Quantum-accelerated global constraint filtering

Kyle E. C. Booth[1,2], Bryan O'Gorman[1,3], Jeffrey Marshall[1,2], Stuart Hadfield[1,2], and Eleanor Rieffel[1]

[1] Quantum AI Laboratory (QuAIL), NASA Ames Research Center, Moffett Field, CA 94035, USA {kyle.booth, jeffrey.s.marshall, bryan.ogorman, stuart.hadfield, eleanor.rieffel}@nasa.gov
[2] USRA Research Institute for Advanced Computer Science (RIACS), Mountain View, CA 94043, USA
[3] University of California, Berkeley, CA 94720, USA

Abstract. Motivated by recent advances in quantum algorithms and gate-model quantum computation, we introduce quantum-accelerated filtering algorithms for global constraints in constraint programming. We adapt recent work in quantum algorithms for graph problems and identify quantum subroutines that accelerate the main domain consistency algorithms for the `alldifferent` constraint and the global cardinality constraint (`gcc`). The subroutines are based on quantum algorithms for finding maximum matchings and strongly connected components in graphs, and provide speedups over the best classical algorithms. We detail both complete and bounded-probability frameworks for quantum-accelerated global constraint filtering algorithms within backtracking search.

Keywords: quantum computing · constraint programming · backtracking search · logical inference · global constraints.

1 Introduction

Quantum computers are designed to leverage quantum-mechanical phenomena to outperform classical computers for certain tasks. While early quantum devices, such as quantum annealers, were limited to the implementation of specialized algorithms, the past decade has seen the advent of general-purpose gate-model quantum computers, capable of implementing any algorithm that can be expressed as a series of quantum logic gates. In this model, quantum gates are applied to qubits, the basic memory unit of quantum processors, reminiscent of classical computation where logic gates are applied to bits. While the current gate-model processors remain small, in the noisy intermediate-scale quantum (NISQ) regime, they have already enabled exciting developments, such as the availability of quantum computers accessible in the cloud [10,13], and the achievement of quantum supremacy in the context of sampling random quantum circuits [3]. Additionally, a well-developed theory of quantum error correction and quantum fault tolerance provides the underpinnings of extensive engineering efforts to realize fault-tolerant, scalable quantum computers [33].

Concurrent work in advancing quantum algorithms is critical to extend the known applications of quantum computing independent of processor design. Recent efforts indicate speedups for a number of problems in graph theory [14], mathematical programming [35], constraint satisfaction [9], and search [25,26].

Building on these results, we investigate quantum subroutines to accelerate filtering algorithms for global constraints. We argue that the search paradigm in constraint programming (CP) represents an attractive framework for deployment of quantum subroutines that accelerate inference algorithms at each node of a search tree. Encapsulation of combinatorial substructure in global constraints provides an elegant mechanism for carving off portions of complex problems into subproblems that can be solved by a quantum co-processor. These smaller subproblems require fewer qubits, making them promising candidates for early fault-tolerant quantum chips. While CP has been used recently to efficiently compile quantum circuits [6], the use of quantum algorithms to accelerate global constraint filtering in CP, has, to the authors' knowledge, not been investigated.

The primary contributions of this paper are as follows:

i. A quantum-accelerated $O(|X|\sqrt{(|X|+|V|)|V|}\log^2|V|)$-time bounded-error algorithm for domain consistency of the `alldifferent` constraint, where $|X|$ is the number of variables and $|V|$ is the number of unique domain values. Our approach follows the main classical algorithm, accelerating the basic subroutines performed at each iteration with quantum analogs. The complexity is dominated by that for finding maximum matchings in bipartite graphs. The best deterministic and randomized classical algorithms known take $O(|X|\sqrt{|X|}|V|)$ and $O(|X|^{\omega-1}|V|)$ time, respectively, where ω corresponds to the asymptotic cost of classical matrix multiplication; the best upper bound known on ω is 2.373.[4] Our approach improves over these time-complexity upper bounds by factors on the order of $\sqrt{|X||V|/(|X|+|V|)}$ and $\sqrt{|X|^{2\omega-4}|V|/(|X|+|V|)}$, respectively, and up to polylogarithmic terms.
ii. A quantum-accelerated $O(|X|\sqrt{(|X|+|V|)|V|}\log^2|V|)$-time bounded-error algorithm for domain consistency of the global cardinality constraint (`gcc`), providing speedups over the best classical approach known.
iii. We discuss complete and bounded-probability frameworks for using quantum-accelerated global constraint filtering in backtracking tree search.

2 Background

Quantum computers work in a fundamentally different way than classical computers: they process quantum information, a generalization of classical information, and can use uniquely quantum operations to carry out computations. At an abstract level there are similarities between the two paradigms: quantum computers have quantum registers that hold quantum states, and a quantum

[4] We note that the instance size at which the asymptotic scaling becomes relevant is so large that, in practice, matrix multiplication takes cubic time.

computation acts on these states by quantum gates. Some of these gates are directly analogous to classical gates such as NOT and CNOT, but others are uniquely quantum. The fundamental unit of information on which quantum computers act is a qubit, a generalization of the classical bit. At the end of a quantum computation, during which quantum gates are applied to qubits in the quantum registers, quantum measurements are made to the qubits to extract classical information, such that a string of bits is returned. The interested reader is referred to a number of sources for a more thorough review of the subject [33,38].

We adopt Dirac's "ket" notation [11], universally used in quantum mechanics and quantum computing, in which a column vector is represented by a "ket" such as $|x\rangle$, where x is a label, equivalent to $\overrightarrow{x}$. Generally, there will be some preferred basis of the vector space referred to as the "computational basis". For example, a 3-dimensional vector space can be spanned by $|0\rangle$, $|1\rangle$, and $|2\rangle$, corresponding to the unit vectors $(1,0,0)^T$, $(0,1,0)^T$, and $(0,0,1)^T$, respectively. Finally, the notation $|x\rangle\,|y\rangle$ is shorthand for the tensor product $|x\rangle \otimes |y\rangle$.

