Early and Efficient Identification of Useless Constraint Propagation for Alldifferent Constraints

Xizhe Zhang[1,3], **Jian Gao**[2], **Yizhi Lv**[3] and **Weixiong Zhang**[4]
[1]School of Biomedical Engineering and Informatics, Nanjing Medical University, Nanjing, China
[2]College of Information Science and Technology, Dalian Maritime University, Dalian, China
[3]School of Computer Science and Engineering, Northeastern University, Shenyang, China
[4]Department of Computer Science and Engineering, Washington University, St. Louis, MO, USA
zhangxizhe@gmail.com; gaojian@dlmu.edu.cn; weixiong.zhang@wustl.edu

Abstract

Constraint propagation and backtracking are two basic techniques for solving constraint satisfaction problems (CSPs). During the search for a solution, the variable and value pairs that do not belong to any solution can be discarded by constraint propagation to ensure generalized arc consistency to avoid the fruitless search. However, constraint propagation is frequently invoked with little effect on many CSPs. Much effort has been devoted to predicting when to invoke constraint propagation for solving a CSP. However, no effective approach has been developed for the *alldifferent* constraint. Here we present a novel theorem for identifying the edges in a value graph of the *alldifferent* constraint whose removal can lead to useless constraint propagation. We prove that if an alternating cycle exists for a prospectively removable edge that represents a variable-value assignment, the edge (and the assignment) can be discarded without constraint propagation. Based on this theorem, we developed a novel optimization technique for early detection of useless constraint propagation which can be incorporated in any existing algorithm for the *alldifferent* constraint. Our implementation of the new method achieved speedup by a factor of 1 to 5 over the state-of-art approaches on 93 benchmark problem instances in 8 domains. Besides, the new algorithm is scalable well and runs increasingly faster than the existing methods on larger problems.

1 Introduction

Constraint programing [Rossi et al., 2006] is a powerful technique for solving difficult combinatorial problems in wide range applications of computer science and beyond. **C**onstraint **S**atisfaction **P**roblem (CSP) defines a set of variables whose values must satisfy some specified constraints. The *alldifferent* constraint [Lauriere, 1978] is a type of global constraint where all variables must have different values. The *alldifferent* constraint is one of the most important and difficult types of constraints and appears in many applications, such as puzzles, graph coloring, scheduling, and recommendation or matchmaking [Hoeve, 2001].

Backtracking [Golomb and Baumert, 1965] and constraint propagation [Apt, 1998] are the two most used techniques for solving CSPs. Searching for a CSP solution amounts to the traversal of all possible variable assignments, resulting in an exponential time complexity in the worst case. Constraint propagation is introduced to accelerate the search process by pruning the useless branches of the search space or equivalently removing the inconsistent variable-value assignments that do not appear in any solution to the problem. If there is no inconsistent value at a certain stage of the search, we say that the constraint propagator has achieved Generalized Arc Consistency (GAC) [Hentenryck et al. 1992] for the *alldifferent* constraint. A constraint propagation algorithm (or constraint propagator, filtering algorithm) is often used to filter out inconsistent variable-value assignments – the sooner the inconsistent assignments can be detected, the more effective the constraint propagator is.

Several constraint propagators have been developed to achieve GAC for the *alldifferent* constraint. Régin first formulated the constraint propagation for the *alldifferent* constraint as the maximum matching problem in a bipartite graph [Régin, 1994]. Régin proved an insightful theorem showing that the edges not belonging to any maximum matching of the bipartite graph must be inconsistent variable-value assignments. These edges can be identified by computing a maximum matching and Strongly Connected Components (SCCs) of the variable-value graph. This theorem forms the basis of nearly all existing constraint propagation algorithms for GAC of the *alldifferent* constraint. As maximum matching can be incrementally computed, the overall computation of the Régin algorithm is dominated by the cost for computing SCCs. Gent et.al [Gent et al., 2008] presented an SCC optimization technique to extend the Régin algorithm. It splits individual SCCs during the search process and only computes the SCCs that contain removed variable-value pairs. In our previous work [Zhang et al., 2018] we presented a new graph theorem that can be used to remove some edges after computing a maximum matching which can accelerate the propagation process. All of these existing methods have the same worst-case complexity because they all need to compute a maximum matching and then SCCs.

Despite the efficiency of the existing constraint propagators, a rarely addressed but critically important problem is that the constraint propagator is frequently invoked but most of the time removes no inconsistent value at all for solving CSPs. It has been reported that a substantial amount of constraint propagation is useless and removes no inconsistent value for the *alldifferent* constraint [Katriel, 2006; Gent et al., 2006; Boisberranger et al., 2013]. We will also present similar and more detailed results on some benchmark instances in Figure 4 in the Experiments section of this paper. Therefore, a critical question is when to invoke constraint propagator for the *alldifferent* constraint during the process of solving a CSP. As constraint propagation contributes a bulk of the overall computation cost, reducing the number of useless constraint propagation can speed up the process of solving a CSP. The sooner a useless constraint propagation can be detected, the more efficient a CSP solver algorithm will be.

