The Extended Global Cardinality Constraint: An Empirical Survey: Extended Abstract*

Peter Nightingale
School of Computer Science, University of St Andrews, St Andrews, Fife KY16 9SX, UK
pwn1@st-andrews.ac.uk

Abstract

The Extended Global Cardinality Constraint (EGCC) is an important component of constraint solving systems, since it is very widely used to model diverse problems. The literature contains many different versions of this constraint, which trade strength of inference against computational cost. In this paper, I focus on the highest strength of inference usually considered, enforcing generalized arc consistency (GAC) on the target variables. This work is an extensive empirical survey of algorithms and optimizations, considering both GAC on the target variables, and tightening the bounds of the cardinality variables. I evaluate a number of key techniques from the literature, and report important implementation details of those techniques, which have often not been described in published papers. Two new optimizations are proposed for EGCC. One of the novel optimizations (*dynamic partitioning*, generalized from AllDifferent) was found to speed up search by 5.6 times in the best case and 1.56 times on average, while exploring the same search tree. The empirical work represents by far the most extensive set of experiments on variants of algorithms for EGCC. Overall, the best combination of optimizations gives a mean speedup of 4.11 times compared to the same implementation without the optimizations.

This paper is an extended abstract of the publication in Artificial Intelligence [Nightingale, 2011].

Constraint programming is a powerful and flexible means of solving combinatorial problems. Constraint solving of a combinatorial problem proceeds in two phases. First, the problem is *modelled* as a set of *decision variables*, and a set of *constraints* on those variables that a solution must satisfy. A decision variable represents a choice that must be made in order to solve the problem. The *domain* of potential values associated with each decision variable corresponds to the options for that choice.

Consider a sports scheduling problem, where each team plays every other team exactly once in a season. No team can play two or more matches at the same time. Each team plays in a particular stadium at most twice during the season. In this example one might have two decision variables per match, representing the two teams. For a set of matches played in the same stadium, a global cardinality constraint [Régin, 1996] could be used to ensure no more than two occurrences of each team.

The second phase consists of using a constraint solver to search for *solutions*: assignments of values to decision variables satisfying all constraints. The simplicity and generality of this approach is fundamental to the successful application of constraint solving to a wide variety of disciplines such as scheduling, industrial design and combinatorial mathematics [Wallace, 1996; Huczynska *et al.*, 2009].

The Global Cardinality Constraint (GCC) is a very important global constraint, present in various constraint solving toolkits, solvers and languages. It restricts the number of occurrences of values assigned to a set of variables. In the original version of the constraint [Régin, 1996], each value is given a lower bound and upper bound. In any solution, the number of occurrences of the value must fall within the bounds. The literature contains many propagation algorithms for this constraint, which trade strength of inference against computational cost, for example bound consistency [Katriel and Thiel, 2005; Quimper *et al.*, 2003], range consistency [Quimper *et al.*, 2004], and generalized arc-consistency (GAC) [Régin, 1996; Quimper *et al.*, 2004]. GCC is widely used in a variety of constraint models, for diverse problems such as routing and wavelength assignment [Simonis, 2009], car sequencing [Régin and Puget, 1997], and combinatorial mathematics [Huczynska *et al.*, 2009].

Returning to the sports scheduling example, GCC can be used to express the stadium constraint (that a team plays in a particular stadium at most twice during the season). Each value (representing a team) is given the bounds $(0, 2)$, and the variables are all slots at a particular stadium.

GCC has been generalized by replacing the fixed bounds on values with *cardinality variables* [Quimper *et al.*, 2004], where each cardinality variable represents the number of occurrences of a value. To avoid confusion, I refer to this as the Extended Global Cardinality Constraint (EGCC). Thus an EGCC constraint has *target* variables (where the number of occurrences of some values are constrained) and *cardinality*

*This paper is an extended abstract of the AI Journal publication [Nightingale, 2011].

variables.

In this paper, I focus on the highest strength of inference (enforcing GAC) on the target variables. This allows the study of various methods in depth, and leads to some surprising conclusions. I also survey methods for pruning the cardinality variables in depth. The main contributions of the paper are as follows.

- A literature survey of GAC propagation algorithms for the target variables, and their optimizations.
- Discussion of important implementation decisions that are frequently omitted from original papers, perhaps due to lack of space. For example, how to find augmenting paths for Régin's algorithm [Régin, 1996].
- The proposal of two new optimizations. One of these is based on modifying the flow network of Régin's algorithm for greater efficiency, and the other is a novel generalization of the dynamic partitioning optimization of AllDifferent [Gent *et al.*, 2008].
- A careful description of three concrete algorithms for pruning the cardinality variables.
- Easily the largest empirical study of GAC propagation methods for the target variables of EGCC. This involves two basic algorithms and seven optimizations.
- Experimental conclusions and implementation advice for GAC for the target variables.
- An empirical study of pruning the cardinality variables, comparing the three methods.

