A Graph Based Synthesis Algorithm for Solving CSPs

Wanlin Pang *
Institute for Information Technology
National Research Council of Canada
Ottawa, Ontario, Canada K1A 0R6
Email: wpang@email.arc.nasa.gov

Scott D. Goodwin
School of Computer Science
University of Windsor
Windsor, Ontario, Canada N9B 3P4
Email: sgoodwin@uwindsor.ca

Abstract

Many AI tasks can be formalized as constraint satisfaction problems (CSPs), which involve finding values for variables subject to a set of constraints. While solving a CSP is an NP-complete task in general, it is believed that efficiency can be significantly improved by exploiting the characteristics of the problem. In this paper, we present a solution synthesis algorithm called ω-CDGT which is an existing algorithm named CDGT augmented with a constraint representative graph called ω-graph. We show that the worst-case complexity of the ω-CDGT algorithm is better than other related algorithms.

Introduction

Constraint satisfaction problems (CSPs) involve finding values for variables subject to constraints which permit or exclude certain combinations of values. Since many problems in AI and other areas of computer science can be formulated as CSPs, it has been a research subject for a long time and researchers have approached the subject in different directions: searching for the CSP's solutions from the possible solution space ((Haralick & Elliott 1980; Freuder 1988; Dechter & Pearl 1988)), reducing a CSP to a simpler and equivalent CSP ((Mackworth 1977; McGregor 1979; Mohr & Henderson 1986; Han & Lee 1988; Chen 1991)), and synthesizing solutions from partial solutions ((Freuder 1978; Seidel 1981; Tsang 1993; Pang & Goodwin 1996)).

Solution synthesis finds all solutions for a given CSP. It can also be used to find all partial solutions with respect to a particular subset of variables. This is useful for solving structured CSPs, where enforcing local consistency is interleaved with a search method. In addition, solution synthesis has inherent parallelism so it is suitable for parallel implementation. The basic idea of solution synthesis is to perform *join and test* continuously on the selected constraints until an n-ary constraint for a problem with n variables is constructed which contains all solutions. The selected constraints mentioned above are chosen according to a given constraint selection scheme. This scheme determines the subset of constraints selected for joining and the order in which they are to be joined. The remaining constraints are used for testing. It is well known that the constraint selection scheme has significant effects on overall efficiency. Since the total number of solutions to a problem may be exponential in the problem size, in general, the space required and the time needed to find all these solutions are also expected to be exponential. The ultimate goal of solution synthesis is perhaps to find the optimal constraint selection strategy with which the time needed and the space required for solving a problem are optimal; that is, the number of total synthesized tuples is minimal and the number of tuples in the largest synthesized constraint is minimal. It is likely that the task of finding an optimal constraint selection is NP-complete. As an alternative, we argue that the ω-graph, a constraint representative graph introduced in (Pang 1998), provides a natural constraint selection strategy for solution synthesis: select a subset of constraints according to the node set of the ω-graph and select them in an order complying with the structure of the ω-graph. In this paper, we propose an ω-graph based synthesis algorithm called *ω-CDGT* and show that combining ω-graph with a synthesis algorithm can lead to a significant improvement in efficiency.

The paper is organized as follows. We first give definitions of constraint satisfaction problems and constraint graphs. We then present the ω-CDGT algorithm and discuss its complexity. Lastly we briefly compare ω-CDGT with other related synthesis algorithms.

Preliminaries

Constraint Satisfaction Problems

A *constraint satisfaction problem (CSP)* is a structure (X, D, V, S). Here, $X = \{X_1, X_2, \ldots, X_n\}$ is a set of variables that may take on values from a set of domains $D = \{D_1, D_2, \ldots, D_n\}$, and $V = \{V_1, V_2, \ldots, V_m\}$ is a family of ordered subsets of X called *constraint schemes*. Each $V_i = \{X_{i_1}, X_{i_2}, \ldots, X_{i_{ri}}\}$ is associated with a set of tuples $S_i \subseteq D_{i_1} \times D_{i_2} \times \ldots \times D_{i_{ri}}$ called *constraint instance*, and $S = \{S_1, S_2, \ldots, S_m\}$ is a family of such constraint instances. Together, an ordered pair (V_i, S_i) is a *constraint* or *relation* which permits the variables in V_i to take only value combinations in S_i.