However, it is technically challenging to determine if a constraint propagation is effective. Katriel [Katriel, 2006] carried out a theoretical analysis regarding when to enforce GAC under the assumption that edges are randomly removed from the value graph. She showed that very few edges were important and removing them would help reduce the domain sizes of some variables. She suggested postponing constraint propagation until a certain number of important edges were removed. However, as important edges are not evenly distributed in the search space, missing a useful propagator invocation early in the search will increase the overall computation cost. Gent et.al [Gent et al., 2008] extended Katriel's idea and proposed a practical approach of dynamic triggers to avoid useless propagation. Nevertheless, this approach failed to improve performance. Gent et.al [Gent et al., 2008] reported that dynamic triggers usually slowed down the overall running time for the *alldifferent* constraint related problems. Boisberranger [Boisberranger et al., 2013] presented a probabilistic approach by calculating the invoking probability for the propagator to enforce Bound Consistency (BC), which is a much weaker form of GAC for the *alldifferent* constraint.

In short, to our best knowledge, there is currently no practical method for the *alldifferent* constraint to accurately and efficiently determine when constraint propagator should be invoked during the process of solving a CSP. Ideally, all useless constraint propagations must be avoided and whenever a constraint propagator is called up, it must help reduce the search space by removing some inconsistent values. The little progress made beyond Régin's seminal theoretical work inspired us to reason that Régin's insightful theorem may have already been exploited to its limit by the existing methods. The existing methods for *alldifferent* propagation are already very fast, having nearly linear running time so that any method for deciding when to invoke a constraint propagator must be accurate and efficient and failure to do so will otherwise slow down the search process. However, it is very difficult to determine whether the propagator should be called upon without actually running the propagator.

In this paper, we present a novel theorem and develop a companion technique for early detection of useless constraint propagation. The main idea is to let the propagator itself decide whether the current constraint propagation is useful or not and terminate the propagation process as soon as it determines the current propagation is not helpful. This idea stems from a novel theorem for identifying unimportant edges in a value graph for the *alldifferent* constraint. We rigorously prove that an edge is not important if there exists an alternating cycle in the graph containing the end nodes of the edge after removing it. To our knowledge, this is the first accurate and efficient approach beyond the Régin's theorem for early determination if a constraint propagation is effective. Exploiting this theorem, we present a novel *alldifferent* propagator that can efficiently identify and promptly stop useless constraint propagations. We implement our algorithm in the state-of-the-art CSP solver by Gent et al. [Gent et al., 2006] and achieve speedup by a factor of 1~5 over the state-of-the-art approach. Furthermore, our propagator performs better on large problems, showing its favorable scalability.

2 Preliminaries and the Existing Methods

Constraint programming. A constraint satisfaction problem (CSP) is defined as a triple (X, D, C), where X is a set of variables $\{x_1, x_2, ..., x_n\}$, D is a set of domains $\{D_1, D_2, ..., D_n\}$ where each variable $x_i \in X$ can take its values in the finite domain $D_i \in D$, C is a set of constraints, which specify all of the allowable value-to-variable assignments. A solution to a CSP $P=(X, D, C)$ is a set of assignments of values $(d_1,...,d_n) \in D_1 \times \cdots \times D_n$ to variables such that for every constraint $c \in C$ on the variables $x_{i1},...,x_{im}$, there exist $(d_{i1}, ..., d_{im}) \in c$. A constraint is generalized arc consistent (GAC) *iff* every value of a variable can be extended to all the other variables of the constraint maintaining satisfiability of the constraint. A global constraint can capture a relation between a non-fixed number of variables. It is a useful tool in modeling real-world CSPs. Among various global constraints, the *alldifferent* constraint is one of the most important. An *alldifferent* constraint c specifies that each pair of variables involved in c cannot take the same value.

Graph Theory. Enforcing GAC on an *alldifferent* constraint requires finding a maximum matching on a bipartite value graph of the *alldifferent* constraint. Given an *alldifferent* constraint c, a value graph of c is a bipartite graph $B(c) = (X_c, D_c, E)$, *where* X_c is a set of variables involved in c, D_c is a value domain, and $(x_i, d) \in E$ *iff* $d \in D_i$.

