Global Reasoning on Sets

Andrew Sadler and Carmen Gervet
ajs2@icparc.ic.ac.uk, cg6@icparc.ic.ac.uk

IC-Parc, Imperial College,London SW7 2AZ, UK

Abstract. Finite set constraint systems represent a natural choice to model combinatorial configuration problems involving set disjointness, covering or partitioning relations. However, for efficiency reasons, alternative formulations based on Finite Domain or 0-1 integer programming are often preferred even though they require much modelling effort. To offer a better trade-off "natural formulation"/efficiency we propose to improve the efficiency of set constraint solvers by introducing global reasoning on a class of finite set constraints. These are n-ary constraints like `atmost1-incommon`, `distinct` upon sets of known cardinality. In this paper we show how the representation of sets within powersets specified as set intervals allows us to derive some global pruning based on mathematical and combinatorial analysis formulas. They improve greatly the filtering enforced by bound consistency methods, and allow to detect failure at early stages. Preliminary results are illustrated on the ternary Steiner and a generic `distinct` problems.

1 Introduction

Finite set constraints solvers have been embedded in quite a few CP languages (e.g. Conjunto[Ger97], Ilog solver, OZ[Mul98], CLPS) and shown their strengths in modelling combinatorial configuration problems like partitioning, covering and matching problems with a natural mathematical formulation[1] . Conjunto and its peers comprise the usual set operation symbols ($\cap, \cup, \setminus$), the set cardinality relation (#) and the set inclusion relation ($\subseteq$). For practical modelling reasons Conjunto provides a set of n-ary constraints like `all_disjoint`, `all_union` which are syntactic abstractions for a collection of respectively binary and ternary constraints ($s_1 \cap s_2 = \emptyset$, $s_1 \cup s_2 = s_{12}$). Set variables range over set domains (sets of sets) specified by intervals whose lower and upper bounds are known sets, ordered by set inclusion. The constraint reasoning is based on local bound consistency techniques extended to handle set constraints [Ger97].

Clearly finite set constraints allow us to represent a combinatorial configuration problem without any abstraction from the mathematical specification. This natural mapping from the specification to the model has several advantages among which: (1) easy to maintain and implement changes to the specification (addition/removal of constraints), (2) genericity of the model applicable to any problem instances, (3) conciseness of the model easily readable by a third programmer. However when tackling real world problems one is searching for the best trade-off between ease of modelling and solving efficiency. This can be achieved either by using alternative modelling approaches with good resolution algorithms or by keeping a set based model and searching for more effective filtering methods. Current solutions are based on the former benefiting from considerable amount of work on efficient filtering and search methods for FD models [BC94,Rég94] [GS00], while this paper proposes solutions to the latter.

We propose a cost effective means to improve the pruning power of finite set solvers by identifying a class of constraints which can benefit from a global reasoning. Consider for example the following

[1] We refer to configuration problems as the search for mappings of a set of objects into a finite abstract set that satisfies some given constraints.

simple system of constraints:

$$[s_1, s_2, s_3] \in \{\{\}..\{a, b, c, d\}\}$$
$$\#s_1 = \#s_2 = \#s_3 = 2$$
$$\texttt{disjoint}(\{s_1, s_2, s_3\})$$

Local bound consistency does not detect unsatisfiability. However if we consider the cardinality constraints together with the n-ary disjointness constraint one can see that there are no solutions by doing a simple pigeon hole test. This can be deduced if we consider the set of constraints globally, ie. the disjointness constraint together with the set cardinality ones, and we check given some mathematical formulas that a solution exists. A simple satisfiability test can first be checked, determining whether a set of 4 elements can be partitioned into 3 sets of 2 which fails ($\frac{4}{3} \neq 2$). A more elaborate test that does not require the sets to have same cardinalities derives an upper bound on the number of possible partitions of 4 elements into 3 sets of cardinality 2, namely the Sterling number of order two, $\frac{4!}{(2!)^3(3!)} = \frac{1}{2}$[Ber71]. If it is less than 1 the problem is unsatisfiable since there is no possible partition. However if the number is greater than 1 we would know how many different partitions there are.

