An Elimination Algorithm for Functional Constraints

Yuanlin Zhang[1], Roland H.C. Yap[2],
Chendong Li[1], and Satyanarayana Marisetti[1]

[1] Texas Tech University, USA
{y.zhang, chendong.li, satyanarayana.marisetti}@ttu.edu
[2] National University of Singapore, Singapore
ryap@comp.nus.edu.sg

1 Introduction and preliminaries

Functional constraints are studied in Constraint Satisfaction Problems (CSP) using consistency concepts (e.g., [4, 1]). In this paper, we propose a new method — *variable substitution* — to process functional constraints. The idea is that if a constraint is functional on a variable, this variable in another constraint can be substituted using the functional constraint without losing any solution. We design an efficient algorithm to reduce, in $\mathcal{O}(ed^2)$, a general binary CSP containing functional constraints into a canonical form which simplifies the problem and makes the functional portion trivially solvable. When the functional constraints are also bi-functional, then the algorithm is linear in the size of the CSP.

We use the standard notations in CSP. Two CSPs are *equivalent* if and only if they have the same solution space. Throughout this paper, n represents the number of variables, d the size of the largest domain of the variables, and e the number of constraints in a problem. The composition of two constraints is defined as $c_{jk} \circ c_{ij} = \{(a, c) \mid \exists b \in D_j, \textit{such that } (a, b) \in c_{ij} \wedge (b, c) \in c_{jk}\}$. Composing c_{ij} and c_{jk} gives a new constraint on i and k.

A constraint c_{ij} is *functional* on j if for any $a \in D_i$ there exists at most one $b \in D_j$ such that $(a, b) \in c_{ij}$. c_{ij} is *functional* on i if c_{ji} is functional on i. When a constraint c_{ij} is functional on j, for simplicity, we say c_{ij} is functional by making use of the fact that the subscripts of c_{ij} are an ordered pair. In this paper, the definition of functional constraints is different from the one in [5, 4] where constraints are functional on each of its variables, leading to the following notion.

A constraint c_{ij} is *bi-functional* if c_{ij} is functional on both i and j. A bi-functional constraint is called *bijective* in [2] and simply functional in [4].

2 Elimination algorithm

Definition 1. *Consider a CSP (N, D, C), a constraint $c_{ij} \in C$ functional on j, and a constraint $c_{jk} \in C$. To* substitute *i for j in c_{jk}, using c_{ij}, is to get a new CSP where c_{jk} is replaced by $c'_{ik} = c_{ik} \cap (c_{jk} \circ c_{ij})$. The variable i is called the* substitution variable*.*

Property 1. Given a CSP (N, D, C), a constraint $c_{ij} \in C$ functional on j, and a constraint $c_{jk} \in C$, the new problem obtained by substituting i for j in c_{jk} is equivalent to (N, D, C).

Based on variable substitution, we can eliminate a variable from a problem so that no constraint will be on this variable (except the functional constraint used to substitute it).

Definition 2. *Given a CSP (N, D, C) and a constraint $c_{ij} \in C$ functional on j, to* eliminate j using c_{ij} *is to substitute i for j, using c_{ij}, in every constraint $c_{jk} \in C$ $(k \neq i)$.*

Given a functional constraint c_{ij} of a CSP (N, D, C), let C_j be the set of all constraints involving j, except c_{ij} and c_{ji}. The elimination of j using c_{ij} results in a new problem (N, D, C') where $C' = (C - C_j) \cup \{c'_{ik} \mid c'_{ik} = (c_{jk} \circ c_{ij}) \cap c_{ik}, c_{jk}, c_{ik} \in C\} \cup \{c'_{ik} \mid c'_{ik} = c_{jk} \circ c_{ij}, c_{jk} \in C, c_{ik} \notin C\}$.

In the new problem, there is only one constraint c_{ij} on j and thus j can be regarded as being "eliminated". By Property 1, the variable elimination preserves the solution space of the original problem.

Property 2. Given a CSP (N, D, C) and a functional constraint $c_{ij} \in C$, the new problem (N, D, C') obtained by the elimination of variable j using c_{ij} is equivalent to (N, D, C).

We now extend variable elimination to general CSPs with functional and non-functional constraints. The idea of variable elimination can be used to reduce a CSP to the following canonical functional form.