A *matching* is a set of edges that share no common node. A *maximum matching* is matching with the maximum number of edges. A *matched node* is a node that connects to an edge in a given matching, or a *free node*, otherwise. An *alternating path* is a path whose edges alternate in and out of the matching. An *augmenting path* is an alternating path whose two end nodes are free. It is noted that maximum matching is typically not unique for a bipartite graph; there may exist multiple maximum matchings for a bipartite graph. An *allowed edge* is an edge belonging to some, but not all, of maximum matchings. Similarly, an *allowed node* is a node covered by some, but not all, of maximum matchings. A *redundant edge* is an edge that does not appear in any maximum matching. An allowed edge can be determined by an early theorem by Berge [Berge, 1973]:

The Berge Theorem. *An edge is allowed, iff, for an arbitrary maximum matching M, it belongs to either an even alternating path that begins at a free node or an even alternating cycle.*

Enforcing the GAC for *alldifferent*. The first algorithm enforcing GAC for the *alldifferent* constraint was proposed by Régin based on the Berge Theorem. In this algorithm, the redundant edges are removed from the value graph to achieve GAC. The algorithm first computes a maximum matching of the value graph and then constructs a directed version of the value graph based on the maximum matching. The redundant edges can be identified by computing the alternating paths beginning at free nodes and the SCCs of the directed bipartite graph. Figure 1 shows a simple example. The unmatched edges between SCCs are the redundant edges and can and should be removed.

Since Régin's GAC algorithm, there have been algorithmic improvements and theoretical analyses. Gent et al. [Gent et al., 2008] presented a method for splitting SCCs into small ones and only checks individual SCCs containing changed variable-value pairs between successive propagation. Because the method avoids the computation of unchanged SCC, it runs 1-5 times faster than the Régin method. In our previous work [Zhang et al., 2018], we proposed a new approach for identifying redundant edges not using Berge's Theorem. We classified redundant edges into two types, the first type of edges can be removed right after finding a maximum matching, and the second type of edges can be removed by computing the SCCs in a smaller subgraph.

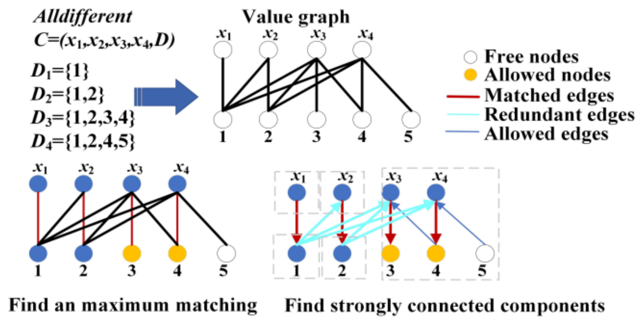

Figure 1: Constraint propagation of an alldifferent constraint.

3 Identify Unimportant Edges of Value Graph

Consider an *alldifferent* constraint *A* (*X*, *D*) and its value graph *B* (*X*, *D*, *E*). We say a value graph is *consistent* if it has no redundant edge. Therefore, enforcing GAC for the *alldifferent* constraint is equivalent to making the value graph be always consistent during the solving process. When a CSP solver moves down in the searching tree, the value graph may become inconsistent because some edges are removed from the consistent value graph by other constraints. If an edge's removal affects the consistency of the value graph, we say it is important. Formally, an edge *e* of a consistent value graph *B* (*X*, *D*, *E*) is *important*, if graph *B* (*X*, *D*, *E*-*e*) has redundant edges; otherwise, we call the edge *unimportant*. Removing unimportant edges will not affect the consistency of the value graph, and therefore there is no need to invoke a constraint propagator. Thus, identifying unimportant edges is critical for developing an efficient *alldifferent* propagator. Here, we present two lemmas and a novel theorem to identify the unimportant edges of a value graph.

Lemma 1. Consider a consistent value graph *B* (*X*, *D*, *E*) and a maximum matching *M* for *B*, an edge *e* (*x*, *y*) ∈*M* is unimportant, *iff*, graph *B*' (*X*, *D*, *E*-*e*) has at least two arc-disjoint *M*-alternating paths connecting node *x* and *y*.

Proof. Sufficiency. Suppose there exist two arc-disjoint alternating paths $P_1(x, y)$ and $P_2(x, y)$ in graph *B*' (*X*, *D*, *E*-*e*) (Figure 2A). Because *e* is a matched edge, P_1 and P_2 must be augmenting paths w.r.t. *M*. We can expand P_1 and get new matching $M' = P_1 \oplus M$, therefore, P_1 became an alternating path and P_1+P_2 is an alternating cycle w.r.t. *M*'. Therefore, *x* and *y* are still in the same alternating cycle. Because the only difference between *B* and *B*' is the removal of edge *e* (*x*, *y*) and *x* and *y* are mutually reachable, all nodes of *B*' are in the same SCC and *B*' remains consistent.

Necessity. Suppose *e* is unimportant, which means that *B*' (*X*, *D*, *E*-*e*) and *B* (*X*, *D*, *E*) have the same SCCs. Because *x* and *y* are in the same SCC, there must be at least two paths connecting *x* and *y*. Because *e* is a matched edge, *B*' must have a new maximum matching *M*'. Therefore, there must be an augmenting path connecting *x* and *y* w.r.t. *M*. Therefore, *B*' (*X*, *D*, *E*-*e*) has at least two arc-disjoint *M*-alternating paths connecting nodes *x* and *y* w.r.t. *M*, which completes the proof. □

Lemma 2. For a consistent value graph *B* and one of its maximum matching *M*, an unmatched edge *e* (*x*, *y*) is unimportant, *iff*, graph *B*' (*X*, *D*, *E*-*e*) has at least two arc-disjoint alternating paths connecting nodes *x* and *y*, and one of the paths starts and ends with matched edges.