This form of global reasoning builds upon known set cardinalities, which occurs in most combinatorial configuration problems, and thus covers a large class of practical problems. In this paper, we present two instances of such a global reasoning applied to the n-ary set constraints $\texttt{atmost1}(s_1, ..., s_n)$ where all the sets should intersect pairwise in atmost one element, and $\texttt{distinst}(s_1, ..., s_n)$ where all the sets should differ pairwise in atleast one element. Our main contribution is to improve the efficiency of set-based constraint solver in a *generic* way by combining basic constraints when a global mathematical reasoning adds further restrictions to the local bound consistency inference rules. The global reasoning builds upon the addition of conditional rules, which are straightforward to model because we deal with collections of sets defined within a powerset. The modelling and cost effective use of these rules would be very cumbersome when considering alternative set modelling approaches, mainly because it is relatively difficult to derive and manipulate cardinalities of intensional sets, ie. sets sharing some properties when the sets are built from composite terms. We illustrate the strength of this approach on instances of the ternary Steiner and generic `distinct` problems.

2 Sets and combinatorial analysis

Combinatorial analysis deals with configurations [Ber71]. It counts and investigates the existence of configurations with certain specified properties. In our context, we view each solution of a set CSP as a configuration, where sets of objects are incrementally built from their domains and the properties are the constraints over the sets. A solution to a set based CSP is in fact a configuration within the lattice $\mathcal{P}(\mathcal{S})$ or powerset of S, where S is the union of all the set domains upper bounds. The search for configurations within S amounts to searching for a certain collection of elements of $\mathcal{P}(S)$. In the absence of constraints/properties this number is exponential in the size of S. However the more constraints there are, the more we can derive new information and reduce the number of potential configurations. A key point is that such new information can be modelled directly using symbols and terms from the set CSP constraint domain.

In particular, there exist some counting functions that determine the maximum number of configurations allowed in a superset S, given some shared properties. For the two n-ary constraints we present in this article, we will consider the following counting functions:

F1. The number of distinct subsets of size n of S (size m) is the binomial number of m with respect to n:

$$C_m^n = \frac{m!}{(m-n)! \times (n)!}$$

F2. There exists a partition of S into k sets of size n if $\frac{m}{k} = n$

When considering the values of these functions on can then investigate how and when they can be used effectively, first to detect unsatisfiability but also to prune further irrelevant set values in an a priori manner. The counting functions presented above provide a mathematical information that is not easily deducible in logic. This information can be used *successfully* to perform some global reasoning if one can ensure that: (1) the functions can be computed fast, (2) the derived numbers are relatively small allowing the limit to be reached before or at the beginning of the search, (3) the formulas can be expressed easily without any need for link variables or constraints, (4) the functions can be maintained incrementally as new sets satisfy the properties at hand. The two functions we have introduced above allow us to achieve these results using set intervals. We present the new inference rules for two n-ary constraints applied to sets of known cardinality.

3 atmost1($[s_1, ..., s_n]$)

The `atmost1` constraint states that l sets of known cardinality should intersect pairwise in atmost one element. The global reasoning is built upon the handling of a collection of sets S_a that intersect in one value a, and the maximum number of sets s_j that can contain this value, denoted S^a. It says basically that if each set s_i has a cardinality c, and the least upperbound of $\top^a = lub(S^a)$ (union of the s_j upperbounds) contains $\#\top^a$ elements, then function `F2` implies that the maximum number of sets that can contain a is $\lfloor \frac{\#\top^a - 1}{c-1} \rfloor$. The problem can be stated as follows:

$$(1) \begin{cases} \texttt{atmost1}([s_1, ..., s_n]), \forall s_i, \#s_i = c \\ \top = \bigcup_i lub(s_i), \quad \#\top = u \\ \forall a \in \top, S_a = \{s_i \mid a \in glb(s_i)\}, S^a = \{s_i \mid a \in lub(s_i)\} \\ \top^a = lub(S^a) = \bigcup_{s_i \in S^a} lub(s_i) \quad Max^a = \frac{\#\top^a - 1}{c-1} (see\ \texttt{F2}) \\ \Delta_a = \top \backslash glb(S_a) \end{cases}$$

