

# Incrementally Solving Functional Constraints

Yuanlin Zhang and Roland H.C. Yap

School of Computing, National University of Singapore

3 Science Drive 3, 117543, Singapore

Email: {zhangyl, ryap}@comp.nus.edu.sg

## Introduction

Binary functional constraints represent an important constraint class in Constraint Satisfaction Problems (CSPs). They have been studied in different contexts [for example (van Hentenryck et al. 1992; Kirousis 1993; van Beek and Dechter 1995; David 1995; Zhang et al. 1999)]. Functional constraints are also a primitive in Constraint Programming (CP) systems. In a CP system (Jaffar and Maher 1994), constraints are incrementally added to and removed from its constraint store which can be modeled as a CSP. The success of CP systems illustrates the need to have efficient incremental CSP algorithms. Existing work on functional constraints deals mainly with static CSPs where all constraints are known a priori. We show that an incremental CSP with pure functional constraints can be solved in *almost* the same time complexity as a static one. To solve more constraints (not only pure functional constraints) in a *mixed* CSP with both functional and non-functional constraints, we propose an algorithm with complexity comparable to the cost of enforcing arc consistency.

**Notation** A *Constraint Satisfaction Problem* ( $N, D, C$ ) consists of a finite set of variables  $N = \{1, \dots, n\}$ , a set of domains  $D = \{D_1, \dots, D_n\}$ , where  $D_i$  is the set of values that  $i$  can take, and a set of constraints  $C = \{c_{ij} \mid i, j \in N\}$ , where each constraint  $c_{ij}$  is a binary relation between variables  $i$  and  $j$ . We require that  $(x, y) \in c_{ij}$ , iff  $(y, x) \in c_{ji}$ . It is convenient to view a CSP as a graph whose nodes are variables and edges are constraints. When we say a CSP is *solved*, we mean either a solution of the CSP is found or it is unsatisfiable. Throughout this paper,  $n$  denotes the number of variables,  $d$  the size of the largest domain,  $e$  the number of constraints. A constraint  $c_{ij}$  is *functional* iff for all  $v \in D_i$  (respectively  $w \in D_j$ ) there exists at most one  $w \in D_j$  (respectively  $v \in D_i$ ) such that  $c_{ij}(v, w)$ . This definition means that  $c_{ij}$  is a function from  $D_i$  to  $D_j$  and vice versa. A CSP is *functional* if all its constraints are functional. Otherwise it is *mixed*. A *functional*

*block* of a mixed CSP is the maximal connected subgraph of the graph of the CSP which has a spanning tree containing only functional constraints. For example, Figure 1(a) is a functional CSP and Figure 1 (b) is a functional block.

## Solving Incremental Functional CSP

Arc consistency can be enforced on a static functional CSP in time  $\mathcal{O}(ed)$  (van Hentenryck et al. 1992). (Zhang et al. 1999) shows that it can be solved in the same time complexity by introducing *variable elimination*. In this section we further show that an incremental functional CSP can be solved in almost the same time complexity.

An obvious application of the static elimination algorithm will lead to an algorithm with  $\mathcal{O}(e^2d)$  time. Here we want a more incremental and efficient algorithm. A key observation is that it is not necessary to apply the complete elimination algorithm every time a new constraint is added when solving the system. It is only necessary to do so when the newly added constraint forms a circuit with those already in the system. Consider example (a) in Figure 1. There are