Definition 1 (Qubit). *A qubit is a quantum system whose state is represented by a two-dimensional complex vector. The computational basis consists of two orthonormal vectors denoted* $|0\rangle$ *and* $|1\rangle$*. Unlike a classical bit, which must be either 0 or 1, a qubit can in general be in a superposition of these states* $a\,|0\rangle + b\,|1\rangle$*, subject to the normalization condition* $|a|^2 + |b|^2 = 1$*, where* a *and* b *are complex numbers referred to as "amplitudes".*

Definition 2 (Quantum register). *An* n*-qubit quantum register holds the state of* n *qubits, represented as a vector in a* 2^n*-dimensional complex vector space. The computational basis* $\{|\mathbf{x}\rangle : \mathbf{x} \in \{0,1\}^n\}$ *consists of* 2^n *orthonormal vectors, labeled by the* 2^n *classical* n*-bit strings* $\{0,1\}^n$ *or the corresponding integers* $\{0, 1, \ldots, 2^n - 1\}$*. Any* n*-qubit state can be written as a superposition (linear combination) of the form* $|\phi\rangle = \sum_{\mathbf{x}\in\{0,1\}^n} a_{\mathbf{x}}\,|\mathbf{x}\rangle$*, subject to* $\sum_{\mathbf{x}} |a_{\mathbf{x}}|^2 = 1$*.*

Definition 3 (Quantum measurement). *A measurement of a quantum register in the computational basis returns classical information. Specifically, it will return each bit string with probability proportional to the amplitude squared. For example, measuring a qubit in state* $a|0\rangle + b|1\rangle$ *returns state* $|0\rangle$ *with probability* $|a|^2$ *and state* $|1\rangle$ *with probability* $|b|^2$*. The qubits of a multi-qubit register may be measured independently, and in general the outcomes will be correlated.*

Quantum information processing is an interesting mix of quantum states, which can take on a continuum of values, and quantum measurement, which enforces discrete outcomes. The design of quantum algorithms, the topic of this paper, involves transforming quantum systems. These transformations can be represented as unitary matrices, and each of these can be decomposed into a sequence of one- and two-qubit transformations called quantum gates.

Definition 4 (Quantum gate and quantum circuit). *A quantum state transformation (i.e., a unitary operator) acting on a quantum register is called a quantum gate. A quantum circuit is a sequence of quantum gates.*

In gate-model quantum computation, the resources by which algorithms are compared include the number of qubits used (the *space complexity*) and the number of primitive gates used (the *gate complexity* or *time complexity*).

2.1 Grover's algorithm and quantum search

In this section, we introduce Grover's algorithm for unstructured search [18], a well-known quantum algorithm, and essential for the speedups in this work.

Definition 5 (Unstructured search problem). *Given an N element unstructured list and blackbox access to predicate $P : \{0, \ldots, N-1\} \to \{0, 1\}$, find a* solution*, $x \in \{0, 1, \ldots, N-1\}$, such that $P(x) = 1$, with the fewest queries to P.*

For a predicate P with m solutions, Grover's algorithm finds a solution to the unstructured search problem with constant success probability using $O(\sqrt{N/m})$ queries to P, even when m is unknown. Classical (including randomized) algorithms require $\Omega(N/m)$ queries, and so Grover's algorithm provides a quadratic speedup in the oracle model. In the quantum case, the predicate is instantiated as an operator $U_P : |x\rangle |0\rangle \mapsto |x\rangle |P(x)\rangle$ that computes $P(x)$ in the $|0\rangle$ register. This speedup is due to quantum computing's ability to evaluate P on a superposition of states according to $U_P \sum_x a_x |x\rangle |0\rangle = \sum_x a_x |x\rangle |P(x)\rangle$. By linearity, the operator U_P computes $P(x)$ over all x in superposition. Grover's algorithm is optimal in the sense that any quantum algorithm for the unstructured search problem must have query complexity $\Omega(\sqrt{N/m})$. With a modification, all m solutions can be retrieved using $O(\sqrt{mN})$ queries.

For the unstructured search problem, it is evident that Grover's algorithm quadratically improves over the best possible classical algorithm. For more complicated problems, however, this is not always the case, as unconditional complexity lower bounds are difficult to obtain. As such, throughout the paper, we claim a speedup for a quantum algorithm when an improvement in time complexity, for a given problem, is shown over the best classical algorithm known.

A sketch of Grover's algorithm. Leveraging the ability of quantum computing to compute on quantum superposition states, Grover's algorithm exploits quantum interference to concentrate amplitude on the target states. We use Fig. 1 to provide a pictorial representation of the algorithm [33]. In this example it suffices to consider real-valued amplitudes only.

Initialization. The algorithm starts with a uniform superposition state, $|\psi\rangle = \frac{1}{\sqrt{N}} \sum_x |x\rangle$, of all N values of the search space. In Fig. 1a, this uniform superposition corresponds to a uniform amplitude histogram (top), where each bar in the histogram represents the amplitude associated with an element in the search, and is represented by a red vector (bottom) at some angle away from the target state (y-axis). In these diagrams, the red vector represents some superposition state of the N values.

Amplitude amplification. This process seeks to amplify the amplitude associated with the target state, while diminishing the amplitudes associated with

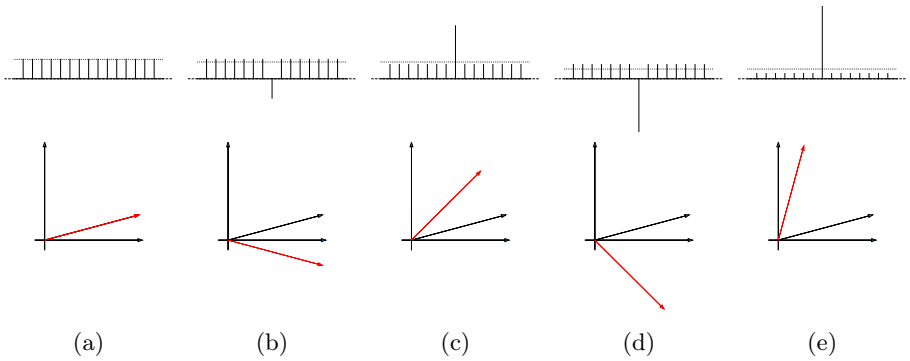


Fig. 1: Grover's algorithm for unstructured search [18], illustrating initialization and two rotations. **Top:** concentration of amplitude on search states (vertical lines) with each iteration, where the dotted horizontal line represents the average amplitude. **Bottom:** the changing superposition state (red vector) as reflection operations are applied, moving towards the target y-axis [33].

all the other states. It is accomplished with $O(\sqrt{N/m})$ applications of two operations which, using a geometrical interpretation, can be seen as reflections of the superposition state vector about axes. The first operation reflects the superposition state about the x-axis, representing a state orthogonal to the target states, through the application of the oracle U_P. As in Fig. 1b, this reflection yields a new superposition state (red vector), a negative amplitude for the target states, and a lowering of the average amplitude. The second operation reflects the superposition state about the uniform superposition state, $|\psi\rangle$. As shown in Fig. 1c, this results in a positive, more concentrated amplitude associated with the target states and a superposition state (red vector) closer to the y-axis.

