Functional Elimination and 0/1/All Constraints

Yuanlin Zhang, Roland H.C. Yap and Joxan Jaffar *
{zhangyl, ryap, joxan}@comp.nus.edu.sg

Abstract

We present new complexity results on the class of 0/1/All constraints. The central idea involves functional elimination, a general method of elimination whose focus is on the subclass of functional constraints. One result is that for the subclass of "All" constraints, strong n-consistency and minimality is achievable in $\mathcal{O}(en)$ time, where e, n are the number of constraints and variables. The main result is that we can solve 0/1/All constraints in $\mathcal{O}(e(d+n))$ time, where d is the domain size. This is an improvement over known results, which are $\mathcal{O}(ed(d+n))$. Furthermore, our algorithm also achieves strong n-consistency and minimality.

1. Introduction

Constraint Satisfaction Problem(s) (CSP) are known to be NP-complete in general (Mackworth 1977). There are two approaches for attacking the computational intractability. One way is to identify those tractable class of problems by suitable restrictions so that it can be solved in polynomial time. A restriction is on the topological structure of the CSP constraint network, one example is (Freuder 1982). Another restriction is to exploit semantic properties of special classes of constraints, examples of this approach are (Dechter 1992; Van Beek and Dechter 1995; Cooper et al. 1994). (Van Beek and Dechter 1995) introduces the class of row-convex constraints which under some conditions can be solved in polynomial time. (Cooper et al. 1994) identifies a class of 0/1/All constraints, a special case of row-convex constraints, and proves that the class of problems generated by any set of constraints not contained in that class is NP-complete.

A different approach is to improve the efficiency of the basic step in searching the solution space of a CSP. Consistency techniques, especially arc-consistency, have been the method of choice for solving finite domain problems (Van Hentenryk 1989). Much effort has been made to find fast algorithms for arc-consistency. For a general CSP, we have AC-3 (Mackworth 1977), AC-4 (Mohr and Henderson 1986) which has an optimal worst-case time complexity $\mathcal{O}(ed^2)$, and AC-6 which provides a better space complexity and average time complexity while giving the optimal worst-case time complexity, where e is the number of constraints and d the size of the largest domain. In addition, many arc-consistency algorithms have been proposed for dealing with CSPs with special properties. The algorithms of interest here are: (Van Hentenryck et al. 1992) gives an arc-consistency algorithm for special constraints such as functional constraints and monotone constraints in time $\mathcal{O}(ed)$, and (Liu 1995) gives another algorithm to deal with increasing functional constraint where each functional constraint is only checked once. (Affane and Bennaceur 1996) and (Zhang 1998) also study functional constraints in CSP.

In this paper we investigate the class of 0/1/All constraints. These 0/1/All constraints, also called *implicational* constraints, represent a significant class of scene labeling problems (Kirousis 1993). The class of functional constraints, which arises frequently in practice (Van Hentenryck et al. 1992), is in fact a sublass of 0/1/All constraints. Using a central idea of functional elimination, which is a general method of elimination on functional constraints, we obtain new complexity results for 0/1/All constraints. First, we prove that for the subclass of "All" constraints, strong n-consistency and minimality is achievable in $\mathcal{O}(en)$ time, where e, n are the number of constraints and variables. The main result is that we can solve 0/1/All constraints in $\mathcal{O}(e(d+n))$ time, where d is the domain size. This is an improvement over known results, which are $\mathcal{O}(ed(d+n))$. Furthermore, our algorithm also achieves strong n-consistency and minimality.

The paper is organized as follows. We start with background material on consistency techniques. Next, we investigate the properties of functional constraints (section 3) and two-fan constraints (section 4). Section 5, gives the elimination method and an algorithm for solving mixed 0/1/All CSPs. We conclude with a discussion on related work and the application of elimination method in general CSP.

* School of Computing, National University of Singapore, Lower Kent Ridge Road, Singapore 119260

2. Preliminaries

Definitions on general CSP follow (Montanari 1974; Mackworth 1977; Freuder 1978; Freuder 1982).

