A Fast Algorithm for Generalized Arc Consistency of the Alldifferent Constraint

Xizhe Zhang[1,2], **Qian Li**[1] and **Weixiong Zhang**[3,4]
[1]School of Computer Science and Engineering, Northeastern University, Shenyang, Liaoning, China
[2]Joint Laboratory of Artificial Intelligence and Precision Medicine of China Medical University and Northeastern University, Shenyang, Liaoning, China
[3]College of Math and Computer Science, Institute for Systems Biology, Jianghan University, Wuhan, China
[4]Department of Computer Science and Engineering, Washington University, Saint Louis, Missouri, USA
zhangxizhe@mail.neu.edu.cn

Abstract

The *alldifferent* constraint is an essential ingredient of most Constraints Satisfaction Problems (CSPs). It has been known that the generalized arc consistency (GAC) of *alldifferent* constraints can be reduced to the maximum matching problem in a value graph. The redundant edges, which do not appear in any maximum matching of the value graph, can and should be removed from the graph. The existing methods attempt to identify these redundant edges by computing the strongly connected components after finding a maximum matching for the graph. Here, we present a novel theorem for identification of the redundant edges. We show that some of the redundant edges can be immediately detected after finding a maximum matching. Based on this theoretical result, we present an efficient algorithm for processing *alldifferent* constraints. Experimental results on real problems show that our new algorithm significantly outperforms the-state-of-art approaches.

1 Introduction

Constraint Programming is a powerful tool for problem solving and has been widely used in various real-world applications. Constraint Satisfaction Problem (CSP) defines a set of variables whose values must satisfy some specified constraints. One of the most useful and important constraints is the alldifferent constraint [Lauriere 1978], which requires that all variables of the constraint must have different values. The alldifferent constraint can be found in wide varieties of combinational problems [Wallace 1996], including various puzzles, graph coloring, and assignment problems.

A typical approach to solve a CSP is to search from all possible variable values. To accelerate this search process, various consistency techniques have been introduced to remove values from the domain of a variable which does not belong to any solution to the problem. A classic filtering algorithm for **G**eneralized **A**rc **C**onsistency (***GAC***) for *alldifferent* constraints is proposed by Régin [Régin 1994]. The algorithm utilizes a theorem of C. Berge [Berge and Minieka 1973] from graph theory and prunes redundant edges that do not appear in any maximum matching of the value graph of *alldifferent* constraints. For weaker forms of the consistency of *alldifferent* constraints, Leconte [Leconte 1996] provides an algorithm for the range consistency based on identifying Hall intervals. Puget [Puget 1998] proposes an algorithm for the bounds consistency, which is weaker than the range consistency. To our knowledge, Régin's algorithm is still the-state-of-art filtering method for ***GAC*** of the *alldifferent* constraint. Gent [Gent, Miguel et al. 2008] discuss several implementation details of Régin's algorithm, especially the computation of the **S**trongly **C**onnected **C**omponents (***SCC***) of the residual graph. A survey for the *alldifferent* constraint can be found in [van Hoeve 2001]. Recently, as a powerful tool, *alldifferent* constraints are extensively used to solve difficult constraint optimization problems, such as subgraph isomorphism [Solnon 2010] and constraint clustering [Duong and Vrain 2017].

In this paper, we developed a simple and fast algorithm for the *alldifferent* constraint. Our new algorithm is based a novel theorem for identifying redundant edges of the value graph of *alldifferent* constraints. We classified the redundant edges into two types: one type of redundant edges can be obtained immediately by finding a maximum matching, and the second type of redundant edges can be found by computing ***SCC****s* in a small sub-graph. Based on this theorem, we designed an efficient algorithm for the *alldifferent* constraint. The major improvement of our algorithm comes from identification of type 1 redundant edges, which allow us to remove a substantial number of redundant edges without any additional computation. Compared with the-state-of-art approach, our new algorithm does not need to construct a residual graph, which has more edges than the original value graph. Therefore, the ***SCC***s computation in our algorithm is also faster than the previous algorithm. We evaluated the performance of our new algorithm on many benchmark instances by using Minion 1.8 software [Gent, Jefferson et al.],

and the result showed that our algorithm outperformed Régin's algorithm on all problem instances.

The paper is organized as follows. We present in Section 2 the definitions related to constraint satisfaction problems and graph theory. We present and prove in Section 3 the novel theorem for identifying the redundant edges of a value graph, and present a fast algorithm for the *alldifferent* constraint. We evaluate the performance of our algorithm on several real problems in Section 4, and conclude with discussions in Section 5.