Let (X, D, V, S) be a CSP, $V_K = \{X_{k_1}, X_{k_2}, \ldots, X_{k_l}\}$ a subset of X. A tuple $(x_{k_1}, x_{k_2}, \ldots, x_{k_l})$ in $D_{k_1} \times D_{k_2} \times$

*Current Address: QSS Group Inc., NASA Ames Research Center, Moffett Field, CA 94035

$\ldots \times D_{k_l}$ is called an *instantiation* of variables in V_K. An instantiation is said to be *consistent* if it satisfies all constraints restricted in V_K. A consistent instantiation of all variables in X is a *solution to the CSP* (X, D, V, S). The task of solving a CSP is to find one or all solutions. The set of all solution is denoted by $\rho(X)$.

A constraint (V_h, S_h) in a CSP (X, D, V, S) is *minimal* if every tuple in S_h can be extended to a solution. A CSP (X, D, V, S) is *minimal* if every constraint is minimal.

A *binary CSP* is a CSP with unary and binary constraints only, that is, every constraint scheme contains at most two variables. A CSP with constraints not limited to unary and binary is referred to as a *general CSP*.

We will also use some relational operators, specifically, *join* and *projection*. Let $C_i = (V_i, S_i)$ and $C_j = (V_j, S_j)$ be two constraints. The *join* of C_i and C_j is a constraint denoted by $C_i \bowtie C_j$. The *projection* of $C_i = (V_i, S_i)$ on $V_h \subseteq V_i$ is a constraint denoted by $\Pi_{V_h}(C_i)$. The *projection* of t_i on V_h, denoted by $t_i[V_h]$, is a tuple consisting of only the components of t_i that correspond to variables in V_h.

Graph Theory Background

In this section, we review some graph theoretic terms we will need later and we define constraint representative graphs, namely, the line graph, the join graph, and the ω-graph.

A *graph* G is a structure (V, E), where X is a set of *nodes* and E is a set of *edges*, with each edge joining one node to another. A *subgraph of* G *induced by* $V' \subset V$ is a graph (V', E') where $E' \subset E$ contains all edges that have both their endpoints in V'. A *partial graph of* G *induced by* $E' \subset E$ is a graph (V, E').

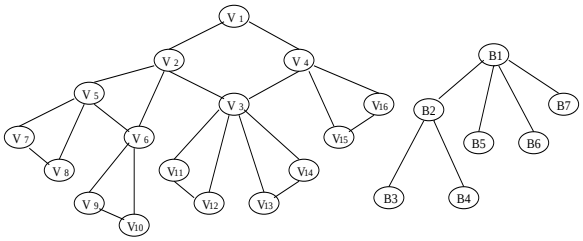
A *path* or a *chain* is a sequence of edges $E_1, E_2, \ldots, E_q$ such that each E_i shares one of its endpoints with E_{i-1} and the other with E_{i+1}. A *cycle* is a chain such that no edge appears twice in the sequence, and the two endpoints of the chain are the same node. A graph is *connected* if it contains a chain for each pair of nodes. A *connected component* of a graph is a connected subgraph. A graph is *acyclic* if it contains no cycle. A connected acyclic graph is a *tree*.

Let $G = (V, E)$ be a connected graph. A node V_i is called a *cut node* if the subgraph induced by $V - \{V_i\}$ is not connected. A *block* (or *nonseparable component*) of a graph is a connected component that contains no cut nodes of its own. An $O(|E|)$ algorithm exits for finding all the blocks and cut nodes (Even 1979).

Let $G = (V, E)$ be a connected graph. The *degree of cyclicity of* G is defined as the number of nodes in its largest block. A graph is k-*cyclic* if its degree of cyclicity is at most k.