Each iteration of Grover's algorithm rotates the initial state, $|\psi\rangle$, towards the target states (y-axis). A straightforward calculation shows that quantum interference effects facilitate transfer of probability amplitude from non-target states to targets states. (Figs. 1d and 1e provide one more iteration.) At the end of the algorithm, a measurement yields one of the solutions with probability proportional to the amplitude squared. We note that Grover's algorithm can be expressed as a quantum circuit consisting of logical gates and queries to U_P.

A few comments are in order. The rotation perspective correctly suggests that choosing the number of iterations is critical; otherwise over-rotation may occur, resulting in less amplitude in the target states. The number of iterations depends only on m, the number of target states, not on which states these are. Specifically, the optimal number of rotations is $\frac{\pi}{4}\sqrt{\frac{N}{m}}$ [7]. If m is not known, it can be estimated using a quantum counting algorithm, or by running the algorithms a number of times, with an increasing number of iterations until discovery of a marked state. Both preserve the overall query complexity of $O(\sqrt{N/m})$ [7]. The algorithm is a bounded-probability algorithm, meaning that it can fail to find a solution element even if one exists, an important aspect discussed in Section 5. To ensure the failure probability of each Grover search is polynomially small in

N, we repeat the algorithm $O(\log N)$ times, contributing only to the logarithmic terms noted in many of the complexity results.

2.2 Related work

The use of quantum search to realize speedups for various problems has seen a surge of activity in recent years. Previous work provides lower and upper bounds for the bounded-error query complexity of various graph problems, including connectivity, minimum spanning tree, and single-source shortest path [5,14]. Related work, leveraged heavily in these papers, investigated the query complexity for various matching problems [2,12]. More recently, quantum search has been applied to problems within mathematical programming, such as semidefinite programming [8,35] and the acceleration of the simplex method [28]. The latter, in a similar fashion to this work, uses quantum search to accelerate the subroutines of the simplex method, such as variable pricing. There also exist recent efforts to use algorithms based on quantum search to speed up tree search methods, including backtracking search [25], and branch-and-bound [26].

3 Quantum subroutines for `alldifferent`

A constraint satisfaction problem (CSP) consists of a set of decision variables $X = \{x_1, \ldots, x_n\}$, with domains $\mathcal{D} = \{D_1, \ldots, D_n\}$, and constraints $\mathcal{C} = \{C_1, \ldots, C_\ell\}$. The domain of a variable is the set of values the variable can be assigned. Each constraint $C \in \mathcal{C}$ acts on a subset of X. A solution is an assignment to the variables of values that satisfies the constraints.

Definition 6 (`alldifferent` constraint). *`alldifferent`$(x_1, \ldots, x_k)$ is a constraint that requires that all of the variables in its scope take on different values (i.e., in a solution to the constraint, $x_i \neq x_j, \forall i \neq j \in \{1, \ldots, k\}$).*

The `alldifferent` global constraint is widely used in CP, and arises naturally in many problems. The main domain consistency filtering algorithm for `alldifferent` was proposed by Régin (see Algorithm 1) and consists of two primary subroutines, FindMaximumMatching and RemoveEdges, leveraging existing graph algorithms [32]. Our approach uses a classical processor that follows Régin's high-level algorithm, accelerating each subroutine using quantum graph algorithms. While recent work has investigated practical optimizations for Régin's algorithm, these do not improve upon its worst-case complexity [16,39].

The algorithm, as in Algorithm 1, begins by constructing a bipartite variable/value graph $G = (X, V, D)$, with vertices $X \cup V$ and edges D. Such a graph has $n = |X| + |V|$ vertices and $m = |D| = \sum_{i=1}^{|X|} |D_i|$ edges, where $m \leq |X||V|$. FindMaximumMatching finds a matching of maximum size in G and RemoveEdges finds edges in G that can never participate in a maximum matching. If FindMaximumMatching returns a matching M whose number of edges $|M| < |X|$, then the constraint cannot be satisfied and the algorithm terminates.

Algorithm 1: The `alldifferent` filtering algorithm of Régin [32]

Result: *False* if no solution, otherwise filtered domains D^*
Build $G = (X, V, D)$;
$M \leftarrow$ FindMaximumMatching(G);
if $|M| < |X|$ **then**
  return *False*;
end
$D^* \leftarrow D \setminus$ RemoveEdges(G, M);
return D^*;

If a matching exists with $|M| = |X|$, the algorithm prunes domains based on the output of RemoveEdges.

The FindMaximumMatching subroutine bears the brunt of the computational complexity [36]. The best deterministic classical algorithms known for finding maximum matchings run in $O(m\sqrt{n})$ time; the algorithm of Hopcroft and Karp (HK) is for bipartite graphs [19], while the algorithm of Micali and Vazirani (MV) applies to general graphs [24,37]. Alt et al. proposed an $O(n^{3/2}\sqrt{m/\log n})$ algorithm [1], however, it only improves upon the aforementioned algorithms for dense graphs. There is also a randomized $O(n^\omega)$-time algorithm [27,20], where ω corresponds to the classical asymptotic cost of matrix multiplication; the best upper bound known on ω is approximately 2.373 [22].

In order to remove edges which participate in no maximum matching, and thus cannot satisfy the constraint, RemoveEdges finds strongly connected components (SCCs) in a directed transformation of G using Tarjan's $O(n + m)$ algorithm [34]. While this subroutine evidently does not bear the computational brunt of `alldifferent` filtering, its acceleration can still be valuable in practice.

In this section, we introduce the quantum query model and definitions needed to describe our approaches. We then detail quantum algorithms for the FindMaximumMatching and RemoveEdges subroutines to accelerate the filtering of the `alldifferent` constraint. For the former, we detail a quantum algorithm proposed by Dörn for finding maximum matchings in general graphs [12]. For the latter, we combine a number of quantum graph algorithms, including an extension of work that identified strong connectivity in graphs [14].

