Computational Complexity of Multi-way, Dataflow Constraint Problems

Gilles Trombettoni and **Bertrand Neveu**
Projet Contraintes, CERMICS/INRIA,
2004 route des lucioles, 06902 Sophia-Antipolis Cedex, B.P. 93, France
Gilles.Trombettoni, Bertrand.Neveu@sophia.inria.fr

Abstract

Although it is acknowledged that multi-way dataflow constraints are useful in interactive applications, concerns about their tractability have hindered their acceptance. Certain local propagation algorithms that solve these constraints are polynomial, others (such as SkyBlue) are exponential. Every system handles a specific problem and the influence of any particular restriction on the computational complexity is not yet precisely determined. In this paper, we present three theoretical results that allow us to classify existing multi-way constraint problems. Especially, we prove that the problem handled by SkyBlue is NP-hard.

1 Introduction

Dataflow constraints are rapidly gaining popularity in interactive applications because they simplify the programming task. They are conceptually simple and easy to understand, and are capable of expressing relationships over multiple data types, including numbers, strings, booleans, bitmaps, fonts, and colors [Vander Zanden, 1996]. Dataflow constraint solvers are used in numerous interactive systems, such as graphical user interfaces, spreadsheets, graphical layout systems and animation.

Dataflow constraints are divided into two main categories. A *one way* dataflow constraint has *one* associated function for recovering its consistency. This function calculates output variables using the current value of input variables. The *spreadsheet model*, more formally known as the *dependency graph* model [Hoover, 1987], only takes into account one-way constraints. This model is widely used in interactive systems, mainly because the solving process is based on an efficient *incremental evaluation* phase that topologically sorts the functions to execute.

A *multi-way* dataflow constraint has *several* functions (called *methods*) that may be used to satisfy it. The solving process of problems that contain multi-way constraints needs an additional *planning phase* that assigns a method to each constraint, before the evaluation phase.

Although multi-way constraints are more expressive than one-way constraints, they are less recognized because of concerns about their tractability. Every solving algorithm handles a specific problem and the conditions that allow us to decide whether it is computationally difficult are not clear by now.

This paper aims at giving a classification for the computational complexity of the main existing multi-way constraint problems.

2 Background

A *multi-way dataflow constraint* system can be denoted as (V, C, M). V is a set of variables with a current value. C is a set of dataflow constraints and M is a set of *methods* that can satisfy the constraints.

Definition 1 *A* **multi-way dataflow constraint** *is an equation that has one or more methods associated that may be used to satisfy the equation.*

A **method** *consists of zero or more inputs, one or more outputs, and an arbitrary piece of code that computes the output variables based on the current value of the input variables [Vander Zanden, 1996]. A* **single-output** *method determines only one variable.*

A (dataflow) constraint system is often represented by a *constraint graph* G_c as shown in Figure 1 (a).

Local propagation is the technique used to solve multi-way constraint systems, typically when new constraints are incrementally added. It works in two phases:

- The *planning phase* directs the edges in G_c by assigning one method to each constraint. The result of this phase (*i.e.*, the solution of the corresponding problem) is a *valid* graph G_m called *method graph* (see Figure 1 (b) and (c)).

  Definition 2 *A method graph G_m is* **valid** *if (1) every constraint has one method associated with it in G_m, and (2) G_m has no* **variable conflicts***, that is,* *each variable is the output of, at most, one method (*i.e.*, has at most one incoming edge).*

- When the method graph G_m contains no directed cycles, the *evaluation phase* executes the methods in some topological order. When a method is executed, it sets the output variables to values such that the constraint is satisfied. When G_m is cyclic, strongly connected components are collected and generally passed to external solvers to be satisfied as a whole.

Ignoring the operations involved in method execution and cycle solving, the evaluation problem is in the class P of polynomial problems. Indeed, topological sort is $O(d \times |C|)$ where d is the maximum number of methods associated to one constraint. We concern ourselves with the computational complexity of the problem solved by the planning phase that will be called *constraint planning problem* in the following.