Proof. Sufficiency. Suppose there exist two arc-disjoint alternating paths $P_1(x, y)$ and $P_2(x, y)$ and $P_1(x, y)$ starts and ends with matched edges (Figure 2B), it is straightforward to see that P_1+P_2 forms an alternating cycle w.r.t. *M* Therefore, nodes *x* and *y* must be in the same SCC. The only difference between *B* and *B*' is the removal of *e* (*x*, *y*), therefore, *B*' must have the same SCCs as *B* and still be consistent.

Necessity. Suppose *e* is unimportant and *B*' (*X*, *D*, *E*-*e*) has the same SCCs as *B*. Because *x* and *y* are in the same SCC, there must be at least two paths with inverted directions connecting *x* and *y*, which completes the proof. □

The main idea underlying Lemma 1 and 2 is that when an edge e (*x*, *y*) is removed from a consistent value graph, the value graph remains consistent if there exists an alternative cycle containing nodes *x* and *y*. When the removed edge is a matched edge, the constraint propagator needs to first repair the maximum matching, and then computes the SCCs. The two arc-disjoint M-alternating paths w.r.t. the original maximum matching is exactly the alternative cycle w.r.t. the new maximum matching (Figure 2A), as proved in Lemma 1. When the removed edge is unmatched, we only need to "repair" the deleted edge, which can be done by finding an alternating cycle connecting the end nodes of the edge as proved in Lemma 2. Furthermore, the above Lemmas can be readily extended to the cases where multiple edges are removed from the consistent value graph, as stated in the following Theorem:

Theorem 1. For a consistent value graph B (X, D, E) and a set of edges $RE=\{e_1(x_1, y_1), e_2(x_2, y_2), \ldots, e_i(x_i, y_i)\}$, $B'(X, D, E-RE)$ is consistent, *iff*, for every node pair (x_i, y_i) in RE, there exist at least two arc-disjoint alternating paths connecting x_i and y_i.

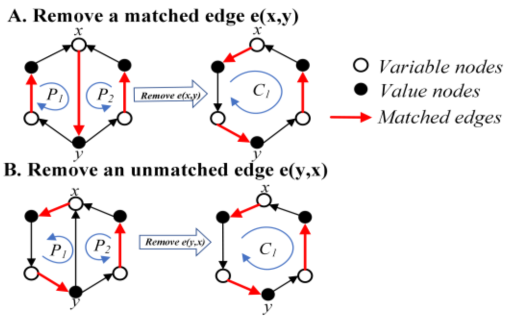

Figure 2: Illustration of Lemma 1 and 2

Figure 3 shows an example of removing two edges from a consistent value graph of an *alldifferent* constraint. The CSP solver may assign values to variables or invoke the propagator on the other constraints, which may remove edges from the consistent value graph. For example, $e(b,2)$ and $e(c,4)$ are removed from the value graph (Figure 3A). As removing the two edges may affect the consistency of the value graph, we need to decide whether the propagator should be called or not. This can be done by first repairing the maximum matching (Figure 3B) and then finding alternating cycles between the end nodes of edges $e(b,2)$ and $e(c,4)$ (Figure 3C). As we only traverse the partial graph to find alternating cycles (yellow shaded area in Figure 3B), our approach is faster than Régin's and Gent's GAC algorithms which traverse the whole value graph.

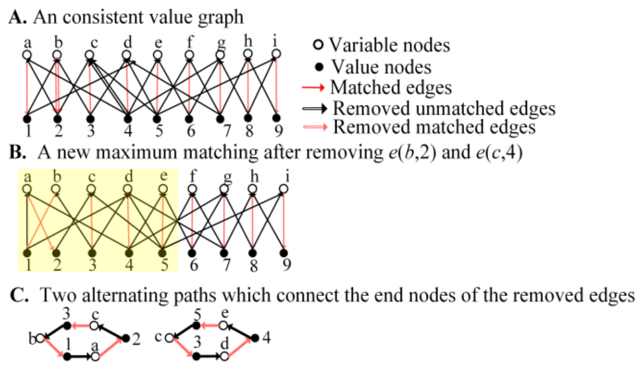
A. An consistent value graph
a b c d e f g h i
1 2 3 4 5 6 7 8 9
Variable nodes
Value nodes
Matched edges
Removed unmatched edges
Removed matched edges
B. A new maximum matching after removing e(b,2) and e(c,4)
C. Two alternating paths which connect the end nodes of the removed edges
3 c
b 1 a 2
5 e
c 3 d 4

Figure 3: An example for removing two unimportant edges from a consistent value graph

4 Fast *Alldifferent* Propagator by Early Detection

We are now ready to present a fast filtering algorithm for *alldifferent* constraint. Theorem 1 allows us to identify unimportant edges of a consistent value graph by finding alternating cycles. Here we will show how to effectively incorporate the identification of unimportant edges into an existing constraint propagator with very little computation.