We defined three global conditional rules to check satisfiability and infer further pruning based on the value of Max^a:

A. If the number Max^a is exceeded the constraint is unsatisfiable

$$\frac{Max^a < \#S_a}{(1) \vdash fail}$$

B. If Max^a is reached then the element a can not belong to any set outside S_a

$$\frac{Max^a = \#S_a}{(1) \vdash \forall s_j \notin S_a, a \notin s_j}$$

C. If Max^a is not yet reached and there are $m - 1$ elements left to be assigned, further inferences can be obtained.

$$\frac{Max^a > \#S_a,\ \#\Delta_a = c - 1}{(1) \vdash \quad \forall s_j \notin S_a, \Delta_a \subset s_j \vee a \notin s_j}$$

In Figure 1, we illustrate graphically the behavior of rules A and B using the lattice representation of partially instantiated sets specified as set intervals.

Experimental studies on the pruning power and effectiveness of this global reasoning are presented in section 5 on the ternary Steiner problem.

Complexity. The time complexity of applying each of the above rules is bounded by $\mathcal{O}(n \times u)$ since we iterate over all the sets and perform ground set operations (membership and inclusion) taking in the worst case u steps. Time complexity for computing/updating sets and cardinalities in (1) like $\top$ and Δ_a is bounded by $\mathcal{O}(n \times u \times \log(u))$ but can be reduced to $\mathcal{O}(\log(u))$ if maintained incrementally[2].

[2] The time complexity results assume an internal representation for ground sets being an ordering list.

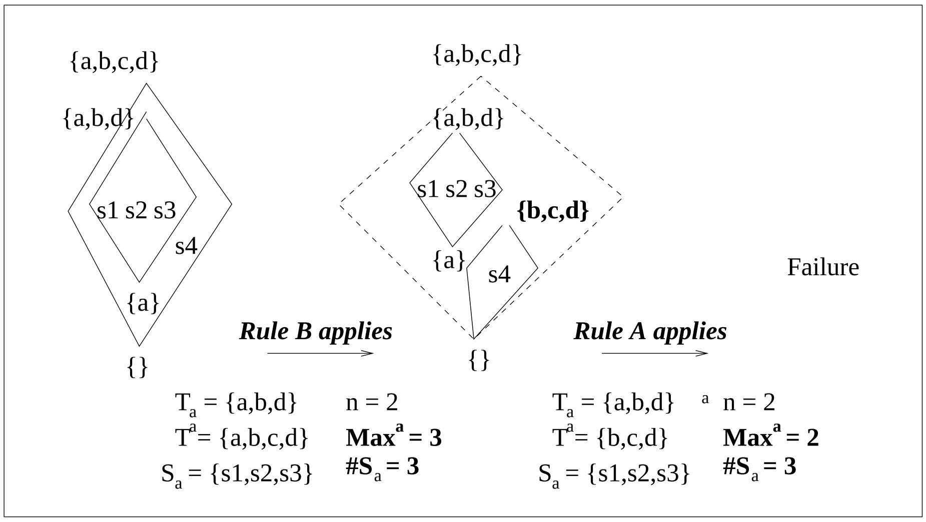

Fig. 1. Rule B is first triggered for the value a since $Max^a = \#S^a$ thus the maximum number of sets containing a has been reached. The element a is deleted from the upper bound of s_4. The new program state triggers rule A since now we have $Max^a < \#S^a$ which leads to failure.