Definition 1 *A* Constraint Satisfaction Problem *(N, D, C) consists of a finite set of variables $N = \{1, \cdots, n\}$, a set of domains $D = \{D_1, \cdots, D_n\}$, where $i \in D_i$, 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. For the problem of interest here, we require that $(x, y) \in c_{ij}$ if and only if $(y, x) \in c_{ji}$. There is always a graph $G = (V, E)$ associated with the CSP where $V = N$ and $E = \{(i, j) \mid \exists c_{ij} \in C\}$. A* solution *to a constraint satisfaction problem is an instantiation of the variables which satisfies all the constraints in the problem.*

Throught this paper, we will use n to represent the number of variables, d the size of the largest domain, C the set of constraints of the CSP, and e the number of constraints in C.

Definition 2 *A CSP is* k-consistent *if and only if given any instantiation of any $k-1$ variables satisfying all of the constraints among those variables, there exists an instantiation of any kth variable such that the k values taken together satisfy all of the relations among the k variables. A CSP is* stongly k consistent *if and only if it is i-consistent for all $i \leq k$. A CSP is* minimal *if each pair of values allowed by each of the constraints is a part of a solution of the CSP.*

Well known consistency techniques like arc and path consistency correspond to strong two and three-consistency.

We now recall definitions of constraints with some special properties.

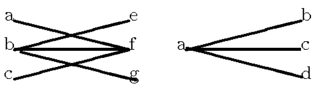
Definition 3 *(Cooper et al. 1994) A constraint, c_{ij}, is a* directed 0/1/All constraint *if for each value $x \in D_i$, c_{ij} satisfies the following:*

1. *for any value $y \in D_j$, $(x, y) \notin c_{ij}$, or*
2. *for any value $y \in D_j$, $(x, y) \in c_{ij}$, or*
3. *there is a unique value $y \in D_j$, $(x, y) \in c_{ij}$.*

A constraint is called functional *if either condition 1 or condition 3 is satisfied. A* two-fan *constraint, also called an "All" constraint, c_{ij} is a constraint where there exists $x \in D_i$ and $y \in D_j$ such that $c_{ij} = (x \times D_j) \cup (y \times D_i)$. A* fan-out *constraint is a constraint c_{ij} such that $\exists x \in D_i$ and $\forall y \ (x, y) \in c_{ij}$.*

Property 1 *(Cooper et al. 1994) After enforcing arc-consistency on 0/1/All constraints, any 0/1/All constraint is either a trivial relation, a bijective function or a two-fan constraint. A trivial relation c_{ij} is either empty or $D_i \times D_j$.*

An example of two-fan (left) and fan-out (right) constraints are illustrated as follows:



Property 2 *(Cooper et al. 1994) The set of 0/1/All constraints is closed under the operations involved in path consistency:*

1. *Intersection of constraints.*
2. *Composition, $\circ$, where $c_{ij} \circ c_{jk} = \{(p, q) \mid \exists r \in D_j$, such that $(p, r) \in c_{ij} \wedge (r, q) \in c_{jk}\}$.*

Definition 4 *(Van Beek and Dechter 1995) A binary relation c_{ij} represented as a $(0, 1)$-matrix is* row convex *if and only if in each row all of the ones are consecutive; that is, no two ones within a single row are separated by a zero in that same row.*

Both functional and 0/1/All constraints are row convex. For row convex constraints there is the result:

Theorem 1 *(Van Beek and Dechter 1995) For a path-consistent CSP, if there exists an ordering of the domains $D_1, \cdots, D_n$ such that all constraints are row convex, the CSP is minimal and strongly n-consistent.*

It is obvious that a path consistency enforcing algorithm will make the 0/1/All constraint system minimal by theorem 1 and property 1 and 2, and thus the problem is solved. However, the complexity of a typical path algorithm is high, such as $\mathcal{O}(n^3d^3)$ in (Mohr and Henderson 1986). In this paper, we obtain more efficient algorithms.