Definition 3. *A CSP (N, D, C) is in* canonical functional form *if for any constraint $c_{ij} \in C$ functional on j, the following conditions are satisfied: 1) if c_{ji} is also functional on i(i.e., c_{ij} is bi-functional), either i or j is not constrained by any other constraint in C; 2) otherwise, j is not constrained by any other constraint in C.*

In a canonical functional form CSP, *the functional constraints form disjoint star graphs.* A *star graph* is a tree where there exists a node, called the *center*, such that there is an edge between this center node and every other node. We call the variable at the center of a star graph, a *free variable*, and other variables in the star graph *eliminated variables.* The constraint between a free variable i and an eliminated variable j is functional on j, but it may or may not be functional on i. In the special case that the star graph contains only two variables i and j and c_{ij} is bi-functional, any one of the variables can be called a free variable while the other called an eliminated variable.

If a CSP is in canonical functional form, all functional constraints and the eliminated variables can be *ignored* when we try to find a solution for this problem. Thus, to solve a CSP (N, D, C) in canonical functional form whose non-eliminated variables are NE, we only need to solve a smaller problem

(NE, D', C') where D' is the set of domains of the variables NE and $C' = \{c_{ij} \mid c_{ij} \in C \text{ and } i, j \in NE\}$.

Any CSP with functional constraints can be transformed into canonical functional form by variable elimination using the algorithm in Fig. 1. Given a constraint c_{ij} functional on j, Line 1 of the algorithm substitutes i for j in all constraints involving j.

```
algorithm Variable-Elimination(inout $(N, D, C)$, out consistent) {
        $L \leftarrow N$;
        while ( There is $c_{ij} \in C$ functional on $j$ where $i, j \in L$ and $i \neq j$){
            // Eliminate variable $j$,
1.          $C \leftarrow \{c'_{ik} \mid c'_{ik} \leftarrow (c_{jk} \circ c_{ij}) \cap c_{ik}, c_{jk} \in C, k \neq i\} \cup (C - \{c_{jk} \in C \mid k \neq i\})$;
2.          $L \leftarrow L - \{j\}$;
            Revise the domain of $i$ wrt $c_{ik}$ for every neighbour $k$ of $i$;
            if ($D_i$ is empty) then { consistent $\leftarrow$ false; return }
        }
        consistent $\leftarrow$ true;
}
```

Fig. 1. A variable elimination algorithm to transform a CSP into a canonical functional form.

Theorem 1. *Given a CSP (N, D, C),* `Variable-Elimination` *transforms the problem into a canonical functional form in $\mathcal{O}(n^2d^2)$.*

A good ordering of the variables to eliminate will result in a faster algorithm. The intuition is that once a variable i is used to substitute for other variables, i itself should not be substituted by any other variable later.

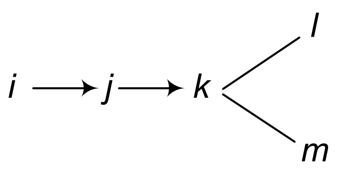

Fig. 2. The constraint graph of a CSP with functional constraints c_{ij} and c_{jk} and non-functional constraints c_{kl} and c_{km}.

Example Consider a CSP with functional constraints c_{ij} and c_{jk}. Its constraint graph is shown in Fig. 2 where a functional constraint is represented by an arrow. If we eliminate k and then j, we first get c_{jl} and c_{jm}, and then get c_{il} and c_{im}. Note that j is first used to substitute for k and later is substituted by

i. If we eliminate j and then k, we first get c_{ik}, and then get c_{il} and c_{im}. In this way, we reduce the number of compositions of constraints. □

Given a CSP $P = (N, D, C)$, P^F is used to denote its directed graph (V, E) where $V = N$ and $E = \{(i, j) \mid c_{ij} \in C$ and c_{ij} is functional on $j\}$.

Definition 4. *Given a directed graph (V, E), a sequence of the nodes of V is a* functional elimination ordering *if for any two nodes i and j, i before j in the sequence implies that there is a path from i and j. A* functional elimination ordering of a CSP problem *P is a functional elimination ordering of P^F.*

Given a directed graph G, a functional elimination ordering can be found by 1) finding all the strongly connected components of G, 2) modifying G by taking every component as one vertex with edges changed and/or added accordingly, 3) finding a topological ordering of the nodes in the new graph, and 4) replacing any vertex v in the ordering by any sequence of the vertices of the strongly connected component represented by v.