Planning algorithms can be divided into three main categories. (1) DeltaBlue [Freeman-Benson *et al.*, 1990] and SkyBlue [Sannella, 1994] work by propagating the conflicts from the perturbations to the leaves of the constraint graph. (2) The propagation of the degrees of freedom scheme (in short PDOF) selects the methods in the reverse order (*i.e.*, executing first the methods that were chosen last). This algorithm has been used in SketchPad [Sutherland, 1963] and QuickPlan [Vander Zanden, 1996]. (3) A third approach is related to the classical problem of graph matching. It gives the *Maximum-matching* algorithm [Gangnet and Rosenberg, 1992].

3 Different types of constraint planning problems

Existing local propagation algorithms solve different planning problems that imply various tradeoffs between expressiveness and performance:

- *required* constraints only, or both required and *preferential* constraints that are satisfied if possible
- single-output constraints only, or multi-output ones
- acyclic constraint graphs only, or cycles allowed
- *method restriction* imposed or relaxed

Definition 3 *The* **method restriction** *imposes that every constraint method of a given problem must use all of the variables in the constraint either as an input or an output variable [Vander Zanden, 1996].*

Moreover, some systems allow directed cycles in the method graph, whereas others do not, which leads in fact to two different (and incomparable) problems. Indeed, a general computational result states that a restriction

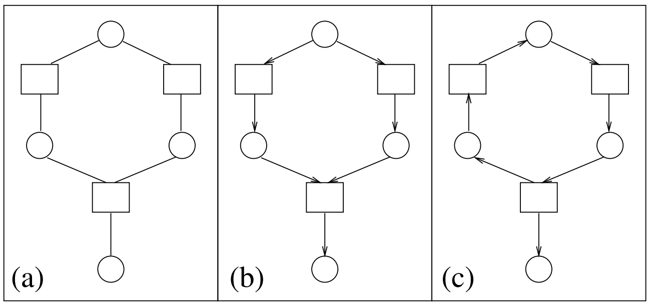

Figure 1: A *constraint graph* is a bipartite graph whose nodes are constraints and variables, respectively represented by rectangles and circles. Each constraint is connected to its variables. This is illustrated in Figure (a). Figures (b) and (c) show two possible *method graphs*. The method selected for each constraint is symbolized by directed edges from the constraint to the output variables, and from the input variables to the constraint. The method graph in (b) contains no directed cycles, as opposed to the method graph in (c).

imposed on a solution does not necessarily make the corresponding problem easier [Papadimitriou, 1994].

Table 1 shows the computational complexity of selected existing constraint planning problems.

4 An NP-complete planning problem with method restriction

Disjunctive constraints do not respect the method restriction (*e.g.*, constraint $a \vee b$ has two methods that output to either variable a or b with no input). However, these constraints are not usually needed in interactive systems. Therefore, all of the propagation algorithms impose the method restriction. The problems handled by these algorithms are in P, except cp_2 (see Table 1) which has not been yet analyzed.

The only known NP-completeness results aim at problems acp_3 and acp_4 [Maloney, 1991] which are not very interesting in practice because they relax the method restriction.

The following theorem states an NP-completeness result about the problem cp_2 for which the method restriction holds and which is handled by SkyBlue.

Theorem 1 *Let G be a dataflow constraint system for which the method restriction holds.*

Then proving the existence of a valid method graph (cyclic or not) corresponding to G is NP-complete.

The proof and the polynomial reduction involved in it are described in following paragraphs.