2 Background and Preliminaries

Constraint programming. A constraint satisfaction problem (***CSP***) is a triple (X, D, C), where X is a set of variables, $\{x_1, x_2, \ldots, x_n\}$, D is a set of domains $\{D_1, D_2, \ldots, D_n\}$, where D_i is the set of possible values for variable x_i, and C is a set of constraints between variables. A constraint $c \in C$ is defined as a subset of the Cartesian product of the domains of the variables that are in C. A solution to a CSP (X, D, C) is a set of values $(d_1,\ldots,d_n) \in D_1\times\cdots\times D_n$, where for every constraint $c \in C$ on the variables $x_{i1},\ldots,x_{im}$, we have $(d_{i1}, \ldots, d_{im}) \in c$. A constraint is generalized arc consistent (***GAC***) *iff* every value of the variables can be extended to all the other variables of the constraint in such a way the constraint is satisfied.

An *alldifferent* constraint $c(x_1,\ldots,x_n)$ is a constraint that specifies that $x_i \neq x_j$ for any $i < j$. For an *alldifferent* constraint c, a bipartite graph $B(c) = (X_c, D_c, E)$ is called a value graph of c, *where* $(x_i, d) \in E$ *iff* $d \in D_i$. To achieve the ***GAC*** on an *alldifferent* constraint, we need to introduce some concepts from graph theory, especially about maximum matching.

Graph Theory. Consider a bipartite graph $B(U,V,E)$, in which U, V are two disjoint sets of nodes and every edge in E connects a node in U to one in V. A *matching* is a set of edges that share no common node. A node is called a *matched node* if it is connected to an edge in the matching, or a *free node*, otherwise. A matching with the maximum number of edges is called a *maximum matching*. An *alternating path* is a path whose edges are alternate in and out of the matching. An *augmenting path* is an alternating path whose two end nodes are free nodes.

A bipartite graph may have more than one maximum matching, so that maximum matching is usually not unique. There may exist many maximum matchings with the same size for a graph. An *allowed edge* is an edge belonging to some, but not all, of maximum matchings. A *redundant edge* is an edge that does not appear in any maximum matchings. Similarly, an *allowed node* is a node covered by some, but not all, of maximum matchings.

Régin's filtering Algorithm. An *alldifferent* constraint is ***GAC*** *iff* every edge of its value graph belong to some matchings that cover X_c in $B(c)$ [Régin 1994]. Therefore, to achieve

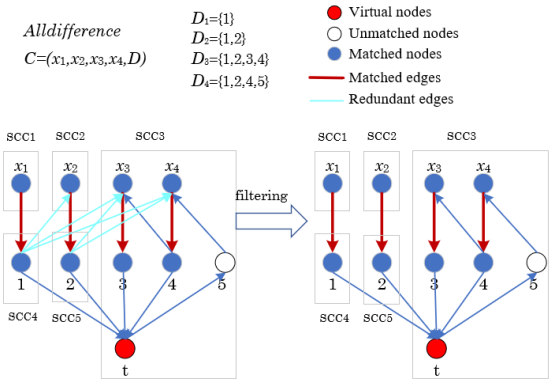

Figure 1: Illustration of Régin's filtering algorithm [Régin 1994]. The algorithm first computes a maximum matching, then constructs the directed residual graph by adding one node and $|D_c|$ edges to original graph, and computes *SCC* of the residual graph. The unmatched edges between independent *SCCs* are redundant.

GAC, we need to remove the redundant edges of its value graph. Régin [Régin 1994] present a filtering algorithm based on the following property:

Property 1 (Berge 1970) *An edge is allowed, iff, for an arbitrary maximum matching M, it belongs to either an even alternating path begins at a free node or an even alternating cycle.*

Therefore, to identify all of the redundant edges of the value graph, the algorithm first computes a maximum matching, and then finds all alternating paths beginning at free nodes and all even alternating cycle based on *SCCs*. The last two steps can be combined by finding ***SCCs*** in a directed residual graph. The residual graph is constructed by adding one virtual node and $|D_c|$ edges to the original value graph. Therefore, the residual graph is always larger than value graph. The unmatched edges between independent ***SCCs*** of the residual graph are redundant edges. An example is shown in Figure.1.