It is shown that an appropriate combination of optimizations is over 4 times faster on average than a careful but unoptimized implementation of Régin's algorithm for our benchmark set.

A fast variant of EGCC is typically orders of magnitude better than a set of occurrence constraints (one constraint for each value of interest). Even when EGCC propagation was least effective, it slowed the solver down by only 1.66 times or less in the experiments.

Background

Régin's algorithm [Régin, 1996] and Quimper's algorithm [Quimper *et al.*, 2004] for pruning EGCC make use of network flow and bipartite matching theory [Cormen *et al.*, 1990] as well as strongly connected components (SCCs) [Tarjan, 1972]. Similarly, Régin's AllDifferent algorithm [Régin, 1994] makes use of results from graph theory, in particular maximum bipartite matching [Berge, 1973] and strongly connected components.

Both Régin's algorithm and Quimper's algorithm prune only the target variables of the EGCC constraint. The cardinality variables may be pruned by a network flow algorithm (also by Quimper) [Quimper *et al.*, 2004], or by simpler means.

Pruning the Target Variables

In this section I will briefly summarise the main results of the paper regarding pruning the target variables.

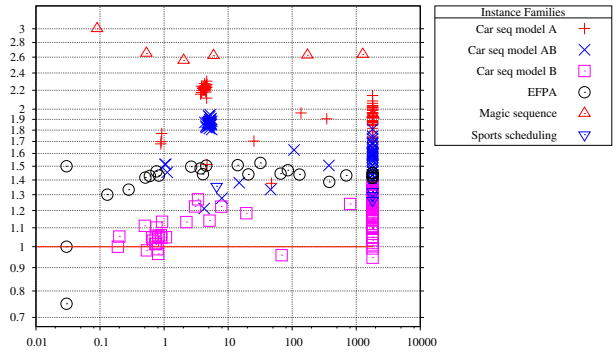

Figure 1: Speedup of Baseline-Régin compared to Baseline-Quimper. The x-axis represents the run time of Quimper's algorithm to solve the instance. The y-axis gives the speedup ratio obtained by using Régin's algorithm as opposed to Quimper's algorithm. Points above 1 on the y-axis indicate Régin's algorithm is faster. The ratio is of node rates, and includes the full cost of the solver not just the EGCC constraints. In this graph we can see that Régin's algorithm almost always performs better than Quimper's algorithm.

Quimper's Algorithm vs Régin's Algorithm

The paper compares the two algorithms for GAC pruning of the target variables. Quimper's algorithm [Quimper *et al.*, 2004] is superior in big-O terms, but Régin's algorithm [Régin, 1996] has the advantage of simplicity, and in fact performs better as shown in Figure 1. Régin's algorithm is used throughout the rest of the paper.

Dynamic Partitioning

Gent *et al.* [2008] proposed an algorithm which partitions an AllDifferent constraint during search. Suppose for example we have AllDifferent$(x_1 \dots x_6)$ and have $x_1 \dots x_3 \in \{1 \dots 3\}$, $x_4 \dots x_6 \in \{4 \dots 6\}$. This can be partitioned into two independent cells: AllDifferent$(x_1 \dots x_3)$ and AllDifferent$(x_4 \dots x_6)$. The main benefit is that if some variable x_i has changed, the propagator need only be executed on the cell containing x_i, not the original constraint. This saves time in the second stage of Régin's AllDifferent algorithm.

A cheap way of obtaining the partition is to use the SCCs of the residual flow network, which are computed as part of Régin's AllDifferent algorithm. In some cases it is possible to find a finer partition than the SCCs. However, experiments showed that using SCCs as the partition is effective in practice [Gent *et al.*, 2008].

In this paper I generalise dynamic partitioning to the EGCC constraint. The partition is obtained from the SCCs of the residual flow network, as in AllDifferent, but unlike AllDifferent cells must contain both variables and values. This proves to be an effective optimisation. Figure 2 shows the results of an experiment comparing dynamic partitioning to a baseline implementation of Régin's algorithm.