A naïve idea is to find a cycle for each removed edge in the directed value graph. If the edge is unimportant, no constraint propagation is needed; otherwise, the whole value graph may need to be traversed. As discussed in Section 2, the bulk of the overall computation for a propagator is used to compute SCCs which can be done in linear time by Tarjan's algorithm [Tarjan, 1972]. If we encounter an important edge, we need first to traverse the whole graph to find an alternating cycle, which will be failed because the edge was important. And then we need to traverser the graph again to find SCCs of the graph. Therefore, the whole computation cost will be at least twice as much time as the original approach, which will significantly slow down the whole algorithm.

To address the issue above, we designed an effective approach to introduce the identification of important edges into Tarjan's SCC algorithm. It is important to note that a depth-first search (DFS) on the value graph is sufficient to find cycles for removed edges. Also, note that Tarjan's SCC algorithm itself runs DFS. As such it can naturally accommodate the finding of a cycle for a deleted edge. More specifically, if a back-edge, a non-tree edge pointing to a visited node, is found during the DFS search, we must have already found a cycle and only need to check whether the cycle contains the end nodes of the deleted edge. If all deleted edges are found in a cycle, we can immediately terminate the process of constraint propagation.

We modified Tarjan's SCC algorithm by checking if the removed edges are important or not. If all removed edges are unimportant, we immediately stop the procedure for finding SCCs. When there are important edges, we continue to find SCCs and remove the redundant edges between the SCCs. Overall, our approach only adds a small amount of computation for checking if the removed edges are in cycles when searching for SCCs. This idea and steps for early detection in SCCs are formulated in Algorithm 1.

The main procedure of the above algorithm is the same as Tarjan's SCC algorithm. We only add lines 12-14 to check whether the deleted edges are in the cycles and lines 18-19 to decide when to stop the current propagation. During the DFS search, if we encounter a back-edge e (a, b), the nodes in the DFS tree between nodes a and b must be in a cycle. Therefore, we use two numbers (the DFS index of first visited node and the last visited node) to represent a cycle. The function addCycles is for maintaining the existing cycles. If we find a back-edge, we either push it as a new cycle or merging it into existing cycles (lines 22-29). The function inCycles is for testing whether an edge e (a, b) is in a cycle, which can be done by comparing the DFS index of the edge (line 32). If we found a partial SCC during the DFS (line 20), we will not check the cycles and the algorithm will run as Tarjan's SCC algorithm. Therefore, our modification can be done very efficiently.

Algorithm 1: Find SCCs With Early Detection

```
1.  Input: directed value graph $B(c) = (X_c, D_c, E)$, deleted
    edges $DE$;
2.  Output: SCCs
3.  function Tarjan ($B$, $DE$)
4.  cycles← ∅; S ← ∅; index←0; unconnected←false;
5.  for each unvisited node $v \in X_c \cup D_c$
6.      Strongconnect($v$);

7.  function Strongconnect(v)
8.  DFS[$v$] ← index; lowLink[$v$] ← index; index ← index + 1;
    $S$.push($v$);
9.  for each $e$ ($v$, $w$) $\in E$ do
10.     if $w \in S$ then
11.         lowLink[$v$] ← min(lowLink[$v$], DFS[$w$])
12.         if unconnected = false then
13.             addCycles (lowLink[w], index-1)
14.             while inCycles($DE$.top()) = true do
15.                 $DE$.pop();
16.     else if $w$ is unvisited then
17.         Strongconnect($w$);
18.         lowLink[$v$] ← min(lowLink[$v$], lowLink[$w$]);
19. if lowLink[$v$] = DFS[$v$] then
20.     pop nodes before $v$ and pop $v$ from $S$, add them to SCCs
    as an SCC; unconnected←true;
21. if unconnected = false and $DE$ is empty then
22.     stop the current propagation

23. function addCycles (a, b)
24. for each pair (a', b') ∈cycles
25.     if interval [a, b] is overlapped with interval [a',b'] then
26.         a'←min(a',a) and b'←max(b',b);
27.         return
28. cycles.push(a, b)
29. return

30. function inCycles (e(a,b))
31. for each pair (a', b') ∈cycles
32.     if both DFS[a] and DFS[b] are in the interval [a', b']
    then
33.         return true
34. return false
```

Algorithm 1 shows the steps for SCCs with early termination. For the remaining parts of the *alldifferent* propagator, such as computing a maximum matching, we use the same method as [Gent et al., 2008].