3 Identify Redundant Edges

For an *alldifferent* constraint c, the variable-value pairs corresponding to the edges that never appear in any maximum matching should be pruned. Therefore, in the rest of the paper, we focus on how to efficiently identify all of the redundant edges of a value graph.

Consider an *alldifferent* constraint $c(X_c, D_c)$ and its value graph $B(c) = (X_c, D_c, E)$. Let A be the set of allowed nodes of $B(c)$, and $\Gamma(A)$ be the set of the neighbor nodes of A. It is obvious that if there exists a solution to the constraint, all nodes of X_c must be matched. Therefore, the allowed nodes can only be found in set D_c. The node set of the value graph can be divided into four sets: A, $\Gamma(A)$, D_c-A, X_c-$\Gamma(A)$. The allowed nodes can be easily identified by our previous work [Zhang, Han et al. 2017], which was originally used to identify the input node of a network.

Property 2 [Zhang, Han et al. 2017] *A node is allowed, iff*

for an arbitrary maximum matching M, it can be reached by an even alternating path that begins at a free node.

Based on Property 2 and using our node set partition, we now present a new theorem for identifying redundant edges:

Theorem: For an arbitrary maximum matching, an unmatched edge e_{mn} is redundant, *iff*

1. $m \in \Gamma(A)$ and $n \in D_c\text{-}A$, or
2. $m \in X_c\text{-}\Gamma(A)$, $n \in D_c\text{-}A$ and e_{mn} does not belong to any alternating cycle.

Proof: Based on the definitions of A, $\Gamma(A)$, $D_c\text{-}A$ and $X_c\text{-}\Gamma(A)$, the edges of a value graph can be divided into three sets: $E(\Gamma(A),A)$, $E(\Gamma(A),D_c\text{-}A)$ and $E(X_c\text{-}\Gamma(A),D_c\text{-}A)$ (Figure.2). Note that based on the definition of $\Gamma(A)$, all neighbor nodes of A should be in $\Gamma(A)$. Therefore, there is no edge between A and $X_c\text{-}\Gamma(A)$.

We first prove that all edges of $E(\Gamma(A), A)$ are allowed edges. Consider an unmatched edge $e_{ba} \notin M$, where $a \in A$ and $b \in \Gamma(A)$. Because a is an allowed node, based on property 2, there must exist an even alternating path P connecting node a and a free node. Consider the path $P+e_{ba}$, if we swap the matched and unmatched edges of the path, we have a new maximum matching M', where $e_{ba} \in M$. Therefore, all edges of $E(\Gamma(A), A)$ are allowed edges and can be in the new maximum matching M'.

Next, we prove that all edges of $E(\Gamma(A), D_c\text{-}A)$ are redundant. Because $\Gamma(A)$ is part of the set of variables, any node of $\Gamma(A)$ should be matched. Otherwise, there is no solution to the constraint. Suppose e_{mn} is an allowed edge, where $m \in \Gamma(A)$ and $n \in D_c\text{-}A$. Because $\Gamma(A)$ is the neighbor set of A and any node of A is allowed, there must exist an alternating path P starting at a free node connected to node m. Therefore, the path $P+e_{mn}$ is an alternating path. Based on Property 2, node n is an allowed node. That contradicts with the definition of node n. Therefore, all edges of $E(\Gamma(A), D_c\text{-}A)$ are redundant edges.

Finally, we prove the edges of $E(X_c\text{-}\Gamma(A), D_c\text{-}A)$ are redundant edges *iff* they are not matched and do not belong to any alternating cycle. Note that it is a corollary of Property 1. Therefore, the proof is completed.

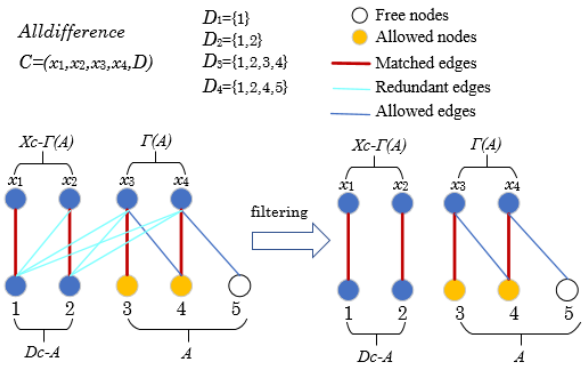

Figure 2: The value graph of an *alldifferent* constraint and its redundant edges.