4 distinct($[s_1, ..., s_n]$)

The `distinct` constraint is the complement of the `atmost1`. It states that all the sets should differ pairwise in *atleast* one element. When dealing with sets of fixed size further restrictions can be imposed if the set cardinality restrictions are handled together with the distinct constraint. We present two conditional rules which perform such reasoning and build upon the binomial number referred to in `F1`, and the cliques of sets sharing their lower bounds[3].

$$(2)\begin{cases} \texttt{distinct}([s_1, ..., s_n]), \forall s_i, \#s_i = c \\ \#glb(s_i) = l_i \\ \top = \bigcup_i lub(s_i), \quad \#\top = u \\ \forall s_i,\ clique_i = \{s_j \mid glb(s_j) = glb(s_i)\} \\ \#clique_i = k_i \\ \top_{clique_i} = \bigcup_{s_j} lub(s_j), \quad \#\top_{clique_i} = u_i \end{cases}$$

D. If the clique of sets sharing a *lower bound* contains more sets than it possibly can (according the binomial number) the problem is unsatisfiable:

$$\frac{C_{u_i - l_i}^{c - l_i} < k_i}{(2) \vdash fail}$$

E. If the limit is reached, then for all the sets outside the given clique, they cannot contain the lower bound at hand.

$$\frac{C_{u_i - l_i}^{c - l_i} = k_i}{(2) \vdash \forall s_j \notin clique(s_i), glb(s_i) \not\subseteq s_j}$$

Complexity. Time complexity for applying each rule is bounded by $\mathcal{O}(n \times u)$. The time complexity for calculating/updating (2) and the $C_{c-l_i}^{m_i - l_i}$ is $\mathcal{O}(n^2 \times u \times \log(u))$ which can be reduced to $\mathcal{O}(\log(u))$ if maintained incrementally.

Figure 2 gives an illustrative example for the different inference rules.

[3] A clique of sets is a collection of sets fully connected in terms of their common lower bounds.

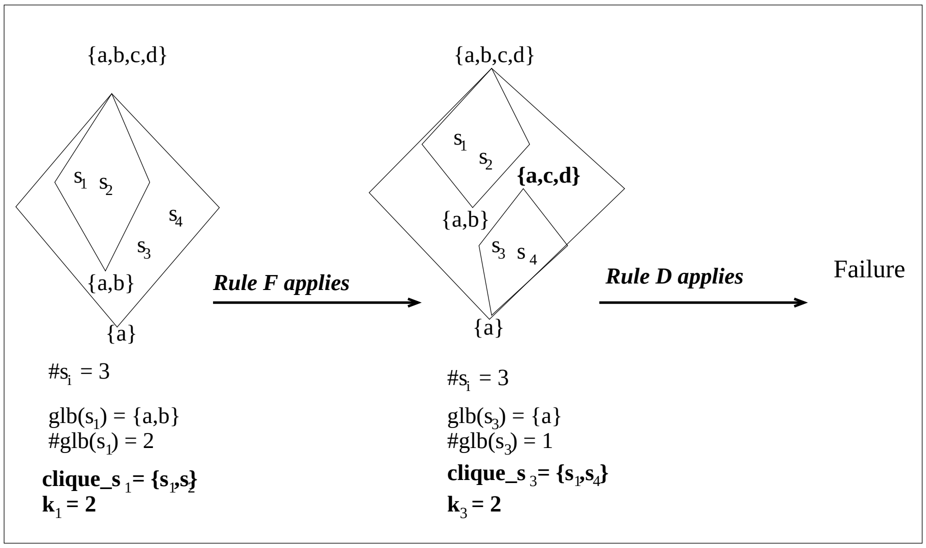

Fig. 2. Rule E is first triggered since for the clique attached to the set s_1 (or s_2) we have $C_{4-2}^{3-2} = 2$, thus sets s_3 and s_4 which already contain a cannot contain b as well. The updated bounds lead to a new program state that triggers rule D for the clique attached to s_3 (or s_4) and detects failure since we have then $C_{3-1}^{3-1} = 1 \neq 2$.

5 Case studies

We illustrate the performance of the global reasoning approach on pure instances of the n-ary constraints. For the `atmost1` constraint we use the ternary Steiner problem, whereas for the `distinct` constraint an empirical study of a large number of generic instances is analysed.

Constraint reasoning is closely tied up with the labelling or search strategy. There are many ways of labelling set variables since one can reason about the *glb* and/or the *lub* of the set domains. In the following studies we *refine* a set domain by adding to the *glb* an element from the *lub* not yet in the lower bound. Since a ground set can contain more than one element, a set domain may need to be *refined* more than once. We considered two simple labelling strategies for selecting the order in which the set variables are considered at each *refine* step:

1. Lexical: The left most set variable is selected and refined till the set is ground.
2. Smallest GLB: The left most variable with the smallest *glb* is selected, and one element is added to each set before attempting to add a second element.

