ domain events. Note that the algorithm re-filters every time a single matching edge is lost, so we can use the linear-time incremental filtering algorithm. This means that the expected time spent on filtering is reduced by a factor of m/n.

Corollary 2. *For an instance of AllDifferent or GCC whose variable-value graph has n nodes and m edges, re-filtering only after a flow edge or an SCSS edge is lost, achieves the same amount of filtering as enforcing arc consistency after each domain event. On expectation, the total time spent on filtering is reduced by a factor of m/n.*

6 Conclusion and Directions for Further Research

In the design of filtering algorithms for global constraints we should take into account the context in which they are used. For example, it is usually more important to find a

```
procedure ReFilter(G, M, S, e)
(* Input: The variable-value graph G, a matching M in G,
an SCSS S of the residual graph and an edge e to be removed. *)
    If e ∈ M Then
      M ← a matching that excludes e.
    Endif
    Remove e from G.
    Enforce arc consistency
    S ← a small (O(n)-size) SCSS for the residual graph.
End

procedure DomainEvent(G, M, e)
(* Input: The variable-value graph G, a matching M in G,
an SCSS S of the residual graph and an edge e to be removed. *)
    If e ∈ S∪M Then
      ReFilter(G, M, S, e)
    Else
    Remove e from G.
End
```

Fig. 3. Filtering Triggers for *AllDifferent*.

fast incremental algorithm than to fine-tune the algorithm that filters a constraint from scratch. Another example is the analysis of propagation patterns (e.g., [3]), which deals with the question of how many times a constraint can be re-filtered before a fixpoint is reached. In this paper we suggested an additional way to look at propagation in the context of the search.

The Monte-Carlo algorithm improves the running time but it may filter less and its analysis disregards the propagation of filtering between the constraints. In other words, when we delay the removal of a useless edge in the Monte Carlo algorithm, we also delay the propagation of this removal. Furthermore, if the CSP contains only complex constraints and the solver delays filtering for all of them, it could take many search steps before *any* propagation is performed, and this is clearly not what we intended. In summary, it is not immediately clear how to apply the Monte Carlo algorithm in practice. An evaluation, and possibly a refinement of our model of the propagation-search process is necessary.

The Las Vegas algorithm we suggest appears to be more promising in practice, because it offers a speedup on expectation without any loss in the worst case. Still its runtime analysis is based on the assumption that the probability of removing an edge is uniformly distributed. Is this assumption reasonable in practice? Again, the model needs to be evaluated and possible refined. Note, however, that the overhead of the Las Vegas algorithm is small and does not change the asymptotic complexity. So from a theoretical standpoint at least, the Las Vegas algorithm is elegant and appealing. A fundamental research topic is thus to study how similar techniques can be designed for other global constraints and the quantify the gains in practice.

Acknowledgements

We wish to thank the CP'AI'OR 2006 referees for numerous helpful suggestions, many of which were incorporated into the discussion in Section 6.

References

1. J.E. Hopcroft and R.M. Karp. An $n^{5/2}$ algorithm for maximum matching in bipartite graphs. *SIAM J. Computing*, 2(4):225–231, 1973.
2. S. Khuller, B. Raghavachari, and N. Young. Approximating the minimum equivalent digraph. *SIAM J. Comput.*, 24(4):859–872, 1995.
3. L. Mercier and P. Van Hentenryck. Strong polynomiality of resource constraint propagation, 2005.
4. C.-G. Quimper, A. López-Ortiz, P. van Beek, and A. Golynski. Improved algorithms for the *global cardinality* constraint. In *CP 2004*, volume 3258 of *LNCS*, pages 542–556. Springer-Verlag, 2004.
5. J.-C. Régin. A filtering algorithm for constraints of difference in CSPs. In *AAAI-94*, pages 362–367, 1994.
6. J.-C. Régin. Generalized Arc-Consistency for Global Cardinality Constraint. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96)*, pages 209–215, 1996.
7. L. Zhao, H. Nagamochi, and T. Ibaraki. A linear time 5/3-approximation for the minimum strongly-connected spanning subgraph problem. *Inf. Process. Lett.*, 86(2):63–70, 2003.