A graph $G = (V, E)$ can be decomposed into a tree of blocks $T_B = (V_B, E_B)$. For example, give a graph $G = (V, E)$ as shown in Figure 1 (A), we can have a block tree as in Figure 1 (B), where $B_1 = \{V_1, V_2, V_3, V_4\}$, $B_2 = \{V_2, V_5, V_6\}$, $B_3 = \{V_5, V_7, V_8\}$, $B_4 = \{V_6, V_9, V_{10}\}$, $B_5 = \{V_3, V_{11}, V_{12}\}$, $B_6 = \{V_3, V_{13}, V_{14}\}$, $B_7 = \{V_4, V_{15}, V_{16}\}$. The cut nodes in this graph are V_2, V_3, V_4, V_5, and V_6

A block tree determines an order on the block set. For example, block set $B = \{B_1, B_2, B_3, B_4, B_5, B_6, B_7\}$ is in the depth-first order. For each block B_k $(2 \leq k)$ there is a cut node V_{a_k} of the graph that separates this block from its parent block, and there is a node V_{a_1} in B_1 which is not in any other blocks. These nodes are called *separating nodes*. For example, the separating nodes of the graph in Figure 1 (A) are V_1, V_2, V_3, V_4, V_5, and V_6



(A) A connected graph (B) A block tree

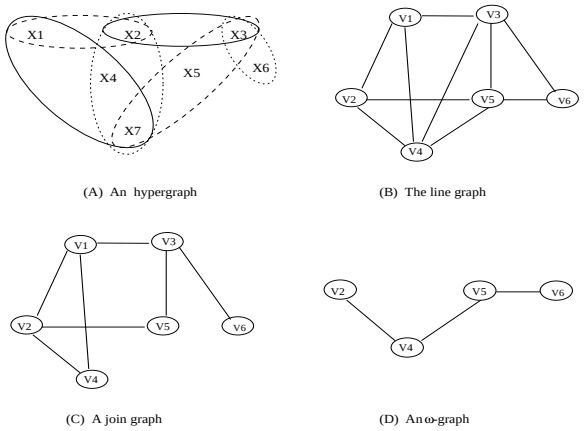
Figure 1: A graph and its block tree

A binary CSP is associated with a simple constraint graph, which has been well studied and widely used for analyzing and solving binary CSPs (Freuder 1988; Dechter & Pearl 1989).A general CSP is associated with a constraint hypergraph, but the topological properties of the hypergraph have not been well studied in the area of constraint satisfaction problems. Instead, constraint representative graphs such as the *line graph*, the *join graph*, and the *ω-graph* have been studied and used to analyzing and solving general CSPs (Jegou 1993; Gyssens, Jeavons, & Cohen 1994; Pang & Goodwin 1998; 2000).

Given a CSP (X, D, V, S) and its hypergraph $H = (X, V)$, the *line-graph* (also called inter graph in (Jegou 1993) and dual-graph in (Dechter & Pearl 1989)) is a simple graph $l(H) = (V, L)$ in which nodes are hyperedges of the hypergraph and with two nodes joined with an edge if these two nodes share common variables. A *join graph* $j(H) = (V, J)$ is a partial linegraph in which some *redundant* edges are removed. An edge in a linegraph is redundant if the variables shared by its two end nodes are also shared by every nodes along an alternative path between the two end nodes. An ω-graph $\omega(H) = (W, F)$ is another constraint representative graph. The node set of an ω-graph is a subset of nodes in the dual graph such that any node in $V - W$ is covered by two nodes in W. There is an edge joining two nodes if either the two nodes share common variables or they cover a node that is not in W.

For example, given a hypergraph $H = (X, V)$ as in Figure 2 (A) with node set $X = \{X_1, X_2, X_3, X_4, X_5, X_6, X_7\}$ and edge set $V = \{V_1, V_2, V_3, V_4, V_5, V_6\}$, where $V_1 = \{X_1, X_2\}$, $V_2 = \{X_1, X_4, X_7\}$, $V_3 = \{V_2, V_3\}$, $V_4 = \{X_2, X_4, X_7\}$, $V_5 = \{X_3, X_5, X_7\}$, $V_6 = \{X_3, X_6\}$. Its line graph $l(H) = (V, L)$ is in Figure 2 (B). There is an edge, for example, between V_1 and V_2 because these two nodes share a common variable X_1. Edge (V_5, V_6) is redundant because the variable X_3 shared by V_5 and V_6 is also shared by every nodes on an alternative path between V_5 and V_6, that is, path (V_5, V_3, V_6). A join graph resulting from re-

moving redundant edges is in Figure 2 (C), and an ω-graph is in (D) in which there is only 4 nodes, since node V_1 is covered by V_2 and V_4, and node V_3 by V_5 and V_6.