3.1 Input models, accounting, and definitions

Many quantum algorithms are posed in the "oracle model", in which *black box* access is given to the quantum operation $U_f : |w\rangle |0\rangle \mapsto |w\rangle |f(w)\rangle$ in unit time, where $f : W \to Y$ is a classical function encoding the input. (U_f generalizes the operator U_P from Section 2.1 for a non-Boolean function f.) The *query complexity* of such algorithms is the number of calls to the oracle U_f. The time complexity is always at least the query complexity. Because the calls to the oracle are often the most significant part of the computation, the two are often the same (up to polylogarithmic factors), but this isn't always the case.

In this work, we aim to provide a practical speedup, and so must account for the cost of implementing any quantum queries used by our algorithms. We

address this by using quantum random access memory (QRAM) [17], a data structure with which oracle queries and updates (including initialization of the QRAM) to the quantum data structure can be made in time polylogarithmic in the size of the database. There are proposals for special-purpose hardware QRAM, with small-scale experiments demonstrating a proof of principle [21], as well as several ways of implementing QRAM directly in the standard circuit model [23]. These circuit implementations assume the same availability of fault-tolerant, gate-model quantum computers as the algorithms that call the QRAM.

The main template employed here takes an algorithm posed in the oracle model, and uses QRAM to implement the queries with logarithmic overhead, taking care to account for the cost of QRAM initialization. Henceforth, by "time complexity", we mean number of logical gates and queries or updates to QRAM. With a circuit implementation of QRAM, this time complexity upper bounds the overall circuit depth, assuming that the QRAM circuits are parallelized. Parallelization of the logical (non-QRAM) parts of the circuit, which we do not attempt here, can only improve this overall depth.

For our quantum algorithms, we store and access the graph in the "list" (or "array") model. For each vertex $v \in \{X \cup V\}$, we query the oracle:

$$U_{N_v} : |i\rangle\,|j\rangle \mapsto |i\rangle\,|j + N_v(i) \bmod d_v\rangle \tag{1}$$

where d_v is the degree of v, $i \in \{1, \ldots, d_v\}$, $j \in \{1, \ldots, n\}$, and $N_v(i)$ is the ith neighbor of vertex v. Using QRAM, the quantum data structure in Eq. (1) can be initialized in time $O(d_v \log d_v)$ and quantum queries made in time $O(\log d_v)$. When the graph is directed, U_{N_v} queries only the outgoing neighbors. There are other ways of formulating quantum access to a graph, but in this paper we use only the list model.

Definitions. Given a graph G and a matching M, a vertex is *exposed* if no edge in the matching M is incident to it. A path (resp., cycle), consisting of a sequence of vertices, is *alternating* if its edges are alternately in the matching M and not in M. The length of the path (resp., cycle) is the number of edges in the path (resp., cycle). A path is *augmenting* if it is alternating and the first and last vertices in the path are exposed.

3.2 Subroutine: FindMaximumMatching

The essence for a quantum filtering algorithm is simple: use a quantum algorithm to solve the maximum matching problem. Recent work proposed a series of algorithms for finding maximum matchings in terms of calls to a quantum oracle [2,12]; however, to the authors' knowledge, this work has never been linked to accelerating global constraint filtering in CP. In the list model, an initially proposed algorithm is capable of finding maximum matchings in $O(n\sqrt{m+n}\log^2 n)$ time [2], while the second, improved algorithm runs in $O(n\sqrt{m}\log^2 n)$ time [12]. The latter improves over both existing deterministic and randomized algorithms for the majority of parameter values, and follows the classical MV algorithm for

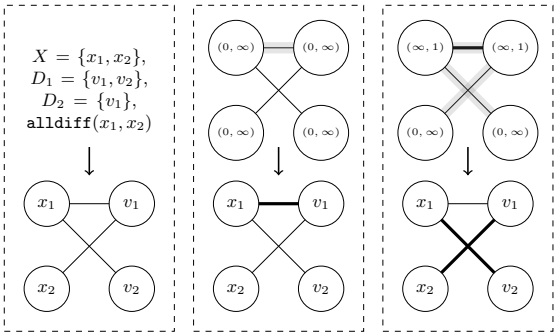
finding maximum matchings in general graphs [24], but accelerates its primary subroutines with quantum search. We give an overview of this algorithm by tracing the classical algorithm in the context of a simple example, and comment on the processes that are sped-up with quantum algorithms.

Example 1. Consider a CSP with variables $X = \{x_1, x_2\}$; domains $D_1 = \{v_1, v_2\}, D_2 = \{v_1\}$; and the constraint $\texttt{alldifferent}(x_1, x_2)$. A trace of the classical MV algorithm for finding a maximum matching is shown in Fig. 2.

We follow the exposition of the MV algorithm of Peterson and Loui [29]. The input to the algorithm is a graph, such as the bipartite variable/value graph G shown in Fig. 2a. Initialization starts with an empty matching, $M = \emptyset$. In phases, the algorithm looks for a set of minimum-length, vertex-disjoint augmenting paths to iteratively extend the current matching until a maximum matching is found. Each phase is performed in $O(m)$ time, and the number of phases is bounded by $O(\sqrt{n})$ [19], resulting in an $O(m\sqrt{n})$-time classical algorithm [24,37]. Each phase of the classical algorithm begins with a matching and conducts three subroutines: Search, BlossAug, and FindPath [24]. As a property of their structure, bipartite graphs do not contain blossoms, precluding the need to cover the algorithmic details associated with dealing with them and, as a consequence, the need for the FindPath subroutine [24]. As such, our FindMaximumMatching subroutine needs only the Search and BlossAug lower-level subroutines and their quantum implementations.

Search. This subroutine performs a simultaneous breadth-first search (BFS) from each exposed vertex. (E.g., in Fig. 2b all vertices are exposed since $M = \emptyset$.) The subroutine labels each vertex with a value pair, $(EvenLevel, OddLevel)$, where $EvenLevel$ (resp., $OddLevel$) is the length of the minimum even- (resp., odd-) length alternating path from an exposed vertex to the current vertex, if any, and ∞ otherwise. Exposed vertices have $EvenLevel = 0$ (and infinite $OddLevel$), resulting in the labeling at the top of Fig. 2b. The subroutine then searches for *bridges*, edges whose vertices both have finite $EvenLevel$ or both have finite $OddLevel$ values. In Phase 1 of the example (Fig. 2b), all edges are bridges. In Phase 2 of the example (Fig. 2c), only edge (x_1, v_1) is a bridge. The Search subroutine passes each discovered bridge to the BlossAug subroutine.