The algorithm `Linear-Elimination` in Fig. 3 first finds a functional elimination ordering (Line 1). Line 4 and 6 are to *process* all the variables in O. Every variable i of O is *processed* as follows: i will be used to substitute for all the variables *reachable* from i *through constraints that are functional in C^0 and still exist in the current C*. Those constraints are called *qualified* constraints. Specifically, L initially holds the immediate reachable variables through qualified constraints (Line 8). Line 9 is a loop to eliminate all variables reachable from i. The loop at Line 11 is to eliminate j using i from the current C. In this loop, if a constraint c_{jk} is qualified (Line 14), k is reachable from i through qualified constraints. Therefore, it is put into L (Line 15).

Theorem 2. *Given a CSP problem, the worst case time complexity of* `Linear-Elimination` *is $O(ed^2)$ where e is the number of constraints and d the size of maximum domain in the problem.*

For the algorithm `Linear-Elimination`, we have the following nice property.

Theorem 3. *Consider a CSP with both functional and non-functional constraints. If there is a variable of the problem such that every variable of the CSP is reachable from it in P^F, the satisfiability of the problem can be decided in $\mathcal{O}(ed^2)$ using* `Linear-Elimination`*.*

For a problem with the property given in the theorem above, its canonical functional form becomes a star graph. So, any value in the domain of the free variable is extensible to a solution if we add (arc) consistency enforcing during `Linear-Elimination`. The problem is not satisfiable if a domain becomes empty during the elimination process . In contrast to our algorithm `Linear-Elimination`, using an arc consistency based (bi-)functional algorithm [4] or view based [3] implementations of propagators for functional constraints may not be able to achieve global consistency in general.

```
algorithm Linear-Elimination(inout $(N, D, C)$) {
 1. Find a functional elimination ordering $O$ of the problem;
 2. Let $C^0$ be $C$; any $c_{ij}$ in $C^0$ is denoted by $c^0_{ij}$;
 3. For each $i \in N$, it is marked as not eliminated;
 4. while ($O$ is not empty) {
          Take and delete the first variable $i$ from $O$;
 6.       if ($i$ is eliminated) continue;
 8.       $L \leftarrow \{j \mid (i, j) \in C$ and $c^0_{ij}$ is functional$\}$;
 9.       while ($L$ not empty) {
              Take and delete $j$ from $L$;
 11.          for any $c_{jk} \in C - \{c_{ji}\}$ { // Substitute $i$ for $j$ in $c_{jk}$;
                  $c'_{ik} \leftarrow c_{jk} \circ c_{ij} \cap c_{ik}$;
                  $C \leftarrow C \cup \{c'_{ik}\} - \{c_{jk}\}$;
 14.              if ($c^0_{jk}$ is functional) then
 15.                  $L \leftarrow L \cup \{k\}$;
              }
 16.          Mark $j$ as eliminated;
          } // loop on $L$
   } // loop on $O$
 } // end of algorithm
```

Fig. 3. A variable elimination algorithm of complexity $O(ed^2)$.

3 Conclusion

We have introduced a variable substitution method to reduce a problem with both functional and non-functional constraints. Compared with the previous work on bi-functional and functional constraints, the new method is not only conceptually simple and intuitive but also reflects the fundamental property of functional constraints. Our experiments (not included here) also show that variable elimination can significantly improve the performance of a general solver in dealing with functional constraints.

References

1. David, P.: When functional and bijective constraints make a CSP polynomial. In *Intl. Joint Conf. on Artificial Intelligence.* (1993) 224–229
2. David, P.: Using pivot consistency to decompose and solve functional CSPs. *Journal of Artificial Intelligence Research* **2** (1995) 447–474.
3. Schulte, C., Tack, G.: Views and iterators for generic constraint implementations. In *Constraint Solving and Constraint Logic Programming.* (2005) 118–132
4. Van Hentenryck, P., Deville, Y., Teng, C.M.: A generic arc-consistency algorithm and its specializations. *Artificial Intelligence* **58** (1992) 291–321
5. Zhang, Y., Yap, R.H.C., Jaffar, J.: Functional elimination and 0/1/all constraints. In *Natl. Conf. on Artificial Intelligence.* (1999) 275–281