(A) An hypergraph (B) The line graph

(C) A join graph (D) An ω-graph

Figure 2: A hypergraph and its representative graphs

Since constraint representative graphs are simple graphs, all of those graph concepts mentioned previously are applicable. For example, an ω-graph (or a join graph) is *k-cyclic* if the number of nodes in its largest block is at most k. An ω-graph can be decomposed into a block tree.

Notice that the line graph or a join graph is also an ω-graph, but in general, an ω-graph is simpler than the line or join graph in terms of the number of nodes, the degree of cyclicity and the width. In particular, (Pang 1998) gives an $O(|V|^3)$ algorithm for constructing an ω-graph for a hypergraph with the following property:

Proposition 1 *Given a hypergraph $H = (X, V)$, there exists an ω-graph whose degree of cyclicity is less than or equal to the degree of cyclicity of any join graph.*

It is shown in (Dechter & Pearl 1988; 1989; Gyssens, Jeavons, & Cohen 1994) that combining the constraint graph with existing CSP algorithms can lead to a significant improvement in efficiency. Based on this proposition, the ω-graph could be the best choice among other graphs. In the next section, we present an ω-graph based synthesis algorithm.

Graph Based Synthesis

A solution synthesis algorithm called *constraint-directed generate and test* (CDGT) is given in (Pang & Goodwin 1996). In CDGT, a constraint on a subset of variables represents a set of partial solutions with respect to that subset of variables. By selecting and joining two lower arity constraints, a possible constraint of higher arity is obtained. This constraint is tightened by testing it against relevant constraints. The tightened constraint represents all the partial solutions with respect to the subset of variables that is the union of the original two variable subsets. This *join and test* process is repeated until a constraint on the whole set of variables is obtained, which contains all solutions. The advantage of this *join and test* procedure over other synthesis procedures, as discussed in (Pang & Goodwin 1996), is that it produces only necessary intermediate constraints, and thus finds all solutions more efficiently than other synthesis algorithms (Freuder 1978; Tsang 1993). However, the question of how to select two lower arity constraints is left for the user, which actually has a significant impact on the performance. In the following, we augment CDGT with a constraint selection strategy based on the ω-graph and present an algorithm called *ω-CDGT*. ω-CDGT performs *joins* on constraints that correspond to the nodes of the ω-graph and tests the joined constraints against the other constraints that are not in the node set. The ordering of joins performed is determined by the depth-first order on the block tree of the ω-graph.

The ω-CDGT Algorithm

Let $I\!P = (X, D, V, S)$ be a CSP and $C = \{C_i = (V_i, S_i) | V_i \in V, S_i \in S\}$ a set of constraints. Let $\omega(H) = (W, F)$ be an ω-graph, $B = \{B_1, B_2, \ldots, B_l\}$ a set of blocks in the depth-first order according to the block tree, and each block $B_k = \{V_{k_1}, V_{k_2}, \ldots, V_{k_{|B_k|}}\}$ a set of nodes in which the first one is the separating node. Let cks denote the set of constraints on $V - W$, that is, $cks = \{C_h = (V_h, S_h) | V_h \in V - W\}$. The *$\omega$-CDGT* algorithm is given as follows.

ω-CDGT$(I\!P, C_V)$

1. **begin**
2. **for** each k from l to 1 **do**
3. $C_{B_k} \leftarrow C_{k_1}$;
4. **for** each j from 2 to $|B_k|$ **do**
5. $C_{B_k} \leftarrow$ *join-test*(C_{B_k}, C_{k_j}, cks);
6. **if** $|S_{B_k}| = 0$ **then return** $C_V \leftarrow (V, \emptyset)$;
7. delete C_{k_j};
8. **end for**
9. $C_{k_1} \leftarrow C_{k_1} \cap \Pi_{V_{k_1}}(C_{B_k})$;
10. **end for**
11. $C_V \leftarrow C_{B_1}$;
12. **for** each k from 2 to l **do** $C_V \leftarrow C_V \bowtie C_{B_k}$;
13. **return** C_V;
14. **end**