Quantum algorithm. The quantum acceleration of the Search subroutine, instead of a classical BFS, uses a quantum BFS search [12] from each exposed vertex, using query access to a list-model representation of the graph, and a stack bookkeeping the discovered vertices, both implemented with QRAM. From a given vertex v, one needs to obtain all n_v neighbors not yet discovered. If the degree of v is d_v, the quantum time complexity to find all neighbors is $O(\sqrt{n_v d_v} \log d_v)$, as discussed in Section 2.1. Because $O(n)$ searches are run, each is repeated $O(\log n)$ times to get the aggregate error down to constant. Noting that each vertex is discovered once only (i.e., $\sum_v n_v = O(n)$) and that $\sum_v d_v = O(m)$, the full quantum time complexity is $O(\sum_v \sqrt{d_v n_v} \log d_v \log n) = O(\sqrt{nm} \log^2 n)$, by the Cauchy-Schwarz inequality.



(a) Initialization. (b) Phase 1. (c) Phase 2.

Fig. 2: Finding maximum matchings in bipartite graphs [24]. Bold, black arcs: edges in matching. Shaded, gray arcs: minimum-length augmenting paths.

BlossAug. This subroutine takes a bridge and performs a simultaneous double depth-first search (DDFS) from each vertex in the bridge until it finds two different exposed vertices. The subroutine ensures that a given vertex can only take part in at most one of the two DFSs. In Phase 1 of the example (Fig. 2b), BlossAug(x_1, v_1) would find exposed vertices x_1 and v_1. In Phase 2 of the example (Fig. 2c), BlossAug(x_1, v_1) would find vertex v_2 in the DFS from x_1, and x_2 in the DFS from v_1. Since we are concerned only with bipartite graphs, which do not contain blossoms [24], the results of the DFSs are then concatenated to generate an augmenting path (e.g., augmenting path (x_1, v_1) for Phase 1 of the example, and augmenting path (v_2, x_1, v_1, x_2) for Phase 2).

Quantum algorithm. To accelerate BlossAug, Dörn use quantum search to speedup the DDFS [12]. Each DFS has a time complexity of $O(\sqrt{nm}\log^2 n)$. The derivation is similar to that for the BFS complexity of Search above.

For each augmenting path found by BlossAug, the algorithm extends the matching along the augmenting path (i.e., an edge in M is removed from M and an edge not in M is added to it) and marks the vertices in the path as 'visited' in the current phase. The marking of vertices ensures the augmenting paths found during a phase are disjoint (i.e., do not share a vertex). Once the set of bridges is empty, the phase is over and the next phase begins. Fig. 2 provides a trace of the algorithm, starting with matching $M = \emptyset$, extending this to $M = \{(x_1, v_1)\}$ in the first phase via bridge (x_1, v_1), and then to the maximum matching $M = \{(x_1, v_2), (v_1, x_2)\}$ in the second phase via bridge (x_1, v_1).

In the interest of space, the details pertaining to the vertex deletion subroutine in the algorithm of Dörn are not included here; however, the time complexity of the subroutine is the same as for Search and BlossAug. The time complexity of each phase is $O(\sqrt{nm}\log^2 n)$, because that is the aggregate time complexity of each subroutine (e.g., Search and BlossAug). Since the number of phases required is at most $O(\sqrt{|X|}) = O(\sqrt{n})$ [19], the complexity of the overall algorithm is $O(n\sqrt{m}\log^2 n)$ or, in terms of our graph properties, $O(|X|\sqrt{(|X| + |V|)|V|}\log^2 |V|)$, for a constant $\Omega(1)$ success probability [12]. The

Algorithm 2: RemoveEdges(G, M)

Data: Bipartite graph $G = (X, V, D)$ and matching M
Result: Set of edges to prune
$G_M \leftarrow$ DirectGraph(G, M);
$D_{\text{used}} \leftarrow$ FindSimplePaths(G_M) ; /* Set of 'used' edges */
$\mathcal{S} \leftarrow$ FindSCC(G_M); /* SCC for each vertex */
return IdentifyEdges$(G, M, D_{\text{used}}, \mathcal{S})$;

classical $O(m\sqrt{n})$-time algorithm of MV, in terms of our graph properties, has time complexity of $O(|X|\sqrt{|X|}|V|)$, indicating an improvement by a factor of $\sqrt{|X||V|/(|X|+|V|)}$, up to polylogarithmic terms.

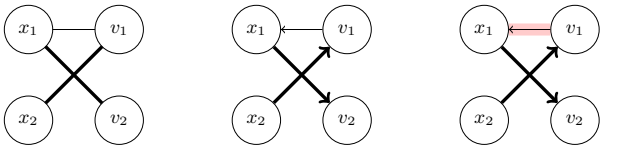
3.3 Subroutine: RemoveEdges

If a maximum matching is found such that $|M| = |X|$, Algorithm 1 proceeds to initiate the RemoveEdges subroutine with graph G and matching M as input (Fig. 3a). The steps of the subroutine are detailed in Algorithm 2.

From Berge [4], an edge belongs to some maximum matching if and only if, for an arbitrary given maximum matching, it belongs to either an even-length alternating path which begins at an exposed vertex, or to an even-length alternating cycle. If an edge does not satisfy Berge's property, it should be pruned. Instead of applying Berge's conditions directly, the problem has been previously translated into a search for edges in directed simple paths and strongly connected components (SCC) in a directed transformation of the graph [32,36].

The input to the RemoveEdges subroutine is the variable/value graph G and a matching M (found with FindMaximumMatching). In DirectGraph the edges in G are directed depending upon whether or not they are in matching M, producing directed graph G_M. Edges in the matching are directed from variables to values ('right-facing') and the remaining edges from values to variables ('left-facing'). For the running example, this is illustrated in Fig. 3b. The output of the subroutine is the set of edges to prune. The RemoveEdges subroutine has classical time complexity $O(m)$ stemming from three lower-level subroutines: FindSimplePaths, FindSCC, and IdentifyEdges. We provide an overview of these subroutines and comment on their quantum analogs.