3. The O (one) algorithm

We first present an algorithm for a CSP (N, D, C) which contains only functional constraints (functional CSP for short) and then give the analysis of certain properties of these constraints.

For the sake of simplicity and clarity, we assume the graph of the CSP is connected without loss of generality.

The algorithms are shown in figure 1 and figure 2.

Algorithm $\mathbf{O}((N, D, C))$ {
  $\forall i \forall x, \ \ x \in D_i \ \ x.touched \leftarrow$ **false**;
  Select any variable $i \in N$
  for each value x of i {
    $x.touched \leftarrow$ **true**;
    $x.delete \leftarrow$ **false**;
    if not Propagate(x, i) **then** $x.delete \leftarrow$ **true**;
  }
  $\forall i \forall x, x \in D_i$
    if (**not** $x.touched$) **or** ($x.coordinate$)$.delete$ **then**
      remove x from i;
}

Figure 1: O-algorithm for functional CSP

The O-algorithm uses an adaptation of brute force searching (Garey and Johnson 1979; Cooper et al. 1994) which takes advantage of the properties of functional constraints. In contrast, other known algorithms (Van Hentenryck et al. 1992) etc. are based on arc consistency where the emphasis is on finding those values

```
Function Propagate (in x, i) {
  L ← {(x, i)};
  for each j ∈ N x_j ← null;
  x_i ← x;
  repeat
    Delete first element (y, j) from L;
    for each c_jk
      if ∃z such that (y, z) ∈ c_jk then
        if  x_k = null then {
          z.coordinate ← x;
          z.touched ← true;
          x_k ← z;
          L ← L ∪ {(z, k)};
        } end else if x_k ≠ z then return false;
  until (L = ∅);
  return true;
}
```

Figure 2: Propagate algorithm for functional CSP

which can be immediately removed by checking only a single constraint. The direct brute force searching method here gives a simpler algorithm. The intuition behind the O-algorithm is that if at some point in the search we cannot continue because of inconsistency, we simply restart at the initial starting point since the path propagated between the start point and the current failure point is unique in a functional CSP.

Definition 5
Given a functional CSP (N, D, C), its value graph *is $G = (V, E)$ where $V = \{(D_i, x) : x \in D_i, i \in N\}$ and $E = (D_i, x), (D_j, y)) \mid \exists i, j$ such that $(x, y) \in c_{ij}\}$. A* spanning tree *of G with respect to (D_i, x) is the maximal sub-graph $\bar{G}$ satisfying:*
1. $(D_i, x) \in \bar{G}$,
2. $\bar{G}$ is a tree, and
3. $(D_j, y) \in \bar{G}$ and $(D_k, z) \in \bar{G}$ implies $y = z$.
A value graph G is stable wrt. *(D_i, x) if all spanning trees of G wrt. (D_i, x) have the same vertices. Here, we may say that the spanning tree is* unique. *Finally, a spanning tree is* complete *if it has $n - 1$ edges.*

Property 3 *In a functional CSP, $x \in D_i$ is a part of a solution if and only if its value graph G wrt. (D_i, x), and any spanning tree is complete.*

Proof. Obviously, if the spanning tree is not complete, then x cannot be extended to a solution of the functional CSP. The proof of stability is by contradiction. Assume there are two different complete spanning trees G' and G'' wrt. (D_i, x), then $\exists (D_j, y) \in G'$ and $(D_j, z) \in G''$ where $y \neq z$. That means, according to the definition of functional constraint, when variable i takes x, variable j has to be both y and z, which is a contradiction. □

It is easy to see that if a spanning tree of $x \in D_i$ is not complete, or if it is not unique, then all values in possible spanning trees wrt. (D_i, x) will also be invalid. The x here is called the *coordinate* of all other nodes on the spanning tree and i is called the *origin*. In the O-algorithm, we associate the following attributes to any value $y \in D_j$:

- *y.touched* indicates if this value has been tried;
- *y.delete* indicates if the value should be removed;
- *y.coordinate* is the coordinate of y with regard to an origin i.