Let $r_K = |V_i \cup V_j|$. Function *join-test*(C_i, C_j, cks) generates an r_K-ary constraint C_K on $V_i \cup V_j$ by enumerating those r_K-ary tuples from every tuple in S_i and every tuple in S_j that satisfy all the constraints in cks. In terms of relational algebra, C_K is the relation that results from performing a join operation on relations C_i and C_j with the condition that every constraint $C_h \in cks$ must be satisfied.

join-test(C_i, C_j, cks)

1. **begin**
2. **for** each $tup_i \in S_i, tup_j \in S_j$ **do**
3. **if** $tup_i[V_i \cap V_j] = tup_j[V_i \cap V_j]$ **then**
4. $tup_K \leftarrow tup_i \bowtie tup_j$;
5. **if** $test(tup_K, cks)$ **then** add tup_K to S_K;
6. **return** $C_K = (V_i \cup V_j, S_K)$;
7. **end**

Function $test(tup_K, cks)$ returns $true$ if tuple tup_K satisfies all the constraints in cks, and *false* otherwise.

test$(tup_I, cks))$

1. **begin**
2. **for** each $C_h = (V_h, S_h)$ in cks **do**
3. **if** $tup_I[V_h] \notin S_h$ **then return** *false*;
4. **return** *true*;
5. **end**

Algorithm ω-*CDGT* differs from the original CDGT in that it selects constraints to be synthesized in a predetermined order which is based mainly on the ω-graph. The inner *for* loop (from the 4th to the 8th line) computes a synthesized constraint $C_{B_j} = (V_{B_j}, S_{B_j})$ which contains all consistent instantiations of variables involved in block $B_j = \{V_{j_1}, V_{j_2}, \ldots, V_{j_{|B_j|}}\}$. The operation at the 9th line tries to minimize the constraint corresponding to the cut node that separates this block B_j from its parent B_i and ensures that the to-be-synthesized constraint C_{B_i} will be directionally consistent with constraint C_{B_j}. The outer *for* loop (from the 2nd to the 10th line) computes synthesized constraints for all blocks in a reversed order on B so that this set of newly synthesized constraints is directionally consistent with respect to the ordering defined on B. By joining the synthesized constraints one by one along the ordering, we have all solutions to the given CSP.

Analysis

Given a CSP (X, D, V, S), its ω-graph $\omega(H) = (W, F)$ with a set of blocks $B = \{B_1, B_2, \ldots, B_l\}$. Let n be the number of variables, a the size of the largest domain, r the arity of the CSP, and $|\rho(X)|$ the number of solutions. Suppose that the ω-graph is k-cyclic.

Proposition 2 *The number of tuples in the largest constraint synthesized by the algorithm ω-CDGT is at most* $\max(a^{\min(rk,n)}, |\rho(X)|)$. *The number of total synthesized tuples is in the order of* $O(l(a^{\min(rk,n)} + |\rho(X)|))$.

Since the number of solutions is a fixed parameter of the problem which is independent of any CSP solver, this proposition indicates that the complexity of algorithm ω-CDGT depends on the number of tuples produced within each block. This implies that if this number is bounded, more specifically, if the ω-graph constructed for an r-ary CSP is k-cyclic where k is less than a fixed number, then the CSP can be solved by using ω-CDGT in polynomial time. Otherwise, for any CSP where $rk < n$, the complexity of solving the CPS with ω-CDGT is better than using other related synthesis algorithms, which will be discussed in the next section.

Comparison with Related Synthesis Algorithms

We review some related synthesis algorithms, namely, Freuder's algorithm *FA* (Freuder 1978), Tsang's algorithm *AB* (Tsang 1993), and Pang's CDGT algorithm (Pang & Goodwin 1996). We then briefly compare them with ω-CDGT. The detailed analytical and empirical evaluation can be found in (Pang & Goodwin 1996; Pang 1998).