FindSimplePaths. To satisfy the first condition of Berge's property, it suffices to find all edges present in at least one directed simple path starting at an exposed vertex. This is achieved by a BFS starting collectively at the exposed vertices, marking each edge considered as 'used'. This will output a set of edges D_{used} with the label 'used'. The number of edges processed during this subroutine is $O(|D_{\text{used}}|)$, therefore giving overall complexity $O(|D_{\text{used}}|)$, which is on the order of m in the worst case. When there are only a few edges to mark as used, the run time of this step can of course be significantly less than $O(m)$. In Fig. 3b, there are in fact no exposed vertices in G_M, and no BFS is even initiated.



(a) Input. (b) Directed graph. (c) Edges to remove.

Fig. 3: Removing edges. Bold, black arcs: edges in matching. Shaded, red arcs: edges removed.

Quantum algorithm. Since the classical algorithm strictly takes time linear in the output size, $O(|D_{\text{used}}|)$, there is no possible asymptotic speedup (quantum or otherwise). This step is therefore implemented with a classical BFS.

FindSCC. To satisfy the second condition of Berge's property, we compute the SCC in G_M. A directed graph is *strongly connected* if there is a path between all pairs of vertices in the graph. A *strongly connected component* (SCC) of a directed graph is a maximal strongly connected subgraph. Classically, the SCCs can be computed in time $O(n+m)$ with Tarjan's algorithm [34].

Tarjan's algorithm is a modified DFS. When a node is discovered in the DFS, it is put on a stack, and obtains index and low-link values, which are initially equal to infinity and updated during the search. On the backtrack, low-link values can be updated: of all forward neighbors on the stack, update the current low-link value to the minimum index value. After this completes, the low-link value of a vertex is the SCC to which it belongs.

This can be achieved by looping over all neighbors at a vertex v where there are two possible outcomes: i) if the neighbor w has not been discovered, advance the DFS to w, ii) if w is on the stack (i.e., already discovered), update the current low-link value of v to the minimum of v's low-link value, and w's index value. Upon looping over all neighbors, if v's low-link is equal to its index value, then this value is the SCC to which it belongs. Moreover, all nodes with this low-link value make up the SCC, and these can be removed from the stack.

FindSCC labels the SCC to which each vertex belongs. The directed graph G_M illustrated in Fig. 3b contains four trivial SCCs.

Quantum algorithm. Existing work has produced quantum algorithms for determining if a graph is strongly connected [14], noting that an adaptation of the approach can yield the identification of SCCs. Here, we describe such an adaptation based directly on Tarjan's algorithm, observing that it conducts essentially two searches at each step. In particular, from a vertex one needs to: i) find an undiscovered neighbor, and ii) find the minimum index value over the neighbors. While backtracking through the DFS, one can perform a search over the index values of all forward neighbours. The complexity of implementing quantum searches is the same as quantum DFS, plus the cost to perform a quantum minimum finding at each node which is $O(\sqrt{d_v}\log d_v)$ [15]. Overall the quantum time complexity is therefore $O(\sqrt{nm}\log^2 n)$, with each search repeated $O(\log n)$ times. In addition to the graph and stack QRAM data structures to perform the DFS, one also needs to maintain a QRAM data structure for the

index values of each vertex, which is used in the quantum minimum finding.

IdentifyEdges. The output of FindSimplePaths and FindSCC is, respectively, a set of edges $D_{\text{used}} \subseteq D$ marked as 'used', and the SCC to which each vertex belongs. IdentifyEdges identifies the set of edges to be removed, satisfying three conditions: i) the edge is not in D_{used}, ii) the edge is not between vertices in the same SCC, and iii) the edge is not in the current matching M. The set R of edges to remove is easy to construct by iterating over all edges and checking condition i-iii), giving a complexity of $O(m)$. In the context of our example, since $D_{\text{used}} = \emptyset$, and all SCCs are trivial, the subroutine returns $G \setminus M$, namely the edge (x_1, v_1) as illustrated in Fig. 3c.

Quantum algorithm. A quantum search can be used to find the edges R that need to be removed. From each variable vertex $x \in X$, a quantum search over the d_x incident edges (of the original, undirected graph) can determine which need to be removed, subject to the three criteria as in the classical version of IdentifyEdges. If there are r_x edges to be removed, where $\sum_x r_x = |R|$, the quantum time complexity is $O(\sqrt{r_x d_x} \log d_x)$ for each x, resulting in a total of $O(\log n \sum_x \sqrt{r_x d_x} \log d_x) = O(\sqrt{m|R|} \log^2 n)$, with each search repeated $O(\log n)$ times. To perform such a quantum search, QRAM access to the graph is required, as is QRAM access to the classical data: M, D_{used}, and the SCCs. Each of these can be set up prior to and maintained during the execution of the above subroutines without changing the overall complexity of RemoveEdges.

Up to logarithmic factors and for constant error probability, the time complexity of quantum RemoveEdges is $O(|D_{\text{used}}| + \sqrt{mn} + \sqrt{m|R|})$, reflecting the three lower-level subroutines. In the worst case this is $O(m)$, occurring when $|D_{\text{used}}| = O(m)$ and/or $|R| = O(m)$, and equivalent to the classical runtime. In cases where $|R| = O(n)$ (or lower), and $|D_{\text{used}}| = O(\sqrt{mn})$ (or lower), the full complexity is $O(\sqrt{mn})$. This upper bound on the quantum runtime improves over the $O(m)$ classical runtime by a factor of up to $\sqrt{m/n}$, or $\sqrt{|X||V|/(|X| + |V|)}$.

4 Extensions to the global cardinality constraint

The global cardinality constraint (`gcc`) is an extension of the `alldifferent` constraint, commonly used in scheduling, rostering, and timetabling problems [31].

Definition 7 (`gcc` constraint). *Given a set of variables $X = \{x_1, \ldots, x_n\}$, a set of values $V = \{v_1, \ldots, v_m\}$, and a set of cardinality bounds $\Delta = \{\delta_1, \ldots, \delta_m\}$, where each $\delta_i \in \Delta$ is defined by $[\ell_i, u_i]$, the constraint $\texttt{gcc}(X, V, \Delta)$ requires that value v_i take place in the solution between ℓ_i and u_i times, inclusively.*

State-of-the-art classical `gcc` filtering employs a $O(|X|\sqrt{|X|}|V|)$-time algorithm for achieving domain consistency [31], leveraging previous work for `alldifferent` [32]. The first stage of the algorithm enforces the domain consistency of `gcc` when all cardinality intervals are fixed to u_i, while the second

stage enforces domain consistency when the cardinality is fixed to ℓ_i, following a previous result that this is sufficient for the domain consistency of `gcc` [30].