Figure 2 gives a simple example of above theorem. After finding a maximum matching (red edges) of the value graph, node 5 is the only free node. The set of allowed nodes are A={3,4,5} because the nodes 3 and 4 can be reached by alternating path start from node 5. The neighbor set of A is $\Gamma(A)$={x_3,x_4}. Based on our theorem, the edges between set $\Gamma(A)$ and $D_c\text{-}A$ can be removed from the value graph. Therefore, we only need to find the unmatched edges between sets {x_1, x_2} and {1,2} and not in any alternating cycle, which is the edge e(x_2,1).

This theorem offers us an efficient way to find redundant edges in a bipartite graph. Based on this theorem, the edges between node sets $\Gamma(A)$ and $D_c\text{-}A$ are denoted by type 1 redundant edges, and the other redundant edges by type 2 redundant edges. The type 1 redundant edges can be easily obtained after finding the set of allowed nodes. The allowed nodes can be found by finding a maximum matching based on Property 2. The basic idea of a maximum matching algorithm, such as Hopcroft–Karp algorithm [Hopcroft and Karp 1973] or Hungarian Algorithm [Kuhn 1955], is to iteratively find all augmenting paths corresponding to the matching M at hand, and then to derive a larger matching M'. A maximum matching is obtained when no augmenting path can be found. The last step of the algorithm is exactly to find all alternating paths starting at the free nodes of the maximum matching. Therefore, all allowed nodes and their neighbors can be obtained in the last step of a maximum matching algorithm, which provides the first part of redundant edges. For the rest of redundant edges, we only need to find ***SCCs*** in a smaller sub-graph $B'(X_c\text{-}\Gamma(A), D_c\text{-}A, E')$, where $E' \subset E$ is the edge\ set connecting $X_c\text{-}\Gamma(A)$ and $D_c\text{-}A$.

The above idea and steps are formulated in Algorithm 1 for filtering value graph B of *alldifferent* constraints.

ALGORITHM1: Fast filtering *alldifferent* constraint

1. **Input**: Value graph $B(c) = (X_c, D_c, E)$, initial matching M;
2. **Repeat**
3. Find all alternating paths AP from all free node based on current matching M;
4. Put the nodes in D_c of AP into set A, and their neighbor nodes in X_c into set $\Gamma(A)$;
5. **If** AP contains the augmenting paths **then**
6. expand the augmenting paths and obtain a new matching M';
7. Let $M=M'$; clear set A and $\Gamma(A)$;
8. **If** $size(M)=size(X_c)$ **then**
9. prune all edges between set $\Gamma(A)$ and set $D_c\text{-}A$;
10. **Until** no augmenting path is found;
11. **If** $size(M)<size(X_c)$ **return** false;

// Above is the first parts of the redundant edges of value graph

12. Find all strong connected component (***SCC***) of bipartite graph $B'(X_c\text{-}\Gamma(A), D_c\text{-}A, E')$;

13. Prune all edges that are unmatched and connect nodes between two ***SCC***s;

The first part of our algorithm is to find a maximum matching, and this can be done in $O(|X_c|^{0.5}|E|)$ by the Hopcroft–Karp algorithm. The second part is to find ***SCC*** of the bipartite graph $B'(X_c\text{-}\Gamma(A), D_c\text{-}A, E')$, and this can be accomplished in $O(|X_c\text{-}\Gamma(A)|+|D_c\text{-}A|+|E'|)$ by the Tarjan algorithm [Tarjan 1972]. Although our algorithm has the same worst-case complexity as the previous works, it has a better performance in practice. This is because many redundant edges can be immediately removed after finding a maximum matching and we only need to find ***SCCs*** in a smaller graph $B'(X_c\text{-}\Gamma(A), D_c\text{-}A, E')$ rather than the original value graph.

The first part of Algorithm 1 (steps 1-10) is basically to find a maximum matching. Therefore, it can be integrated with many maximum matching algorithms, such as the Hopcroft–Karp algorithm or the Hungarian Algorithm. Furthermore, the improvement of our algorithm is the identification of type 1 redundant edges in the matching process, therefore, it can be combined with many optimization technics for *alldifferent* constraints, such as incremental matching, domain counting, priority queue, staged propagation or computing *SCC*s independently [Gent, Miguel et al. 2008].