Freuder's synthesis algorithm *FA* is designed to achieve any level of consistency for a given n-variable CSP. The nth level of consistency ultimately obtained by FA contains all solutions. The basic idea of FA is to incrementally construct k-ary constraints for $1 < k \leq n$, where a k-ary constraint is a join of two or more $k-1$-ary constraints. In general, FA considers all k-ary constraints for all k where $1 \leq k \leq n$; that is, for each k, it produces C_n^k k-ary constraints, and totally it constructs $2^n - 1$ constraints. FA is suitable for achieving k-ary consistency but it produces many unnecessary intermediate constraints when it is used for finding all solutions.

Tsang's algorithm AB is an improvement of FA. F each k where $1 < k \leq n$, instead of constructing all of the C_n^k k-ary constraints as FA does, FA constructs only $n-k+1$ k-ary constraints by joining two adjacent $k-1$-ary constraints (according to a predefined variable partial ordering), and in total it produces $(n(n+1)/2)$ constraints.

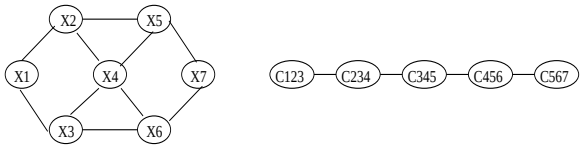
Pang's algorithm CDGT improves FA and AB in three ways: (i) CDGT does not necessarily produces k-ary constraints for every $1 < k$; (ii) if it is necessary to produce k-ary constraints, only one k-ary constraint is synthesized; (iii) each synthesized constraint in CDGT is tighter than that synthesized in FA and AB due to its *join and test* procedure.

CDGT outperforms FA and AB in terms of both time and space cost (see details in (Pang & Goodwin 1996)). If the ω-graph associated with a CSP is not separable (only one block) ω-CDGT degenerates to CDGT, which means that ω-CDGT still outperforms FA and AB. In general, based on proposition 2, the worst-case time and space complexity of ω-CDGT depends on the degree of cyclicity of the ω-graph while the worst-case complexity of other synthesis algorithms depends on the number of variables. Further, according to proposition 1, the worst-case complexity of ω-CDGT is even better than a decomposition algorithm presented in (Gyssens, Jeavons, & Cohen 1994) which is claimed better than other graph based algorithms (Dechter & Pearl 1988).

Another well-known synthesis algorithm is Seidel's *invasion algorithm* (Seidel 1981) which finds all solutions to a given binary CSPs. It is similar to decomposition methods in a way that it decomposes the given CSP into a set of sub-CSPs which are linearly connected (a special tree structure). On the other hand, the invasion algorithm is also similar to the synthesis method in a way that when a sub-CSP is formed, all its solutions are synthesize immediately, and this set of solutions is made consistent with the preceding sub-CSP. Therefore, when the decomposition process is completed, all solutions to each sub-CSP have been obtained, and these sub-CSPs are directionally consistent. By joining them one by one in a reverse order, the set of all solution to the original CSP is obtained.

For example, suppose that the given binary CSP has a constraint graph as shown in Figure 3 (A). The CSP has 7 variables (each corresponds to a node) and 10 constraints (each corresponds to an edge). The invasion algorithm decomposes this CSP into 5 sub-CSPs as in Figure 3 (B).

At the beginning, the algorithm finds all solutions (denoted by C_{123}), with respect to variables 1, 2, 3, by joining constraints C_{12} and C_{23}. Then it joins constraints C_{24} and C_{34} to obtain C_{234}. At the same time, those tuples in C_{234} that are not consistent with C_{123} are deleted. There-



(A) The constraint graph (B) A linear decomposition

Figure 3: An example of invasion algorithm

fore, these two new synthesized constraints are directionally consistent. When C_{567} is obtained, the algorithm joins them from C_{567} to C_{123}. The resulting constraint contains all solutions.

Like any other decomposition method, the efficiency of the invasion algorithm depends on the maximum size of sub-CSPs. For some problems like n-queens problem, the maximum size of sub-CSPs will be n.

The ω-graph based synthesis algorithm can be described in the similar way so that these two algorithms can be compared. Based on the ω-graph, the ω-CDGT algorithm finds all solutions with respect to the variables in each block and enforce directionally consistency at the same time. When every block is processed, ω-CDGT joins these new constraints one by one in the reverse order which produces all solutions to the original problem.