Important Edges

Katriel [2006] observed that many value removals affecting a GCC constraint result in no other value removals, and so

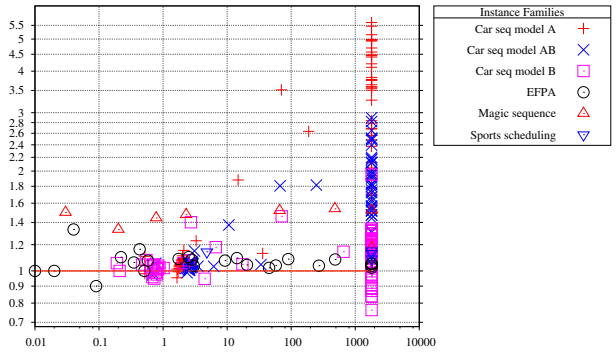

Figure 2: Speedup of dynamic partitioning compared to the baseline Régin's algorithm, in the same format as Figure 1.

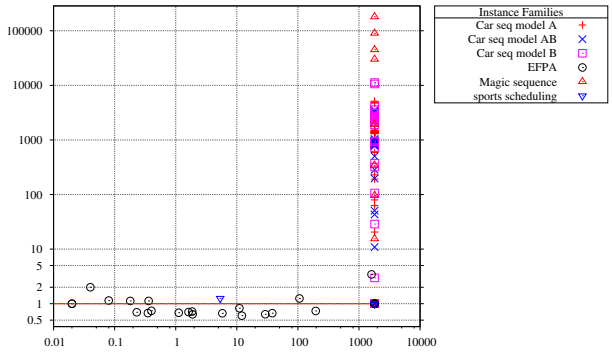

Figure 3: Time comparison between EGCC-Sum and Occurrence. The x-axis is the time taken by Occurrence, and the y-axis is the proportion of total times, Occurrence divided by EGCC-Sum. The timeout was 1800s.

work processing them is wasted. She introduces the concept of an *important edge* of the residual graph. An important edge is one whose removal causes the pruning of some variable-value pair. Therefore, when an *un*important edge is removed, it is not necessary to run the propagator.

There are approximately 3 important values per variable at any time, regardless of domain size. We found that exploiting important edges is useful with the basic algorithm, but when combined with other optimisations it does not repay its overhead in our experiments. It is possible that when the target variables have a large number of values this idea will come into its own.

Combining Optimisations

Overall I found that the following optimisations work well together: incrementally updating the maximum flow, incrementally maintaining an internal representation of the flow network, and dynamic partitioning. With these optimisations the experiments show a mean speed up of 4.11 times, and a maximum of 20.9 times while performing an identical search.

Pruning the Cardinality Variables

The paper compares three algorithms for pruning the cardinality variables. The simplest (Simple) counts the occurrences of each value in the target variables' domains. The second (Sum) combines Simple with an additional constraint stating that the cardinality variables sum to r (where r is the number of target variables).[1] Sum is only correct when the EGCC covers every value in the domain of the target variables. The third (Flow) uses a strong network flow algorithm by Quimper [Quimper *et al.*, 2004]. The following table compares the three. For each cardinality algorithm, it shows the number of instances solved, and the number of instances with a reduced node count vs the weaker algorithms.

[1] Sum is the method implemented in the Gecode solver (Guido Tack, personal communication).

	Instances solved (of 194)	Saved nodes vs Simple	Saved nodes vs Sum
Simple	108	—	—
Sum	109	20	—
Flow	111	33	23

Sum adds very little overhead and is usually worthwhile compared to Simple. Flow on the other hand is very expensive and is not normally worthwhile, but it does solve two instances that Sum cannot.

Comparing to a Decomposition

The Occurrence constraint takes an array of target variables, one cardinality variable and a value. It constrains the cardinality variable to be the number of occurrences of the value in the array. EGCC trivially decomposes into one Occurrence constraint for each value of interest. Figure 3 compares EGCC (with Sum) to Occurrence. As expected, EGCC can solve many instances where Occurrence hits the time limit. EGCC can be orders of magnitude faster by reducing search. For some EFPA instances EGCC is not effective, and on these instances EGCC is at most 1.66 times slower than Occurrence.

Summary

The paper presented an extensive survey of propagation methods for the EGCC constraint, studying the pruning of both target variables and cardinality variables, surveying many methods from the literature and presenting some methods that have not been previously reported.

I focused on generalized arc-consistency for the target variables (GAC-On-X) and evaluated two basic algorithms from the literature along with five optimizations found in the literature, and two novel optimizations. In each case I have reported on their implementation and given an empirical analysis of their behaviour. While it was impossible to experiment with every possible combination of optimizations, I took care to compare each optimization against an appropriate baseline method, and to avoid straw men. Particular attention was

paid to evaluating *combinations* of optimizations, which is (naturally) not usually a feature of papers that propose optimizations. The experiments presented here comprise easily the deepest experimental analysis of GAC-On-X algorithms. Based on them, I was able to conclude that some optimizations are key and others are less generally useful.