Theorem 2 *Given a functional CSP , the O-algorithm is correct and enforces the CSP to be minimal and strongly n-consistent. The complexity of O-algorithm is $\mathcal{O}(ed)$.*

Proof.
(1) In order to find all the solutions, the algorithm need only check one domain, say D_i, because if there is a solution for the CSP it must contain a value of D_i. The purpose of Propagate is to identify whether a spanning tree with regard to $x \in D_i$ is complete and unique. The **repeat** loop is to find the maximal spanning tree of x. The condition statement is to check the uniqueness of the spanning tree. The Propagate can only detect one spanning tree. Suppose the tree is not complete or unique, the question is how to delete the nodes of the other spanning trees? We use the property that all nodes of other spanning trees will never be visited again, or even if it is visited, its coordinate will labeled as deleted. That property is implemented by the variable *x.touched*.
(2) The minimal and strong n-consistency is immediate by the above proof.
(3) It is straightforward to show that the complexity of O-algorithm is $\mathcal{O}(ed)$. □

4. The A ("All") algorithm

In this section, we analyse two-fan constraints (also called "All" constraints) and give an algorithm for this class. Without loss of generality, we will assume that the CSP only contains "All" constraints.

For the ease of presentation, we introduce the following notations:

Definition 6 *Given a two-fan constraint c_{ij}, a* pivot *of c_{ij} with respect to i, is denoted by the notation p_i^{ij}. The pivot p_i^{ij} is defined to be the value $x \in D_i$ such that $\forall y \in D_j$ $(x, y) \in c_{ij}$. The* coordinate *of p_j^{ij} with respect to D_i is defined to be p_i^{ij}.*

A two-fan constraint c_{ij} can be simply represented by the two pivots $(p_i^{ij},\ p_j^{ij})$. The use of coordinate here is analogous to its use in functional constraints. In the A-algorithm the coordinate is the only value in D_i such that j can take any value; and furthermore for values other than the coordinate there is a unique choice in j. Thus, the role of coordinates in the A-algorithm is an adaptation of that in the O-algorithm.

Like the O-algorithm, the A-algorithm is also based on a search procedure. Before we present the algorithm, we will first highlight some important properties of two-fan constraints. We begin by recalling an important observation mentioned in (Cooper et al. 1994). Here we formalize it to emphasize its importance.

Definition 7 *Given a CSP (N, D, C), an instantiation of a set of variables $S \subseteq N$ is* separable, *if it satisfies all constraints among S, and any constraint $c_{ij} \in C$ between variables $i \in S$ and $j \in N - S$ allows j to take any value under the current instantiation of i.*

For a single two-fan constraint c_{ij}, it is immediate that the instantiation of i by the pivot p_i^{ij} is separable.

Proposition 1 *Given any CSP (N, D, C) and a separable instantiation of a set of variables. If the CSP has a solution, then the instantiation is part of some solution.*

The correctness of the above proposition is immediate. The usefulness of this proposition, is that after a separable instantiation is found, we can subtract out those variables and all constraints involved in at least one of those variables. and thus we get a smaller problem to work on (search). By continuing in this fashion, at the end, the combination of all the separable instantiations is a solution to the original problem. One task of the A-algorithm is to identify some set of variables whose instantiation is separable since the two-fan constraint gives a strong hint on how to achieve that goal.

The rationale for identifying the separable instantiations, by using some special properties of two-fan functions, is that a faster algorithm can be achieved. The identification step is achieved using the A-propagate procedure in a similar fashion to Propagate in the O-algorithm. It works as follows. First, select a starting variable and instantiate it to a value x. The next step is to try to instantiate its uninstantiated neighbor variables. For any uninstantiated k such that there exists $c_{ik} \in C$, we have two choices. In one case, we have that x is p_i^{ik}, and this stops the identification procedure along direction of c_{ik}. In the other case, by definition, we have a unique choice in D_k and thus we need repeat the propagation above to deal with the neighbors of k because in the direction of c_{ik} the instantiation has not yet been found to be separable. Finally we get a set of variables whose instantiation is separable. A trivial case is that the set of variables is N itself. One problem in the procedure is that the instantiation step for a variable may fail. Fortunately, this failure case only occurs when the instantiation step tries to set a variable to two different values which is a contradiction. In (Cooper et al. 1994), they simply return to the starting variable and select the next value available. However, there is a better and faster way for resolving the failure because of the following properties.