Regardless of the detailed decomposition techniques used in both algorithms, ω-graph based synthesis algorithm has two advantages: first, ω-CDGT can be used to solve general CSPs; second, the tree structure in which the sub-CSPs are connected is more general than the linear structure, which implies that the maximum size of sub-CSPs in ω-CDGT is expected to be less than that in the invasion algorithm.

Conclusion

Since finding solutions to a CSP is an NP-complete task in general, an optimal CSP algorithm is the one that best exploits the characteristics of the problem which we believe are determined by the underlying constraint graph. The ω-graph provides a more precise characterization of the CSP than other graphs such as the line and join graph, and combining the ω-graph with existing algorithm should lead to improvements in efficiency. In this paper, we present a solution synthesis algorithm which is an existing synthesis algorithm CDGT augmented with the ω-graph and we showed that the worst-case complexity of this new algorithm is better than related synthesis and decomposition algorithms.

However, ω-CDGT is not optimal because the number of tuples produced within each block may be affected by the ordering of joins performed within each block and we do not have an ordering of joins within each block that produces an optimal result. If we consider that an ultimate goal of constraint solving is to find an optimal algorithm (not necessarily a polynomial one) for solving CSPs, we may say that ω-graph based algorithm ω-CDGT bring us one step closer towards achieving this goal; that is, we need only to find an optimal algorithm for solving sub-CSPs associated with the blocks in the ω-graph. If the ω-graph is k-cyclic and if $k << |W|$, it may not be that difficult to find an optimal ordering of joins within each block. Needless to say, this is left for future investigation.

References

Chen, Y. 1991. Improving Han and Lee's path consistency algorithm. In *Proceedings of the 3rd IEEE International Conference on Tools for AI*, 346–350.

Dechter, R., and Pearl, J. 1988. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence* 34:1–38.

Dechter, R., and Pearl, J. 1989. Tree clustering for constraint networks. *Artificial Intelligence* 38:353–366.

Even, S. 1979. *Graph Algorithms*. Potomac, Maryland: Computer Science Press.

Freuder, E. 1978. Synthesizing constraint expressions. *Communications of the ACM* 21(11):958–966.

Freuder, E. 1988. Backtrack-free and backtrack-bounded search. In Kanal, L., and Kumar, V., eds., *Search in Artificial Intelligence*. New York: Springer-Verlag. 343–369.

Gyssens, M.; Jeavons, P.; and Cohen, D. 1994. Decomposing constraint satisfaction problems using database techniques. *Artificial Intelligence* 66:57–89.

Han, C., and Lee, C. 1988. Comments on Mohr and Henderson's path consistency algorithm. *Artificial Intelligence* 36:125–130.

Haralick, R., and Elliott, G. 1980. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence* 14:263–313.

Jegou, P. 1993. On some partial line graphs of a hypergraph and the associated matroid. *Discrete Mathematics* 111:333–344.

Mackworth, A. 1977. Consistency in networks of relations. *Artificial Intelligence* 8(1):99–118.

McGregor, U. 1979. Relational consistency algorithms and their application in finding subgraph and graph isomorphisms. *Information Science* 19:229–250.

Mohr, R., and Henderson, T. 1986. Arc and path consistency revisited. *Artificial Intelligence* 28:225–233.

Pang, W., and Goodwin, S. D. 1996. A new synthesis algorithm for solving CSPs. In *Proceedings of the 2nd International Workshop on Constraint-Based Reasoning*, 1–10.

Pang, W., and Goodwin, S. D. 1998. Characterizing tractable CSPs. In *The 12th Canadian Conference on AI*, 259–272.

Pang, W., and Goodwin, S. D. 2000. Consistency in general CSPs. In *The 6th Pacific Rim International Conference on AI*, 469–479.

Pang, W. 1998. *Constraint Structure in Constraint Satisfaction Problems*. Ph.D. Dissertation, University of Regina, Canada.

Seidel, R. 1981. A new method for solving constraint satisfaction problems. In *Proceedings of IJCAI-81*, 338–342.

Tsang, E. 1993. *Foundations of Constraint Satisfaction*. San Diego, CA: Academic Press.