The main reason is to investigate whether the effectiveness of the global reasoning is dependent on the labelling strategy at hand or not.

5.1 Ternary Steiner problem

A ternary Steiner system of order n is a collection of $\frac{n(n-1)}{6}$ sets in $\{1, 2, \ldots, n\}$ such that any two sets have at most one element in common [LR80]. Clearly this problem can be naturally modelled using the global constraint `atmost1` together with the set cardinality constraint.

$$\texttt{atmost1}([s_1, ..., s_k])$$
$$\forall s_i \in S, \#s_i = 3$$

The following tables show the results of a comparative study done with different instances of the Steiner problem ($n = 7, 9, 15$). The following tables show the CPU time and pruning performances of the local (standard bounds consistency as in Conjunto) versus global reasoning on sets, with the different labelling strategies introduced earlier.

Note that the local reasoning applies bound consistency inference rules [Ger97] over the following constraint between any pair of set variables: $\#(s_1 \cap s_2) \leq 1$.

	Global		Local	
n	nb. failures	CPU time	nb. failures	CPU time
7	0	50ms	6	40ms
9	1704	26460ms	4521	2280ms
15	0	5690ms	90	360ms

The global reasoning reduces the number of failure points drastically (by about $\frac{2}{3}$ for Steiner 9, and completely for Steiner 7 & 15. The downside is the CPU time which is increased by 10. This comes from the current implementation of the conditional rules which in this early version is not incremental. However given the complexity results we strongly believe that an incremental version will be comparable if not faster than the local solver in the above cases.

	Global		Local	
n	nb. failures	CPU time	nb. failures	CPU time
7	12	80ms	72166	38710ms
9	414891	4 hours	-	>3 days
15	-	-	-	-

The symbol $-$ indicates that no results were found within a day. The second table shows that this labelling strategy makes a very poor usage of the local inference rules since the constraint $\#(s_1 \cap s_2) \leq 1$ requires one set to be ground in order to prune the other set domain. On the other hand the global conditional rules allow for and early detection of unsatisfiable states for Steiner 7 and 9 in reasonable time. It clearly illustrates how important it is to ensure global reasoning and a priori pruning in order to remain as much independent of the search strategy as possible.

5.2 Distinct

We studied pure instances of the n-ary `distinct` constraint over sets of known cardinality. Let us consider the following notations:

$$\begin{cases} n : \text{Number of sets} \\ \top : \text{Initial greatest upperbound of all the sets} \\ c : \text{Cardinality of each set} \end{cases}$$

Each instance is simply generated as follows:

$$problem(n, \top, c) = \texttt{distinct}([s_1, \ldots, s_n]) \\ \forall s_i, \#s_i = c$$

We considered the following parameter values: $n \in \{10, 20, 40, 80\}$, $\#\top \in \{4, 6, 8, 10\}$ and $c \in \{3, 4, 5, 6\}$. The full table of results can be found in the appendix, where we consider all possible parameter combinations, and provide comparative results with the global versus local reasoning combined with the two labelling strategies introduced previously. An analysis of the table shows that:

1. The global reasoning allows us to detect *unsatisfiability* almost always before search or at its beginning (very few choice points) typically within a few milliseconds.
2. Once again the labelling strategy has little impact on the performance of the global reasoning algorithm, which is not the case for the local bound consistency approach which performs well only with a *l*exical strategy since pruning occurs only when sets become ground.
3. In most cases, the global inference rules allow for a significant reduction to the number of choice points and consequently failures, typically at least one order of magnitude when finding a valid configuration and several order of magnitude when dealing with unsatisfiable cases.

6 Conclusion and future work

A combinatorial problem can be modelled in different ways. The choice of the representation often seeks the best trade-off between "natural and generic formulation" and efficient solving. While set constraints offer a natural and concise representation of combinatorial configuration problems, alternative formulations are often preferred for efficiency reasons. In this article we have presented a first step towards improving set constraint solvers by introducing global constraint reasoning over set constraints.