5 Experimental Analysis

To evaluate the performance of our new method, we used Minion constraint solver 1.8 [Gent et al., 2006] to implement our algorithm. All experiments were run on a Windows 10 workstation with an Intel Xeon E5-2680 v2 processor of 2.8 GHz and 16GB DDR3 1600MHz RAM. The Minion constraint solver already has an implementation of *alldifferent* propagator with several optimization techniques, such as incremental matching [Régin, 1994], BFS matching [Cormen et al., 1990], staged propagation [Schulte and Stuckey, 2004], priority Queue [Schulte and Stuckey, 2004], assign optimizing and SCC splitting [Gent et al., 2008]. To implement our algorithm, we used BFS maximum matching and Tarjan's SCC algorithms. We adopted the depth-first chronological backtracking as the search strategy.

The benchmark problems for valuation were from the CSP Lib (http://www.csplib.org/) and Minion constraint solver (https://constraintmodelling.org/minion/), which include several typical *alldifferent* constraint problems. These problems including Langford's number problem (prob024 in CSPLib), Golomb ruler problem (prob006 in CSPLib), Quasigroup existence (prob003 in CSPLib), Social golfers (prob010 in CSPLib), Graceful graphs (prob053 in CSPLib), Magic Squares (prob019 in CSPLib), N-Queens (prob054 in CSPLib) and Sports scheduling (prob026 in CSPLib). We generated 93 problem instances with different sizes for the above eight problems by using Minion, Conjure [Frisch et al., 2005], and Savile row [Nightingale et al., 2017] generator. We generated Golomb ruler instances with the number of ticks from 50 to 80 and the max length of the ruler from 2500 to 6400; graceful graphs instances with five cliques and number of nodes in each clique from 25 to 40; Langford's number instances with 20 and 100 sets and numbers in a set from 5 to 49, quasigroup existence instances with orders from 30 to 65, social golfers instances with 9 groups, 4 golfers in each group and number of weeks from 11 to 15; magic squares instances with orders from 4 to 70, n-queens instances with the number of queens from 128 to 2048; sports scheduling instances with the number of teams from 30 to 80.

To appreciate the importance of detecting useless constraint propagation, we first analyzed how frequent useless constraint propagation was invoked. We selected 31 instances from the eight problems and used Minion's default *alldifferent* propagator, which was proposed by [Gent et al., 2008]. These instances are relatively small and can be solved within a time limit to allow an efficient benchmarking of the effectiveness of the propagator. To this end, we counted the fraction of the total invocations in which the propagator does not remove any inconsistency (Figure 4). As expected, most of the invocations are useless. For example, for problems of Sports scheduling, Quasigroup existence and Graceful graphs, more than 80% to 90% invocations are useless and can thus be skipped to reduce overall computation.

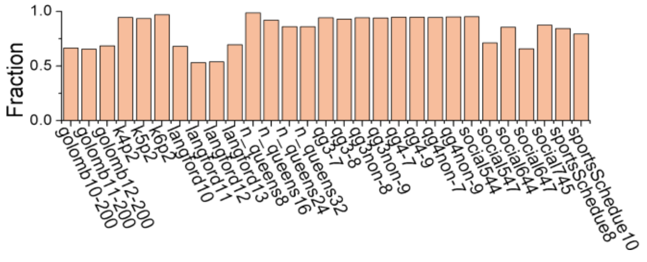

Figure 4: Fraction of useless propagator invocation

Next, we compared our algorithm with Régin's algorithm [Régin, 1994], Gent's algorithm [Gent et al., 2008], and Zhang's algorithm [Zhang et al., 2018] on the 93 problem instances. The individual optimizing techniques used in these algorithms are listed in Table 1. Because most of the problem instances tested are very large and cannot be solved quickly, we set the time limit to 1,200 seconds and counted the number of nodes searched per second following [Gent et al., 2008]. Figures 5 show the ratios of the numbers of nodes searched per second of our algorithm and the other three algorithms, showing the expected speedup the new algorithm offered. As expected, our algorithm is faster than the previous algorithms for most instances, with up to more than 5 times speedup.

Optimizing Technique	Régin's	Gent's	Zhang's	Our
Incremental matching	✓	✓	✓	✓
Staged propagation	✓	✓	✓	✓
Priority Queue	✓	✓		✓
Assign optimizing		✓		✓
Matching optimizing			✓	
SCC splitting		✓		✓
Early detection (Our)				✓

Table 1: Optimizing technique used in the experiments.