4 Experimental Results

We evaluated the performance of our new algorithm. We first analyzed the fraction of type 1 redundant edges among all redundant edges. We then compared the performance of our algorithm with the-state-of-art approaches on a large collection of benchmark instances for *alldifferent* constraints. Our algorithm was implemented based on Minion constraint solver 1.8 [Gent, Jefferson et al.]. All experiments were run on a Windows 7 workstation with a quad-core Intel i7-3770 processor of 3.9 GHz and 32GB DDR3 1600MHz RAM.

In our experiment, we first generated a series of synthetic bipartite graphs by using the scale-free network model of [Shen-Orr, Milo et al. 2002] and the *ER* random network model of [Bollobás 2013], where we set $|X_c|=|D_c|=1000$. Type 1 redundant edges can be obtained right after maximum matching. Therefore, our algorithm will be more efficient if there are more type 1 edges. When the average node degree increases, most of the redundant edges are type 1 and the size of $B'(X_c\text{-}\Gamma(A), D_c\text{-}A, E')$ is relatively small (Figure 3). This means that after finding a maximum matching, most of the redundant edges can be found, and the remaining redundant edges can be obtained in a small value graph $B'(X_c\text{-}\Gamma(A), D_c\text{-}A, E')$. It will greatly increase the efficiency of our algorithm in practice. We also assessed the fraction of type 1 redundant edges in some real problems (Figure 4). We counted the total number of redundant edges during the searching process and computed the fraction of type 1 and type 2 redundant edges. It is evident that many problems also have a large portion of type 1 redundant edges.

Next, we implemented our new algorithm by using the Minion software, Version 1.8 [Gent, Jefferson et al.]. This software already has an implementation of Régin's [Régin 1994] Filtering algorithm, and other optimization techniques, such as incremental matching [Régin 1994], BFS matching [Cormen 2009] and staged propagation [Schulte and Stuckey 2004]. For the implementation of our algorithm, we use Hopcroft–Karp algorithm to obtain maximum matching and Tarjan algorithm to compute ***SCC*s**. We also use incremental matching technique, which is same as [Régin 1994]. We compared our algorithm with the following implementations: 1) Régin's Filtering algorithm with incremental matching (**IncMatch**); 2) IncMatch using the FF-BFS matching algorithm instead of Hopcroft-Karp algorithm (**IncMatch-BFS**); 3) IncMatch-BFS with staged propagation (**IncMatch-BFS-Staged**). The searching strategy we used to solve the problems is the depth-first chronological backtracking. The benchmark instances were chosen from [Gent, Miguel et al. 2008], including Langford's number problem (prob024 in CSPLib [Gent and Walsh 1999]), golomb ruler problem (prob006 in CSPLib), balanced quasigroup with holes (QWH) [Kautz, Ruan et al. 2001], quasigroup existence (prob003 in CSPLib), social golfers (prob010 in CSPLib), graceful graphs (prob053 in CSPLib), N-Queens (prob054 in CSPLib) and sports scheduling.

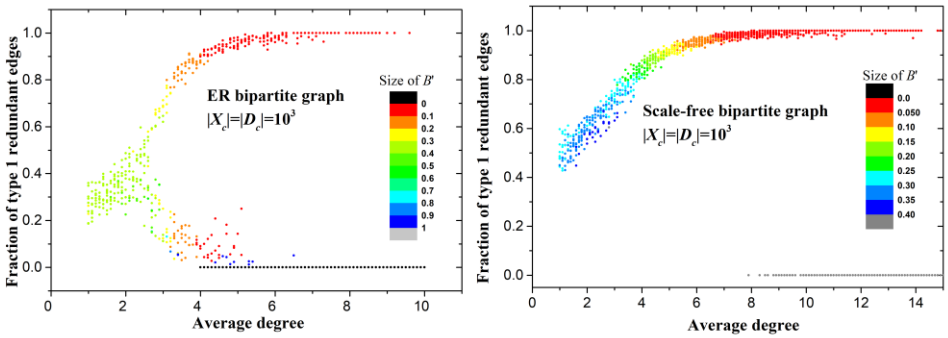

Figure 3: Fraction of type 1 redundant edges in the ER and Scale-free bipartite graphs.