We have introduced a set of conditional rules for a class of n-ary constraints. The key idea was to make powerful use of counting functions from combinatorial mathematics which apply to powersets having some given properties. To do so we modelled combinatorial configuration problems using finite sets constraints where sets range over set domains specified by sets intervals. Thus we benefited from a natural formulation of a mathematical problem within a powerset structure.

We have considered two main global reasoning for the n-ary constraints `atmost1` and `distinct` applied to sets of known cardinality. The preliminary results show that global reasoning allows us to detect failures very early in the search and to infer in a cost effective manner more pruning than local consistency. The global reasoning also proved to be reasonably independent from the search strategy, unlike local consistency.

Clearly more work is needed both from a theoretical and practical point of view. We believe that more pruning can be achieved at least for the `atmost1` constraint but at a much higher cost with respect to time and space complexity. However it would be very interesting to prove whether one reaches the optimum level of pruning or not for both constraints. With respect to the implementation, the complexity results clearly demonstrate that the time complexity would benefit tremendously from an incremental version of the conditional rules where one can access and modify the different intensional sets and cardinalities in constant time. Furthermore, the inference rules presented here assume a common and known cardinality for all the sets in the n-ary constraint. The rules could be generalised to allow for different cardinalities.

Note that combinatorial design problem are symmetry problems by definition. We haven't used any symmetry breaking strategy and still the global reasoning approach allowed us to reach solutions in reasonable time. However the overall efficiency would definitely benefit from symmetry breaking strategies applied to set based CSPs.

References

[BC94] N. Beldiceanu and E. Contejean. Introducing Global Constraints in CHIP. In Elsevier Science, editor, *Mathematical Computation Modelling*, volume 20(12), pages 97–123. Pergamon, 1994.

[Ber71] C. Berge. Principles of Combinatorics. In *Mathematics in Science and Engineering*, volume 72. Academic press edition, 1971.

[Ger97] C. Gervet. Interval propagation to reason about sets: Definition and implementation of a practical language. In E. Freuder, editor, *CONSTRAINTS journal*, volume 1(3), pages 191–244. 1997.

[GS00] I.P. Gent and B.M. Smith. Symmetry Breaking During Search. In *Proceedings of ECAI'2000*, pages 599–603, 2000.

[LR80] C.C. Lindner and A. Rosa. *Topics on Steiner Systems*, volume 7 of *Annals of Discrete Mathematics*. North Holland, 1980.

[Mul98] T. Muller. Solving set partitioning problems with constraint programming. *PAPPACT'98*, pages 313–332, 1998.

[Rég94] J.-C. Régin. A filtering algorithm for constraints of difference in csps. In *AAAI'94*, Seattle, USA, 1994.

Table 1. The numbers recorded for each test are the number of failures with the time taken to terminate (in millisecs) (a + indicates a solution was found, a - indicates one was not). The global reasoning results are in grey. An absence of time indicates that the timeout was reached.

Labelling strategy lexical

NumElements	Num Sets of cardinality 3: 10		20		40		80	
4	196 200ms-	2 40ms-	196 220ms-	2 80ms-	196 500ms-	2 40ms-	196 1420ms-	2 50ms-
6	45 50ms+	20 170ms+	190 150ms+	30 570ms+	39618 –	4 40ms-	22189 –	4 90ms-
8	45 40ms+	25 200ms+	190 160ms+	65 850ms+	780 890ms+	ms-	43261 –	6 100ms-
10	45 30ms+	31 150ms+	190 150ms+	76 830ms+	780 1000ms+	174 5590ms+	3160 8660ms+	327 34960ms+

NumElements	Num Sets of cardinality 4: 10		20		40		80	
4	1 10ms-	1 20ms-	1 10ms-	1 20ms-	1 0ms-	1 20ms-	1 0ms-	1 20ms-
6	45 50ms+	15 120ms+	64792 –	3 50ms-	46915 –	3 30ms-	27504 –	3 40ms-
8	45 40ms+	22 190ms+	190 160ms+	52 1000ms+	780 980ms+	112 6050ms+	107666 –	5 60ms-
10	45 30ms+	27 190ms+	190 160ms+	69 1120ms+	780 960ms+	156 6910ms+	3160 9060ms+	355 47300ms+