Furthermore, our new algorithm exhibits a better performance on the problems of Langford's number, Golomb ruler, and Graceful graph (Figure 5). We believe this is primarily because these problems have large value graphs. On a large problem instance, an invocation of constraint propagator in our algorithm only needs to transverse a small part of the value graph and prunes a large portion of the search space. In contrast, for small problems with small value graphs, our algorithm may have minor performance improvements. To support that, we designed two simulated experiments focus on SCC computation to evaluate the performance of our algorithm on a seizes of synthetic bipartite graphs with different sizes. Similar to the filtering process of the *alldifferent* constraint, we randomly removed edges from these graphs and computed the SCC until there exist multiple SCCs in the graph. Figure 6A showed the speedup ratio of the total runtime of our algorithm compared to Tarjan's SCC algorithm. As we expected, denser graphs have higher speedup ratios. Furthermore, we also performed the above experiment on bipartite graphs which have the same size as the *alldifferent* instances. Figure 6B showed that, for graphs whose sizes are the same as Golomb, Graceful graph, Langford, and Magic square problems, the speedup ratio is greater than the others, such as social golfer, quasigroup. The speedup ratio difference is correlated with the average degree of the graphs (Figure 6B).

As indicated by the better performance on larger problem instances, the new algorithm enjoys excellent scalability. To quantify this scalability, we used instances of varying sizes for Langford's number, Golomb ruler, and Graceful graph problems. The speedup of our method over Gent's algorithm increases steadily and significantly with the problem size (Figure 7). This suggests that the new algorithm is the method of choice for large *alldifferent* constraint problems in practice.

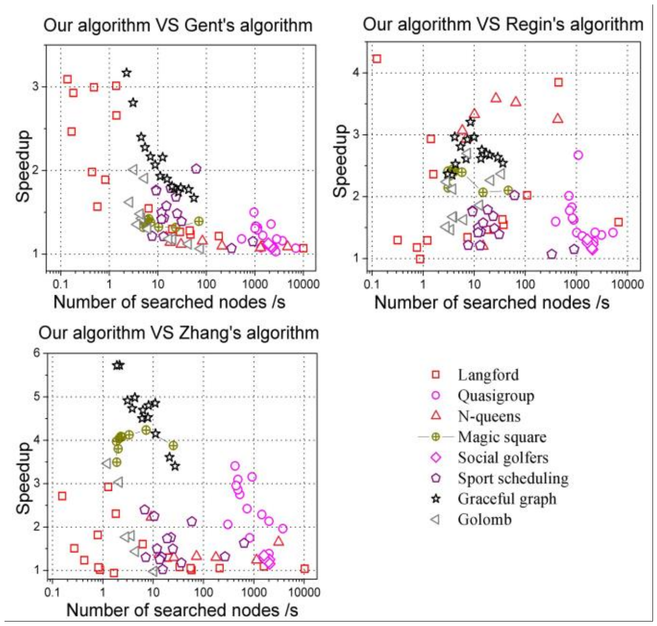

Figure 5: Speedup of our algorithm over Gent's, Régin's and Zhang's algorithms

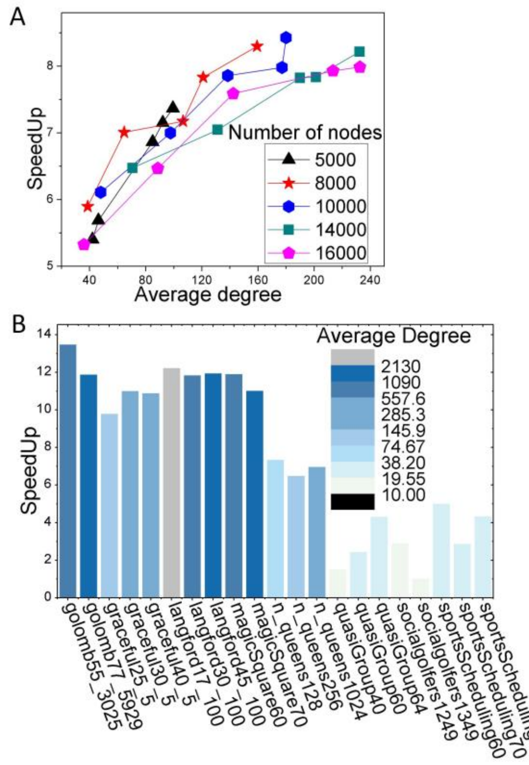

Figure 6: Speedup of our algorithm over Tarjan's algorithm. **A**. Speedup versus average degree on graphs with different sizes; **B**. Speedup on bipartite graphs which have the same size as the *alldifferent* instances.

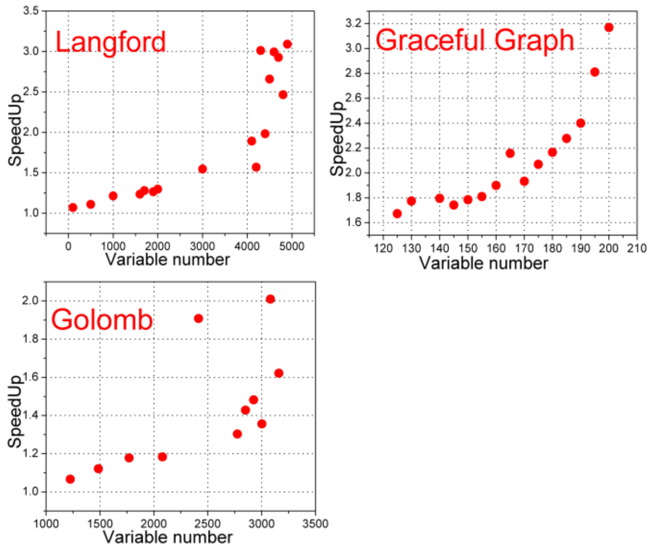

Figure 7: Speedup of our algorithm over Gent's algorithm for problem instances of different sizes.

