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

Peter Nightingale

Introduction

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 generalised arc consistency (GAC) on the target variables.

EGCC(X, V, C)

X is a vector of target variables
V is a vector of domain values of interest
C is a vector of cardinality variables

For each value V_i with cardinality variable Ci, there are C_i occurrences of V_i in X

Motivation

Paper is partly empirical survey of existing algorithms....
- Quimper's algorithm vs Régin's algorithm
- Three algorithms for cardinality variables
- Many more

... And partly new optimisations for EGCC
- Dynamic partitioning
- Dynamic triggers

Help future solver implementors
- Simple algorithms better than complex ones, despite big-O complexity
- Insight into which parts of code to optimise, despite big-O complexity, again
- How to prune cardinality variables

Techniques for EGCC might apply elsewhere
- Dynamic partitioning for graph/network constraints

Pruning the Cardinality Variables

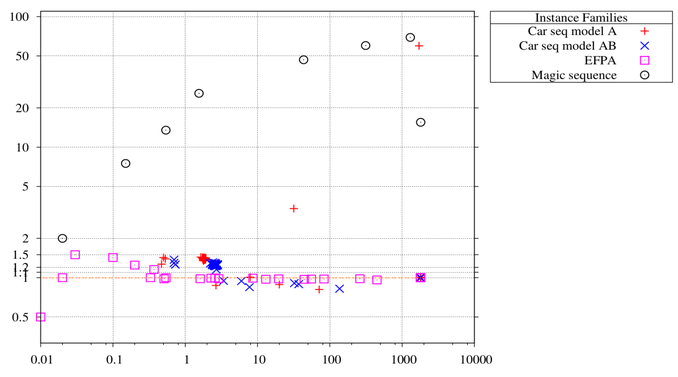
I surveyed three methods:

Simple – for each value, count occurrences in the domains of the target variables (upper bound), count target variables assigned to the value (lower bound)

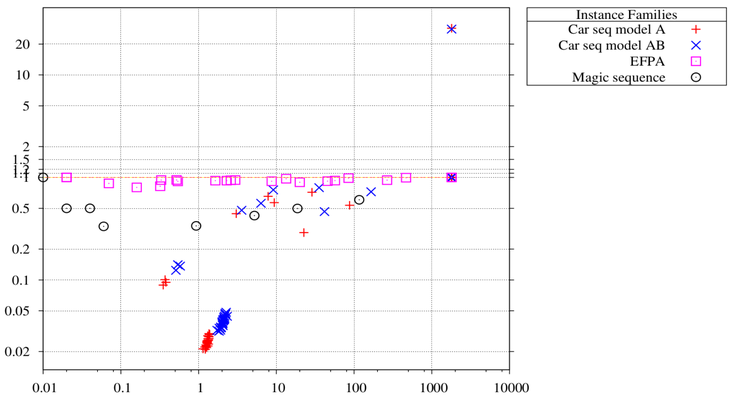
Sum – simple plus implied sum constraint

Flow – for each value, find maximal flows that maximise and minimise occurrences of the value. Much more expensive than Sum.

Simple vs. Sum – The sum constraint is often worthwhile and usually does not have a high cost..



Sum vs. Flow – The plot below shows that on some instances the Flow method can be 50 times slower, but also can solve two more instances within the time limit.



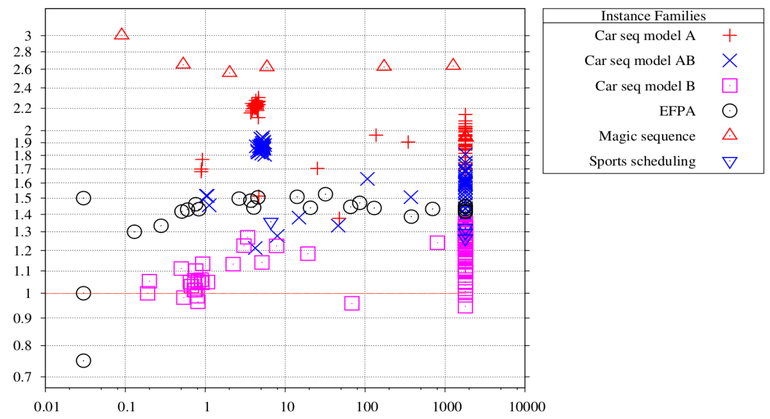
Quimper's vs. Régin's algorithm

There are two algorithms for enforcing GAC on the target variables:

Régin (1996) – Finds one maximal flow, SCC analysis once. Based on network flow, $O(n^2d)$

Quimper et al (2004) – Divides the EGCC into two constraints for the lower and upper bounds (on cardinality), Finds two matchings and runs SCC analysis twice. Based on bipartite matching, $O(n^{1.5}d)$

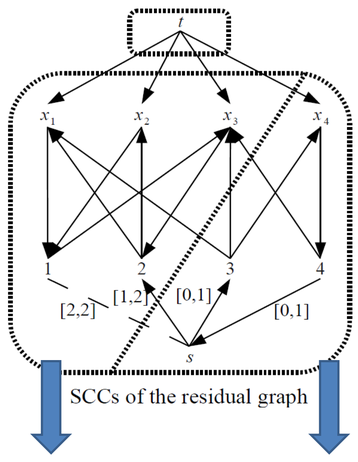
Against the big-O analysis, Régin's algorithm is much better:



Spends most time in SCC analysis not finding the matching/flow.

Dynamic Partitioning

The idea is simple: when the network of the EGCC constraint partitions into two pieces, split the constraint accordingly.

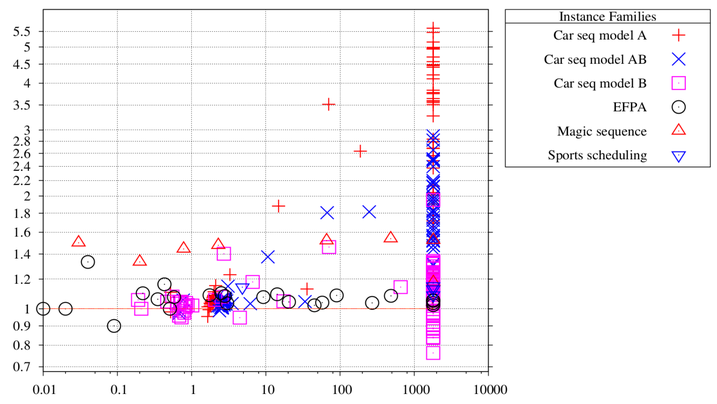


x_1,x_2,x_3
1,2

x_4
3,4

This makes the SCC analysis incremental and can make it more than 5 times faster (plot below compares whole solver time).

Dynamic Partitioning also works well for the AllDifferent constraint and could be promising for other graph or network based constraints.



Summary

The paper 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 also report important implementation details of those techniques, which have often not been described in published papers. As well as a survey, two new optimizations are proposed for EGCC. Overall, the best combination of optimizations gives a mean speedup of 4.11 times, taking the whole time of the solver.