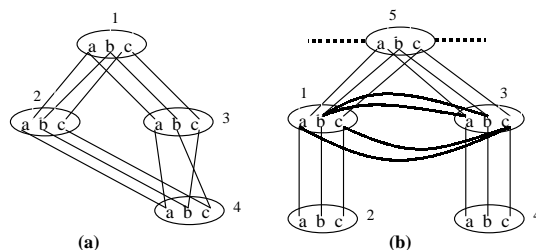


Figure 1: (a) A Functional CSP; (b) A Functional Block

four variables  $\{1, 2, 3, 4\}$  with the domain  $\{a, b, c\}$  in the system. Constraints are added into the system in the following order. Firstly,  $c_{12} = \{(a, a), (b, b), (c, c)\}$ . We need to mark a variable, say 2, with respect to  $c_{12}$  as *eliminated*. Then we mark 1 as *free* and *revise* the domain of 1 with respect to  $c_{21}$ , i.e. remove values in  $D_1$  which are not allowed by  $c_{21}$ . Secondly,  $c_{34} = \{(a, a), (b, c), (c, b)\}$ . Mark 4 as *eliminated* and 3 as *free*, revising  $D_3$ . Thirdly,  $c_{13} = \{(a, a), (b, b), (c, c)\}$ .

Both 1 and 3 are free variables. The property we want is that in any connected component of the constraint graph, there is only one free variable. Thus, we keep, say 1, as free and eliminate 3. Then revise  $D_1$  with respect to  $c_{31}$ . So far, no real elimination has occurred but we can verify that there is a solution for the current CSP since  $D_1$  is not empty. Lastly,  $c_{24} = \{(a, a), (b, c), (c, b)\}$ . But both variables 2 and 4 have been eliminated. Here we want to ensure that a new constraint is only on free variables and not eliminated ones. Since an eliminated variable is marked with respect to a particular constraint, we can follow this until a free variable is found. From 4 we get 3 and from 3 we get 1 which is free. Elimination also occurs during this tracing. A new constraint  $c_{14} = \{(a, a), (b, c), (c, b)\}$  is obtained by composing  $c_{13}$  and  $c_{34}$ , and 4 is marked as eliminated with respect to  $c_{14}$ . Discard  $c_{34}$  from the system. Similarly we trace 2 to 1. The fact that 2 and 4 share the same free variable 1, implies a circuit is formed. We can further eliminate 2 and 4 (compose  $c_{12}$ ,  $c_{24}$ , and  $c_{41}$ ) in sequence resulting in a new unary constraint  $c_{11} = \{(a, a), (b, c), (c, b)\}$ . We cannot assign variable  $b$  and  $c$  simultaneously to variable 1. Revising  $D_1$  with respect to  $c_{11}$  gives  $D_1 = \{a\}$ . Discard constraint  $c_{24}$  and  $c_{11}$  from the system. Now the system contains  $\{c_{12}, c_{14}, c_{13}\}$  and is satisfiable.

The above example of incremental solving can be implemented efficiently by disjoint set union algorithms (Tarjan 1975):

**Theorem 1** *Given at time  $t$ , a total of  $e$  constraints are added into an incremental functional CSP which has  $n$  variables. Using disjoint set union with union by rank and path compression, the satisfiability of the incremental system can be determined in worst case time complexity of  $\mathcal{O}(eda(2e, n))$ , where  $\alpha$  is the inverse Ackerman function.*

## Solving the Functional Block in a Mixed CSP

In a mixed CSP, the algorithm described in previous section does not prune as much as it could given the presence of non-functional constraints. Consider the example (b) from Figure 1. There are variables  $\{1, 2, 3, 4, 5, \dots\}$  with domain  $\{a, b, c\}$  in the CSP. Constraints are added into the system as follows. Firstly,  $c_{12} = \{(a, a), (b, b), (c, c)\}$ . Secondly,  $c_{34} = \{(a, a), (b, c), (c, b)\}$ . They are processed as before. Thirdly, a non-functional constraint,  $c_{13} = \{(a, c), (b, b), (b, a), (c, c)\}$ , so ignore. Fourthly, some other constraints on 5 are added. Fifthly,  $c_{15} = \{(a, a), (b, b), (c, c)\}$ . Because of the other functional constraints on 5, we mark 5 as free and 1 as eliminated. Lastly,  $c_{53} = \{(a, a), (b, b), (c, c)\}$ . Mark 5 as free and 3 as eliminated.