6 Conclusion and Discussion

Alldifferent constraints appear broadly in many real-world constraint problems, and efficiently propagating *alldifferent* constrains is of great importance for constrain programming. Despite many efforts, no theoretical advance has been made since Régin's seminal work on formulating constraint propagation as maximum matching in a bipartite graph, and the existing methods based on Régin's theorem seemed to reach an algorithmic bottleneck. The lack of effective means to predict useless constraint propagation is the main roadblock preventing further development to advance CSP research. In this paper, we presented a novel theorem for the identification of unimportant edges in the value graph of an *alldifferent* constraint problem whose removal does not need to invoke constraint propagation. This is a long-waited theoretical improvement to the Régin's result. Exploiting this theoretical result, we developed a new optimizing technique for early determination of useless constraint propagation and an efficient filtering algorithm that can significantly reduce or even avoid useless constraint propagations. We compared our algorithm with Régin's algorithm and the other state-of-the-art approaches on benchmark instances, showing that the new algorithm significantly outperforms the existing approaches. More importantly, the performance of our algorithm improves as instance sizes increase, indicating that the new algorithm has favorable scalability.

The idea of merging the detection of useless constraint propagation with backtracking search is general and can be adopted for other constraint propagators beyond the *alldifferent* constraint. As many other global constraints, such as cardinality constraints [Thalheim, 1992], can be modeled by value graphs, our method has a great potential for efficiently solving constraint satisfaction problems that contain *alldifferent* and similar global constraints.

Acknowledgments

We would like to thank the anonymous reviewers for their constructive comments. The work was partially supported by the National Natural Science Foundation of China (Granted No. 61972063).

References

[Apt, 1998] Apt, Krzysztof R. The Essence of Constraint Propagation. *Theoretical Computer Science,* pages 179-210, 1998.

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

[Boisberranger et al., 2013] Du Boisberranger J, Danièle Gardy, Lorca, X and Truchet, C. When Is It Worthwhile to Propagate a Constraint ? A Probabilistic Analysis of All Different. Analytic Algorithmics and Combinatorics, 2013.

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

[Frisch et al., 2005] Alan M. Frisch, Chris Jefferson, Bernadette Martínez-Hernández, and Ian Miguel. The Rules of Constraint Modelling. In *International Joint Conference on Artificial Intelligence*, 2005.

[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. Generalized arc consistency for the alldifferent constraint: An empirical survey. Artificial Intelligence, 172 (18): 1973-2000, 2008.

[Golomb and Baumert, 1965] Solomon W. Golomb and Leonard D.Baumert. Backtrack Programming. *Journal of the ACM (JACM)*, 12(4):516-524,1965.

[Hentenryck et al., 1992] Pascal Van Hentenryck, Yves Deville, and Choh-Man Teng. A Generic Arc-Consistency Algorithm and Its Specializations. *Artificial Intelligence*, 57 (2–3): 291–321, 1992.

[Hoeve, 2001] Willem Jan Hoeve. The Alldifferent Constraint: A Survey. In *Proceedings Sixth Annual Workshop of the ERCIM Working Group on Constraints*, 1–42, 2001.

[Katriel, 2006] Irit Katriel. Expected-Case Analysis for Delayed Filtering. International Conference on Integration of Artificial Intelligence. Springer-Verlag, 2006.

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

[Nightingale et al., 2017] Peter Nightingale, Özgür Akgün, Ian Gent, Christopher Jefferson, Ian Miguel, and Sprackle n, P. Automatically Improving Constraint Models in Savile Row. *Artificial Intelligence*, 251, 2017.

[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.

[Rossi et al., 2006] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming*. Elsevier, 2006.

[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.

[Tarjan, 1972] Robert Tarjan. Depth-first search and linear graph algorithms. SIAM journal on computing 1(2): 146160, 1972.

[Thalheim, 1992] Bernhard Thalheim. Fundamentals of Car dinality Constraints. In *Lecture Notes in Computer Scien ce (Including Subseries Lecture Notes in Artificial Intelli gence and Lecture Notes in Bioinformatics)*, 1992.

[Zhang et al., 2018] Xizhe Zhang, Qian Li, and Weixiong Z hang. A Fast Algorithm for Generalized Arc Consistency of the Alldifferent Constraint Joint Laboratory of Artific ial Intelligence and Precision Medicine of China Medical University And. *Proceedings of the Twenty-Seventh Inte rnational Joint Conference on Artificial Intelligence*, Jul y, 1398–1403, 2018.