For each stage, the classical filtering algorithm constructs a bipartite variable/value graph, $G = (X, V, D)$. Then, a capacity, $cap(x_i) = 1$, is associated with each variable node, and $cap(v_i) \geq 0$, with each value node. A matching in this graph is a subset of edges such that no more than the capacity of a given node is adjacent to that node. The algorithm then finds matchings of maximum cardinality for $cap(v_i) = u_i$ (first stage) and for $cap(v_i) = \ell_i$ (second stage). To do this, the algorithm of HK [19] can be used on an augmented graph, G', where value nodes are duplicated $cap(v_i)$ times, and the capacity of each node in G' is set to one; however, the complexity with this naive implementation will scale with the number of edges in the augmented graph. Instead, Quimper et al. describe an alteration of HK that runs on G by ensuring that in the DFS each free vertex v is visited at most $c(v) - d_M(v)$ times, where in each phase, $d_M(v)$ is the number of edges in the current matching M adjacent to node v [31]. This ensures the complexity is bounded by the number of edges in G, yielding an $O(|X|\sqrt{|X|}|V|)$ algorithm. Pruning the domains using Tarjan's algorithm [34], with the matchings at each stage, is sufficient to prune the domains for the domain consistency of `gcc` [30]. Given that Tarjan's algorithm is less computationally expensive than the maximum matching algorithm, the best-known overall classical complexity for achieving domain consistency for `gcc` is $O(|X|\sqrt{|X|}|V|)$.

Quantum algorithm. The filtering algorithm for `gcc` utilizes the same subroutines used in `alldifferent`, indicating that the detailed quantum subroutines can also be used to accelerate `gcc` filtering. For each of the two stages in the filtering algorithm, we can use the quantum-accelerated version of FindMaximumMatching. For this, the algorithm of Dörn [12] is modified, following the modification Quimper et al. made to HK, to ensure that, in each stage, free vertices can be visited $c(v) - d_M(v)$ times. This is done by associating an integer counter with each vertex, in QRAM. The quantum-accelerated RemoveEdges subroutine can then be applied to the maximum matchings found at both stages. The time complexity of the overall `gcc` filtering algorithm follows that of finding maximum matchings, which is $O(|X|\sqrt{(|X| + |V|)|V|}\log^2|V|)$.

5 Integration in backtracking search

As noted, the quantum search algorithms we employ have some probability of failure. For the purposes of the discussion here, the quantum FindMaximumMatching subroutine is extended such that the output is a valid matching with $|M| = |X|$ (as verified with a classical check) or *False*; the quantum subroutine is said to fail if such a matching exists but is not found. If we let ϵ be the failure probability of a given quantum subroutine, the acceptable value of ϵ depends on how the quantum subroutine is used in the search.

Exact method. When the quantum subroutine has perfect soundness (e.g., quantum FindMaximumMatching), then one approach is to require perfect

completeness.[5] This can be achieved by running the classical subroutine whenever the quantum one does not return a satisfying item (e.g., a valid matching of sufficient size). Let $t(n) = \mathsf{poly}(n)$ be the runtime of the classical subroutine. By repeating the quantum subroutine $O(\log(n))$ times, ϵ can be brought to $o(1/t(n))$, so that when a satisfying item exists, the expected cost of running the classical subroutine because the quantum one fails to find it is $o(1)$; therefore, at all nodes with a satisfying item, we get a quantum speedup on average. If no such item exists, the quantum subroutine will not return one. At nodes without a satisfying item, we run the classical algorithm, yielding no speedup for those nodes.

Bounded-error and heuristic methods. Alternatively, suppose we want the overall tree search to fail (i.e., not find a solution if one exists) with at most a constant probability $O(1)$ (which can be made arbitrarily small without changing the asymptotic runtimes). Let T_{Q} be the number of tree search nodes at which the quantum subroutine is run. It suffices then to repeat the quantum subroutine $O(\log T_{\mathrm{Q}})$ times to get $\epsilon = O(1/T_{\mathrm{Q}})$. However, if T_{Q} is exponential in n, this can overwhelm the quantum speedup. To preserve the speedup, we could restrict the tree search to calling the quantum subroutine only $T_{\mathrm{Q}} = \mathsf{poly}(n)$ times. In practice, tree search algorithms often only explore a polynomial number of nodes, either due to limited resources, or because that is sufficient for the problem instance at hand. In cases that the tree search explores more than a polynomial number of nodes, the quantum filtering can be disabled; in this case, quantum search benefits a large number of nodes early in the tree. The search algorithm can also be run in "heuristic mode", using the quantum subroutine at every node. In this case, the effect of subroutine failures on the overall tree search is strongly dependent on the tree search algorithm used and in general cannot be bounded.

6 Conclusions

We introduce quantum-accelerated filtering algorithms for global constraints, with subroutines for the `alldifferent` constraint and the global cardinality constraint (`gcc`). This work is intended to be a first step towards a larger effort of using quantum algorithms to accelerate constraint programming. In the long-term, quantum computing is a promising technology for approaching hard computational problems, and we demonstrate here that the constraint programming community is well-positioned to benefit from this progress.

Acknowledgements. K.B., J.M., and S.H. were supported by NASA Academic Mission Services (NAMS), contract number NNA16BD14C. K.B. was also supported by the NASA Advanced Exploration Systems (AES) program. B.O. was supported by a NASA Space Technology Research Fellowship. We thank the anonymous reviewers and Prof. J. Christopher Beck whose valuable feedback helped improve the final version of the manuscript.

[5] For an algorithm intended to find an item with a certain property, we say that the algorithm has perfect completeness if it always finds such an item, if one exists, and the algorithm has perfect soundness if it never returns an item without the property.