The values of D_i fall into two classes. One, called the *pivot class* P, contains all the pivots while the other, called the *nonpivot class* NP, contains all the other values.

Definition 8 *Given a two-fan CSP, a value $x \in i$ is* valid *if x is part of a solution of the CSP.*

Property 4 *Given a two-fan CSP and a variable i with domain D_i, we have for the*

- *NP class of D_i: if two of the values are valid, then any value will also be valid;*
- *P class of D_i: if three of the values are valid, then any value will also be valid.*

Proposition 2 *In the procedure of identifying the set of variables with separable instantiations, if there is a contradiction, then for the starting variable there are at most two valid values from the P class and no value from the NP class.*

Proof. It is obvious for NP class. For P class, only coordinates of the two contradicted values are possible. For all other values, contradiction still remains. □

The A algorithm is given in figures 3 and 4.

Algorithm A (**in** (N, D, C)) {
  Select any value $x \in D_i$ for any variable $i \in N$
  A-Propagate($x, i, N, M, consistent, p_1, p_2$);
  if not *consistent* **then** {
    A-Propagate($x, p_1, N, M, consistent, -, -$);
    if not *consistent* **then**
      A-Propagate($x, p_2, N, M, consistent, -, -$);
  }
  if *consistent* **then** {
      $N \leftarrow N - M$;
      if $N \neq \emptyset$ **then** A((N, D, C));
  } **else** report no solution for (N, D, C)
}

Figure 3: Algorithm for Two-fan Constraints

Theorem 3 *Given a two-fan CSP, there exists an algorithm such that strong n-consistency can be enforced in time complexity $\mathcal{O}(en)$.*

Proof. The A-algorithm is correct according to proposition 1 and 2. The complexity of A-Propagate is at most e and it is called at most n times. The A-algorithm finds one solution to the CSP. To achieve the strong n-consistency and minimality, it can be slightly modified using property 4 to check each variable rather than a set of variables in the main loop. The time complexity is still $\mathcal{O}(en)$,the same as in the A-algorithm. □

5. The OA algorithm

Now we are in a position to deal with a CSP with 0/1/All constraints. First we simplify the CSP by removing those values not allowed by any complete or two-fan constraint. Secondly, we remove those complete constraints and trivial two-fan functions. The above procedure will take no more than $\mathcal{O}(ed)$ time. Now, the new CSP, called a mixed 1/All CSP, contains only

```
Procedure A-Propagate(in x, i, N, out M, con, p1, p2) {
  L ← ∅;
  for each j ∈ N    x_j ← null;
  con ← true;
  for each j such that c_ij ∈ C {
    p_j^ij.coordinate = p_i^ij;
    L ← L ∪ {(p_j^ij, j)};
  }
  repeat {
    Delete first element (y, j) from L
    for each c_jk
      if there is only one z such that (y, z) ∈ c_jk then
        if  x_k = null then {
          L ← L ∪ {(z, k)};
          x_k ← z;
          z.coordinate ← y.coordinate;
        } else if x_k ≠ z then {
            con ← false;
            p1 = y.coordinate;
            p2 = z.coordinate;
            M ← all the other uninstantiated variables
        }
  } until (L = ∅) or (not con);
}
```

Figure 4: A-Propagate for two-fan CSP

functional and two-fan constraints. While it may be possible to directly use O- and A-algorithms to design an algorithm for the mixed system, we however use an integrated approach using follwing theorem.