NumElements	Num Sets of cardinality 5: 10		20		40		80	
4	1 20ms-	1 20ms-	1 20ms-	1 10ms-	1 10ms-	1 20ms-	**1** **20**ms-	**1** **20**ms-
6	9786 4730ms-	2 20ms-	9786 8750ms-	2 20ms-	9786 20870ms-	2 20ms-	**9786** **62350**ms-	**2** **30**ms-
8	45 40ms+	20 190ms+	190 150ms+	45 1090ms+	780 930ms+	90 6680ms+	**74953** –	**4** **50**ms-
10	45 40ms+	25 240ms+	190 150ms+	65 1390ms+	780 970ms+	139 7980ms+	**3160** **8860**ms+	**300** **53640**ms+

NumElements	Num Sets of cardinality 6: 10		20		40		80	
4	1 10ms-	1 20ms-	1 10ms-	1 10ms-	1 10ms-	1 20ms-	1 10ms-	1 20ms-
6	1 20ms-	1 10ms-	1 20ms-	1 20ms-	1 20ms-	1 20ms-	1 10ms-	1 20ms-
8	45 40ms+	15 150ms+	190 120ms+	33 830ms+	110061 –	3 30ms-	51338 –	3 40ms-
10	45 40ms+	22 260ms+	190 150ms+	52 1370ms+	780 950ms+	117 8330ms+	3160 8710ms+	254 56580ms+

.

Labelling strategy smallest GLB

NumElements	Num Sets of cardinality 3: 10		20		40		80	
4	1116 690ms-	2 10ms-	6326 10000ms-	2 20ms-	15556 –	2 20ms-	11947 –	2 40ms-
6	25981 –	4 70ms+	24026 –	14 400ms+	19644 –	6 30ms-	14090 –	6 90ms-
8	25473 –	0 60ms+	23028 –	16 460ms+	18264 –	44 2720ms+	12981 –	10 120ms-
10	34597 –	0 90ms+	28571 –	5 410ms+	21542 –	56 3560ms+	14726 –	144 23050ms+

NumElements	Num Sets of cardinality 4: 10		20		40		80	
4	1 10ms-	1 10ms-	1 20ms-	1 20ms-	1 20ms-	1 20ms-	1 20ms-	1 20ms-
6	22845 –	3 90ms+	23154 –	4 30ms-	20566 –	4 20ms-	15559 –	4 60ms-
8	28927 –	1 80ms+	26948 –	11 520ms+	22619 –	41 3350ms+	16532 –	8 120ms-
10	38648 –	0 70ms+	33527 –	9 650ms+	26862 –	42 4440ms+	18731 –	157 35620ms+

NumElements	Num Sets of cardinality 5: 10		20		40		80	
4	1 40ms-	1 20ms-	1 0ms-	1 10ms-	1 0ms-	1 20ms-	1 0ms-	1 20ms-
6	31316 –	2 20ms-	27777 –	2 20ms-	26767 –	2 20ms-	20237 –	2 40ms-
8	43393 –	4 120ms+	39820 –	14 670ms+	33829 –	34 4300ms+	23619 –	6 70ms-
10	43812 –	0 90ms+	38762 –	16 840ms+	31975 –	44 5590ms+	22424 –	126 36180ms+

NumElements	Num Sets of cardinality 6: 10		20		40		80	
4	1 20ms-	1 20ms-	1 10ms-	1 20ms-	1 20ms-	1 0ms-	1 30ms-	1 0ms-
6	1 10ms-	1 0ms-	1 10ms-	1 0ms-	1 20ms-	1 0ms-	1 20ms-	1 0ms-
8	35013 –	3 160ms+	35738 –	9 870ms+	32638 –	4 30ms-	23778 –	4 60ms-
10	44682 –	1 160ms+	42932 –	11 1010ms+	34948 –	41 6180ms+	25195 –	113 45890ms+