In this example, nothing is pruned although  $c_{13}$  could have been actively used to prune  $D_5$ . To get better pruning, we propose an algorithm which eliminates a vari-

able as soon as possible. Consider the same example again.

Firstly,  $c_{12}$ . Revise  $D_1$  with respect to  $c_{21}$ . Secondly,  $c_{34}$ . Repeat first step. Thirdly,  $c_{13}$ . Fourthly, some other constraints on 5. Fifthly,  $c_{15}$ . Eliminate 1 immediately. As a consequence two new constraints are added. The first is  $c_{52} = \{(a, a), (b, b), (c, c)\}$ , the composition of  $c_{51}$  and  $c_{12}$ . The second is  $c_{53} = \{(a, c), (b, b), (b, a), (c, c)\}$  (composition of  $c_{51}$  and  $c_{13}$ ). Revise  $D_5$  with respect to the two new constraints. Discard  $c_{12}$  and  $c_{13}$ . Sixthly,  $c'_{53} = \{(a, a), (b, b), (c, c)\}$ . Eliminate 3. Add  $c_{54} = \{(a, a), (b, b), (c, c)\}$  (composition of  $c'_{53}$  and  $c_{34}$ ) and  $c_{55} = \{(a, c), (b, b), (b, a), (c, c)\}$  (composition of  $c_{53}$  and  $c'_{35}$ ).  $D_5$  is revised to be  $\{b, c\}$ . Discard  $c_{53}$  (non-functional) and  $c_{55}$ . Now the final system has constraints  $\{c_{51}, c_{52}, c_{53}, c_{54}\}$  and is satisfiable.

**Theorem 2** *Given at time  $t$ , a total of  $e$  constraints are added into an incremental functional CSP which has  $n$  variables. By appropriate choice of elimination variable, any functional block of a CSP can be solved in a worst case time complexity of  $\mathcal{O}(ed^2 \log e)$ .*

When adding a functional constraint, the rule is that we choose to eliminate the variable with more constraints incident on it.

## Discussion and Conclusion

The most related work in CSP is bucket elimination (Dechter 1999), which is designed for a general CSP (NP-complete) and thus the complexities of corresponding algorithms are high. It may not directly lead to efficient algorithms for both static and incremental functional systems. The effort here may motivate work on more efficient bucket elimination algorithms for special classes of constraints.

Two algorithms are proposed to solve functional constraints in an incremental system. They are especially useful for CP systems (Jaffar and Maher 1994). When applied to a CP system, the first algorithm is more efficient while the second may achieve more pruning than the first. The choice of the two algorithms in a CP system will depend on the tradeoff between efficiency and pruning ability.

## References

- van Beek, P. and Dechter, R. 1995. On the Minimality and Global Consistency of Row-Convex Constraint Networks. *Journal of the ACM*, 42(3):543–561.
- David, P. 1995. Using Pivot Consistency to Decompose and Solve Functional CSPs. *Journal of Artificial Intelligence Research*, 2:447–474.
- Dechter, R. 1999. Bucket Elimination: A unifying framework for reasoning. *Artificial Intelligence* 113:41–85.
- Jaffar, J. and Maher, M.J. 1994. Constraint Logic Programming. *Journal of Logic Programming* 19/20:503–581.

- van Hentenryck, P., Deville, Y., and Teng, C.M. 1992. A Generic Arc-Consistency Algorithm and its Specializations. *Artificial Intelligence* 58:291–321.
- Kirousis, L.M. 1993. Fast Parallel Constraint Satisfaction. *Artificial Intelligence* 64:147–160.
- Tarjan, R.E. 1975. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):146–160.
- Zhang, Y., Yap, R.H.C., and J. Jaffar 1999. Functional Elimination and 0/1/All Constraints. *Proceedings of the 16th AAAI*:275–281.