Theorem 4 *Given a CSP, two variables i and j with c_{ij} being functional, we can eliminate one of variable i or j, to give a new CSP which leaves the solution of the original CSP unchanged.*

The motivation of this theorem comes from the O-algorithm. If there is a functional constraint between i and j, we can eliminate variable j and redirect all constraints involving j to i. We remark that other elimination methods in symbolic computation, eg. Gaussian elimination, can also be thought of as a specialization of the this elimination idea.

The Eliminate algorithm given in figure 5 uses the result of theorem 4 but specialized to the context of 0/1/All constraints. In figure 5, $FC = \{c_{ij} \mid c_{ij} \in C \text{ is functional}\}$ and $FV = \{i, j \mid \exists c_{ij} \in FC\}$. The algorithm for solving the mixed CSP is shown in figure 6.

Theorem 5 *Given a mixed CSP, the OA-algorithm has a time complexity of $\mathcal{O}(ed + en)$ and enforces the CSP to be minimal and strongly n-consistent.*

Proof. Without loss of generality, in this proof we assume all the functional constraints are connected. After the elimination procedure, there are only two-fan functions. So, the algorithm is correct and the resulted CSP

```
Procedure Eliminate (inout (N, D, C), out consistent) {
  consistent ← true;
  Let i ∈ FV, L ← {j | ∃c_ij ∈ FC}
  repeat {
    Select and delete j ∈ L
    for each c_jk ∈ C {
      c'_ik ← c_ij ∘ c_jk;
      if ∃c_ik ∈ C c'_ik then ← c'_ik ∩ c_ik;
      switch (c'_ik) {
        case ∅ : consistency ← false;
            return;
        case functional: L ← L ∪ {k};
            c_ik ← c'_ik;
        case fan-out: if the pivot appears in D_i, remove
                the other values, and vice versa;
            C ← C − {c_ik};
        case two-fan: c_ik ← c'_ik;
      }
  } until L = ∅;
}
```

Figure 5: Elimination algorithm for functional Constraints

```
Algorithm OA-algorithm {
  Eliminate ((N, D, C), consistent);
  if consistent then A((N, D, C))
  else report inconsistency
}
```

Figure 6: Algorithm for mixed CSP

is minimal and strongly n-consistent. For the elimination procedure, each **for** loop takes at most $d * d_i$ times where d_i is the degree of node i because all operations on constraint manipulation can be done in d time. Altogether, we have at most n nodes to deal with and thus the complexity is $\mathcal{O}(ed)$. □

6. Related Work and Discussion

The directly related works on 0/1/All constraints are (Cooper et al. 1994) and (Kirousis 1993), both of which give a sequential algorithm with time complexity of $\mathcal{O}(ed(n + d))$ to find one solution. Note that the n-ary 0/1/All constraints system defined in (Kirousis 1993) is actually a binary constraint system.

In this paper, we obtain beter results with a time complexity of $\mathcal{O}(en)$ for a CSP with only "All" constraints and $\mathcal{O}(e(d + n))$ for CSPs with mixed 0/1/All constraints. In both cases, this time complexity obtains a solution to the CSP as well as enforcing strong n-consistency and minimality. Thus, a higher degree of consistency is obtained compared to (Cooper et al. 1994; Kirousis 1993) with more efficient algorithms.

The other related works (Van Hentenryck et al. 1992; Liu 1995; Affane and Bennaceur 1996; Zhang 1998) are done mainly in the context of arc consistency. Those works consider only functional constraints. Here we give simpler algorithms and an explicit analysis which fully reflects the global property of functional constraints. More specifically, (Van Hentenryck et al. 1992) do not consider finding the global solution, (Liu 1995) only deals with giving a more efficient algorithm for dealing with increasing functional constraints. (Affane and Bennaceur 1996) introduces a new kind of consistency, label-arc consistency, and show that the pure functional constraints with limited extensions to other constraints can be solved, but no detailed analysis of their algorithms is given. (Zhang 1998) embeds the techniques dealing with functional constraint in arc-consistency algorithms in a similar way to (Liu 1995) and proposes the problem of *conflict of orienting* from which all the above mentioned algorithms (except (Van Hentenryck et al. 1992)) suffer.