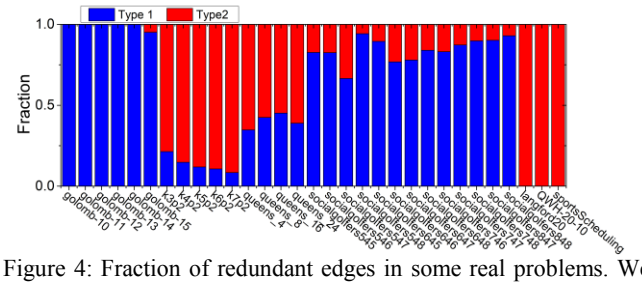

Figure 4: Fraction of redundant edges in some real problems. We counted the total number of redundant edges during searching process.

We compared our algorithm with the three existing methods mentioned above on these instances of constraint satisfaction problems. In the experiments, we limited time to 4 hours and nodes to 10^6. For those problems which the solution can be found within the time limit, we compared the raw runtime (Figure 5 and Table 1). For those problems which exceed the time limit, we counted the number of nodes searched per second (Figure 6). As shown in Table 1 and Figure 5-6, our algorithm is never slower than the other algorithm compared and can typically run 1-6 times faster than these methods. To reiterate, the gain on efficiency for our algorithm comes from the identifying and removing type 1 constraints with little computation. On those problems that do not have type 1 redundant edges, our algorithm also has a better performance because it finds ***SCC***s in the value graphs rather than the residual graphs, which have more edges than the original value graphs.

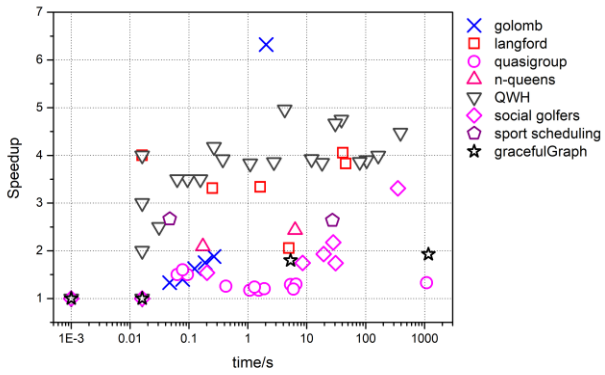

Figure 5: Speedups of our algorithm over the IncMatch-BFS-staged method. The solution of these instances can be found within the time limit, so we compared the raw runtime (seconds).

5 Conclusion

We developed a fast filtering algorithm for realization of generalized arc consistency for the *alldifferent* constraint. We presented a novel theorem for identifying redundant edges in a constraint graph which need to be removed. We showed that our algorithm significantly outperformed the best GAC algorithms on all of benchmark problems that we tested, significantly reduced computation time on these real problems. Our algorithm can be used to improve the performance of solving constraint satisfaction problems that contain *alldifferent* constraints.

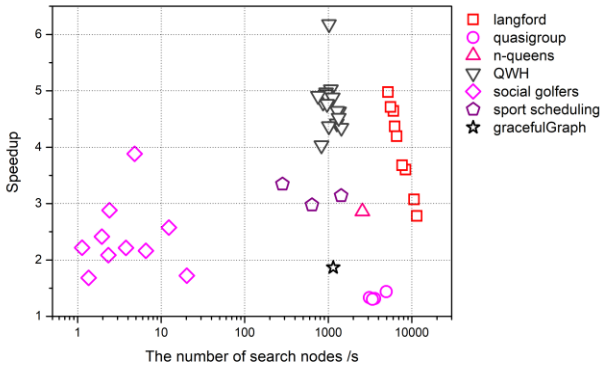

Figure 6: Speedups of our algorithm over the IncMatch-BFS-staged method. The solution of these instances cannot be found within the time limit, so we compared the number of nodes searched per second.

Problem	Our	IncMatch	IncMatch-BFS	IncMatch-BFS-staged
golomb-10-200	0.047	0.063	0.063	0.063
golomb-11-200	0.078	0.109	0.125	0.109
golomb-12-200	0.125	0.203	0.203	0.203
langford10	5.016	10.094	10.078	10.188
langford19	46.203	174.047	171.656	175.953
langford20	41.188	160.906	161.734	166.281
qg3nonidempotent9	1.875	2.281	2.344	2.297
qg4idempotent9	6.516	8.469	8.734	8.563
qg4nonidempotent8	1.313	1.563	1.609	1.609
qg4nonidempotent9	0.422	0.516	0.547	0.531
queens-16	0.172	0.359	0.359	0.359
queens-24	6.313	15.391	15.703	15.438
QWH-25-1	18.281	68.781	69.188	70.078
QWH-25-9	79.109	299.969	301.547	306.109
QWH-30-2	30.563	140.125	140.828	142.500
QWH-30-5	389.594	1723.09	1722.92	1743.61
socialgolfers547	31.641	62.266	62.125	61.547
socialgolfers647	276.141	1171.92	1166.84	1172
sportsScheduling8	0.063	0.125	0.109	0.125
sportsScheduling10	27.078	70.344	69.922	71.891
graceful graphs K5×P2	5.359	9.484	9.453	9.547
graceful graphs K6×P2	1178.39	2230.48	2221.98	2248.06