References

1. Alt, H., Blum, N., Mehlhorn, K., Paul, M.: Computing a maximum cardinality matching in a bipartite graph in time $O(n^{1.5}m \log n)$. Information Processing Letters **37**(4), 237–240 (1991)
2. Ambainis, A., Špalek, R.: Quantum algorithms for matching and network flows. In: Annual Symposium on Theoretical Aspects of Computer Science. pp. 172–183. Springer (2006)
3. Arute, F., Arya, K., Babbush, R., Bacon, D., Bardin, J.C., Barends, R., Biswas, R., Boixo, S., Brandao, F.G., Buell, D.A., et al.: Quantum supremacy using a programmable superconducting processor. Nature **574**(7779), 505–510 (2019)
4. Berge, C.: Graphs and hypergraphs. North-Holland (1973)
5. Berzina, A., Dubrovsky, A., Freivalds, R., Lace, L., Scegulnaja, O.: Quantum query complexity for some graph problems. In: International Conference on Current Trends in Theory and Practice of Computer Science. pp. 140–150. Springer (2004)
6. Booth, K.E.C., Do, M., Beck, J.C., Rieffel, E., Venturelli, D., Frank, J.: Comparing and integrating constraint programming and temporal planning for quantum circuit compilation. In: Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS) (2018)
7. Boyer, M., Brassard, G., Høyer, P., Tapp, A.: Tight bounds on quantum searching. Fortschritte der Physik: Progress of Physics **46**(4-5), 493–505 (1998)
8. Brandao, F.G., Svore, K.M.: Quantum speed-ups for solving semidefinite programs. In: 2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS). pp. 415–426. IEEE (2017)
9. Campbell, E., Khurana, A., Montanaro, A.: Applying quantum algorithms to constraint satisfaction problems. Quantum **3**, 167 (2019)
10. Devitt, S.J.: Performing quantum computing experiments in the cloud. Physical Review A **94**(3), 032329 (2016)
11. Dirac, P.A.M.: A new notation for quantum mechanics. In: Mathematical Proceedings of the Cambridge Philosophical Society. vol. 35, pp. 416–418. Cambridge University Press (1939)
12. Dörn, S.: Quantum algorithms for matching problems. Theory of Computing Systems **45**(3), 613–628 (2009)
13. Dumitrescu, E.F., McCaskey, A.J., Hagen, G., Jansen, G.R., Morris, T.D., Papenbrock, T., Pooser, R.C., Dean, D.J., Lougovski, P.: Cloud quantum computing of an atomic nucleus. Physical review letters **120**(21), 210501 (2018)
14. Dürr, C., Heiligman, M., Høyer, P., Mhalla, M.: Quantum query complexity of some graph problems. SIAM Journal on Computing **35**(6), 1310–1328 (2006)
15. Dürr, C., Høyer, P.: A quantum algorithm for finding the minimum. arXiv preprint quant-ph/9607014 (1996)
16. Gent, I.P., Miguel, I., Nightingale, P.: Generalised arc consistency for the alldifferent constraint: An empirical survey. Artificial Intelligence **172**(18), 1973–2000 (2008)
17. Giovannetti, V., Lloyd, S., Maccone, L.: Quantum random access memory. Physical review letters **100**(16), 160501 (2008)
18. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing. pp. 212–219 (1996)

19. Hopcroft, J.E., Karp, R.M.: An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. SIAM Journal on computing **2**(4), 225–231 (1973)
20. Ibarra, O.H., Moran, S.: Deterministic and probabilistic algorithms for maximum bipartite matching via fast matrix multiplication. Information Processing Letters **13**(1), 12–15 (1981)
21. Jiang, N., Pu, Y.F., Chang, W., Li, C., Zhang, S., Duan, L.M.: Experimental realization of 105-qubit random access quantum memory. npj Quantum Information **5**(1), 1–6 (2019)
22. Le Gall, F.: Powers of tensors and fast matrix multiplication. In: Proceedings of the 39th international symposium on symbolic and algebraic computation. pp. 296–303 (2014)
23. Matteo, O.D., Gheorghiu, V., Mosca, M.: Fault-tolerant resource estimation of quantum random-access memories. IEEE Transactions on Quantum Engineering **1**, 1–13 (2020)
24. Micali, S., Vazirani, V.V.: An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In: 21st Annual Symposium on Foundations of Computer Science (sfcs 1980). pp. 17–27. IEEE (1980)
25. Montanaro, A.: Quantum walk speedup of backtracking algorithms. arXiv preprint arXiv:1509.02374 (2015)
26. Montanaro, A.: Quantum speedup of branch-and-bound algorithms. Physical Review Research **2**(1), 013056 (2020)
27. Mucha, M., Sankowski, P.: Maximum matchings via Gaussian elimination. In: 45th Annual IEEE Symposium on Foundations of Computer Science. pp. 248–255. IEEE (2004)
28. Nannicini, G.: Fast quantum subroutines for the simplex method. arXiv preprint arXiv:1910.10649 (2019)
29. Peterson, P.A., Loui, M.C.: The general maximum matching algorithm of Micali and Vazirani. Algorithmica **3**(1-4), 511–533 (1988)
30. Quimper, C.G., Golynski, A., López-Ortiz, A., Van Beek, P.: An efficient bounds consistency algorithm for the global cardinality constraint. Constraints **10**(2), 115–135 (2005)
31. Quimper, C.G., López-Ortiz, A., Van Beek, P., Golynski, A.: Improved algorithms for the global cardinality constraint. In: International Conference on Principles and Practice of Constraint Programming. pp. 542–556. Springer (2004)
32. Régin, J.C.: A filtering algorithm for constraints of difference in CSPs. In: AAAI. vol. 94, pp. 362–367 (1994)
33. Rieffel, E.G., Polak, W.H.: Quantum computing: A gentle introduction. MIT Press (2011)
34. Tarjan, R.: Depth-first search and linear graph algorithms. SIAM Journal on Computing **1**, 146–160 (1972)
35. Van Apeldoorn, J., Gilyén, A., Gribling, S., de Wolf, R.: Quantum SDP-solvers: Better upper and lower bounds. Quantum **4**, 230 (2020)
36. Van Hoeve, W.J.: The alldifferent constraint: A survey. arXiv preprint cs/0105015 (2001)
37. Vazirani, V.V.: A simplification of the MV matching algorithm and its proof. arXiv preprint arXiv:1210.4594 (2012)
38. Yanofsky, N.S., Mannucci, M.A.: Quantum computing for computer scientists. Cambridge University Press (2008)
39. Zhang, X., Li, Q., Zhang, W.: A fast algorithm for generalized arc consistency of the alldifferent constraint. In: IJCAI. pp. 1398–1403 (2018)