One result, which we obtain for the class of functional or "1" constraints, is a new algorithm for functional constraints with time complexity $\mathcal{O}(ed)$. In terms of time complexity, it is the same as existing results. However, an advantage of the algorithm here is its conceptual simplicity and ability to achieve minimality. The development of the O-algorithm clarifies how functional elimination can be applied in general, resulting in its use in the OA-algorithm and the special form of propagation in the A-algorithm. In addition, both the O and OA algorithms avoid the problem of conflict of orienting for CSPs which are known in advance.

An important consequence of the techniques developed here for functional elimination is that functional elimination is of broad applicability in the more general context of arbitrary CSP problems. It is possible to show that the elimination method for functional constraints here, for general CSP problems, will be at most $\mathcal{O}(ed^2)$ time. This means that, it can be incorporated into general arc-consistency algorithms without any increase in time complexity, while at the same time obtaining more consistency. Other results with elimination methods on linear equations, (Harvey and Stuckey 1998; Zhang 1998), also suggest the efficacy of such elimination approaches.

7. Conclusion

The 0/1/All constraints play an important role both theoretically (Cooper et al. 1994) and in practice (Van Hentenryk 1989; Kirousis 1993). This paper gives fast algorithms and analyses for functional, two-fan, and mixed CSPs. The elimination method used in solving the mixed CSPs is of broader applicability in a more general CSP setting. In addition, its incremental nature makes it suitable in a solver engine of a constraint logic programming language (Jaffar and Maher 1994).

References

M. S. Affane and H. Bennaceur, A Labelling Arc Consistency Method for Functional Constraints, *Proceedings of CP96*, Cambridge, MA, USA, 1996

P. van Beek and R. Dechter, Minimality and Global Consistency of Constraint Networks, *Journal of ACM*, Vol 42(3), 543-561, 1995

M. C. Cooper, D. A. Cohen and P. G. Jeavons, Characterizing Tractable Constraints, *Artificial Intelligence* 65, 347-361, 1994

R. Dechter, From Local to Global Consistency, *Artificial Intelligence* 34, 1-38, 1992

E. C. Freuder, Synthesizing Constraint Expressions, *Communications of the ACM*, Vol 21(11), 958-966,1982

E. C. Freuder, A sufficient condition for backtrack-free search , *Journal of ACM*, Vol 29(1),24-32,1982

M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to NP-Completeness* Freeman, San Francisco, CA, 1979

W. Harvey and P.J. Stuckey, Constraint Representation for Propagation, *Proceedings CP98*, Pisa,Italy, 1998

J. Jaffar and M. J. Maher, Constraint Logic Programming, *Journal of Logic Programming* 19/20, 503-581,1994

L. M. Kirousis, Fast Parallel Constraint Satisfaction, *Artificial Intelligence* 64, 147-160, 1993

B. Liu, Increasing Functional Constraints Need to be checked only once, *International Joint Conference on Artificial Intelligence 95*, 1995

A. K. Mackworth, Consistency in Networks of Relations,, *Artificial Intelligence* 8(1),118-126,1977

R. Mohr and T. C. Henderson, Arc and Path Consistency Revisited, *Artificial Intelligence* 28, 225-233, 1986

U. Montanari, Networks of Constraints: Fundamental Properties and Applications, *Information Science* 7(2), 95-132, 1974

P. van Hentenryk, *Constraint Satisfaction and Logic Programming*, MIT Press, 1989

P. van Hentenryck, Y. Deville, and C. M. Teng, A Generic Arc-Consistency Algorithm and its Specializations, *Artif. Int.* 58, 291-321, 1992

Y. Zhang, Consistency Techniques for Linear Arithmetic and functional Constraints, *Master's Thesis*, National University of Singapore, 1998

Y. Zhang and H. Wu, Bound Consistency on Linear Constraints in Finite Domain Constraint Programming, Proceedings of ECAI98, Brighton, UK, 1998