Table 1: Comparition of time to first solution (second)

Acknowledgments

This research was supported by the Natural Science Foundation of China under grant number 91546110, and China Scholarship Council under grant number 201606085011.

References

[Berge, 1973]Berge Claude. *Graphs and hypergraphs*, American Elsevier Publishing Company, New York, 1973.

[Bollobás, 2013]Béla Bollobás. *Modern graph theory*, Springer Science & Business Media, 2013.

[Cormen *et al.,* 2009]Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*, MIT Press, Cambridge, Massachusetts, 2009.

[Duong *et al.,* 2017]Thi-Bich-Hanh Dao, Khanh-Chuong Duong, and Christel Vrain. *Constrained clustering by constraint programming*. Artificial Intelligence 244:70-94, 2017.

[Gent *et al.,* 2006]Ian P. Gent, Christopher Jefferson, and Ian Miguel. *MINION: A Fast, Scalable, Constraint Solver*,(slides) in Proceedings of the 17th European Conference on Artificial Intelligence (ECAI), 2006.

[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.

[Gent *et al.,* 1999]Ian P. Gent and Toby Walsh. *CSPLib: a benchmark library for constraints*. International Conference on Principles and Practice of Constraint Programming, Springer, 1713:480-481, 1999.

[Hopcroft *et al.,* 1973]John E. Hopcroft and Richard M. Karp. *An n^5/2 algorithm for maximum matchings in bipartite graphs*. SIAM Journal on Computing 2(4): 225-231, 1973.

[Kautz *et al.,* 2001]Kautz Henry, Yongshao Ruan, Dimitris Achlioptas, Carla Gomes, Bart Selman, and Mark Stickel. *Balance and filtering in structured satisfiable problems*. Electronic Notes in Discrete Mathematics, 9:2-18, 2001.

[Kuhn, 1955]Harold W. Kuhn. T*he Hungarian method for the assignment problem*. Naval Research Logistics (NRL) 2(1-2): 83-97, 1955.

[Lauriere, 1978]Jena-Lonis Lauriere. *A language and a program for stating and solving combinatorial problems*. Artificial intelligence 10(1): 29-127, 1978.

[Leconte, 1996]M. Leconte. *A bounds-based reduction scheme for constraints of difference*. Proceedings of the Constraint-96 International Workshop on Constraint-Based Reasoning, 1996.

[Puget, 1998]Jean-Francois Puget. *A fast algorithm for the bound consistency of alldiff constraints.* Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, Iaai 98, July 26-30, 1998, Madison, Wisconsin, Usa DBLP, pages 359-366, 1998.

[Régin,1994]Jean-Charles Régin. *A filtering algorithm for constraints of difference in CSPs*. Twelfth National Conference on Artificial Intelligence American Association for Artificial Intelligence, pages 362-367, 1994.

[Schulte *et al.,* 2004]Christian Schulte and Peter J. Stuckey. *Speeding up constraint propagation*. Principles and Practice of Constraint Programming - CP 2004, International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings DBLP, pages 619-633, 2004.

[Shenorr *et al.,* 2002] Shai S. Shen-Orr, Ron Milo, Shmoolik Mangan, and Uri Alon. *Network motifs in the transcriptional regulation network of Escherichia coli*. Nature genetics 31(1): 64, 2002.

[Solnon, 2010]Christine Solnon. *Alldifferent-based filtering for subgraph isomorphism*. Artificial Intelligence 174(12-13): 850-864, 2010.

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

[Hoeve, 2001]Willem-Jan van Hoeve. *The alldifferent constraint: A survey*. Computer Science, 2001.

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

[Zhang *et al.,* 2017]Xizhe Zhang, Jianfei Han, and Weixiong Zhang. *An efficient algorithm for finding all possible input nodes for controlling complex networks*. Scientific Reports 7(1): 10677, 2017.