I would like to draw particular attention to the results with dynamic partitioning, a novel generalization of an optimization for AllDifferent [Gent *et al.*, 2008]. With EGCC dynamic partitioning was 1.56 times faster on average, with a maximum of 5.6 times. The largest gains were seen on the most difficult instances where the solver timed out. The gain for EGCC is less pronounced than for AllDifferent [Gent *et al.*, 2008], albeit on entirely different benchmarks, and with a different combination of other optimizations.

For the best combination of optimizations, I found a mean improvement of more than 4 times in runtime over a careful but unoptimized implementation of Régin's algorithm. This confirms that optimizations are an essential part of a practical implementation of EGCC.

Regarding the cardinality variables, I was able to confirm that the implied sum constraint used by Gecode is indeed valuable, and also that the stronger flow-based pruning algorithm given by Quimper et al [Quimper *et al.*, 2004] can also be valuable, since it solves more instances within a time limit than either other method.

Finally, a fast variant of EGCC is typically orders of magnitude better than a set of occurrence constraints. Even when EGCC propagation was not effective, it slowed the solver down by only 1.66 times or less in the experiments.

Acknowledgements

I owe a debt to many people for helpful discussions, in particular Ian Gent, Ian Miguel, and Guido Tack. I would also like to thank Ian Gent and Ian Miguel for comments on a draft of this paper. I thank Chris Jefferson for helpful discussions and for pointing out that EGCC cannot be entailed, and Andrea Rendl for use of Tailor [Gent *et al.*, 2007] for generating benchmarks. This work was supported by EPSRC grants EP/E030394/1 and EP/H004092/1.

References

[Berge, 1973] Claude Berge. *Graphs and Hypergraphs*. North-Holland Publishing Company, 1973.

[Cormen *et al.*, 1990] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[Gent *et al.*, 2007] Ian P. Gent, Ian Miguel, and Andrea Rendl. Tailoring solver-independent constraint models: A case study with Essence′ and Minion. In *Proceedings of SARA 2007*, pages 184–199, 2007.

[Gent *et al.*, 2008] Ian P. Gent, Ian Miguel, and Peter Nightingale. Generalised arc consistency for the alldifferent constraint: An empirical survey. *Artificial Intelligence*, 172(18):1973–2000, 2008.

[Huczynska *et al.*, 2009] Sophie Huczynska, Paul McKay, Ian Miguel, and Peter Nightingale. Modelling equidistant frequency permutation arrays: An application of constraints to mathematics. In *Proceedings CP 2009*, pages 50–64, 2009.

[Katriel and Thiel, 2005] Irit Katriel and Sven Thiel. Complete bound consistency for the global cardinality constraint. *Constraints*, 10(3):191–217, 2005.

[Katriel, 2006] Irit Katriel. Expected-case analysis for delayed filtering. In J. Christopher Beck and Barbara M. Smith, editors, *CPAIOR*, volume 3990 of *Lecture Notes in Computer Science*, pages 119–125. Springer, 2006.

[Nightingale, 2011] Peter Nightingale. The extended global cardinality constraint: An empirical survey. *Artificial Intelligence*, 175(2):586–614, 2011.

[Quimper *et al.*, 2003] Claude-Guy Quimper, Peter van Beek, Alejandro López-Ortiz, Alexander Golynski, and Sayyed Bashir Sadjad. An efficient bounds consistency algorithm for the global cardinality constraint. In *Proceedings 9th Principles and Practice of Constraint Programming (CP 2003)*, pages 600–614, 2003.

[Quimper *et al.*, 2004] Claude-Guy Quimper, Alejandro López-Ortiz, Peter van Beek, and Alexander Golynski. Improved algorithms for the global cardinality constraint. In *Proceedings 10th Principles and Practice of Constraint Programming (CP 2004)*, pages 542–556, 2004.

[Régin and Puget, 1997] Jean-Charles Régin and Jean-François Puget. A filtering algorithm for global sequencing constraints. In *Proceedings 3rd Constraint Programming (CP 97)*, pages 32–46, 1997.

[Régin, 1994] Jean-Charles Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings 12th National Conference on Artificial Intelligence (AAAI 94)*, pages 362–367, 1994.

[Régin, 1996] Jean-Charles Régin. Generalized arc consistency for global cardinality constraint. In *Proceedings 13th National Conference on Artificial Intelligence (AAAI 96)*, pages 209–215, 1996.

[Simonis, 2009] Helmut Simonis. A hybrid constraint model for the routing and wavelength assignment problem. In *Proceedings of CP-2009*, pages 104–118, 2009.

[Tarjan, 1972] Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[Wallace, 1996] Mark Wallace. Practical applications of constraint programming. *Constraints*, 1(1/2):139–168, 1996.