4.1 Polynomial reduction

In Section 4.2, we prove that the known NP-complete problem "*Exact Cover by 3-Sets*" [Papadimitriou, 1994]

problem	method restriction	single-output	complexity	proof	algorithms
acp_1	yes	yes	P	[Sutherland, 1963]	PDOF, DeltaBlue
acp_2	yes	no	P	[Vander Zanden, 1996]	QuickPlan
acp_3	no	yes	NPC	[Maloney, 1991] 3	–
acp_4	no	no	NPC	[Maloney, 1991] 1 and 2	–
cp_1	yes	yes	P	[Gangnet & Rosenberg, 1992]	Maximum-matching
$\boldsymbol{cp_2}$	**yes**	**no**	**??**	–	**SkyBlue**
$\boldsymbol{cp_3}$	**no**	**yes**	**??**	–	–
$\boldsymbol{cp_4}$	**no**	**no**	**??**	–	–

Table 1: Computational complexity of constraint planning problems. Cycles in the constraint graph are allowed. Constraints are *required* (not *preferential*). Every problem depends on three characteristics: (1) a problem that only accepts *a*cyclic method graphs is designed by acp_i ($i \in \{1..4\}$), whereas a problem that accepts both acyclic and cyclic solutions is designed by cp_i; (2) the method restriction; (3) the presence of single-output constraints only. The complexity of problems cp_2, cp_3 and cp_4 is not yet known, especially cp_2 which is handled by SkyBlue.

can be reduced to "*constraint-planning*" (*i.e.*, cp_2). This reduction will be called *planning reduction*.

Definition 4 (Exact Cover by 3-Sets) *Let X be a finite set, such that $|X| = 3q$ for some integer q. Let E be a family of sets that contain 3 elements of X each. Every element of X belongs to at least one 3-set of E.*[1]

Does E contain an exact cover for X, that is, a subset S of E such that every element of X belongs to exactly one 3-set of S?

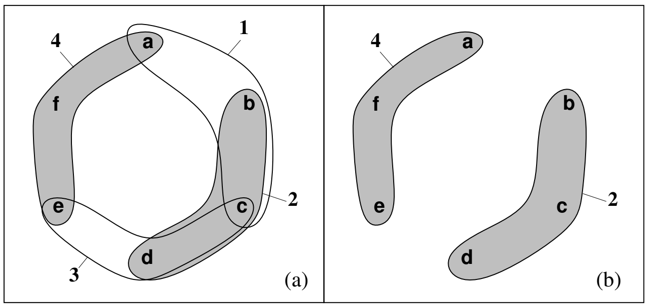

Figure 2: An instance of the "*Exact Cover by 3-Sets*" problem. $X = \{a...f\}$. The 3-sets in $E = \{1...4\}$ are represented by hyper-arcs, as shown in (a). The unique solution $S = \{2, 4\}$ is shown in (b).

Let $G = (X, E)$ be an instance of "*Exact Cover by 3-Sets*". Let $G' = (V, C, M)$ be an instance of "*constraint-planning*" obtained by a planning reduction applied to G as follows:

- Variables in V are divided into two sets *VarX* and *VarE*. Constraints in C are divided into two sets *ConstX* and *ConstE*.
- Each element i of X corresponds to one variable *i-at-most* of *VarX*.
- A 3-set $p = \{a, b, c\}$ in E corresponds to a constraint set_p of *ConstE* connecting six variables. set_p has two triple output methods which indicate whether the 3-set p is either present or absent in the solution. The *present method* outputs to the three variables *a-at-most*, *b-at-most*, *c-at-most* of *VarX* (It ensures that no other 3-set will cover the corresponding elements.) The *absent method* outputs to the three variables *a-p*, *b-p*, and *c-p* of *VarE*.
- When element i of X can be covered by n different 3-sets $\{p_1...p_n\}$ of E, n variables $\{i\text{-}p_1...i\text{-}p_n\}$ of *VarE* are constructed. One constraint *i-at-least* of *ConstX* is also built, that connects these variables. *i-at-least* has n single-output methods, one for each variable in the constraint. The method that outputs to variable $i\text{-}p_k$ ensures that element i of X is covered by (at least) the 3-set p_k.

Figure 2 shows an instance of "*Exact Cover by 3-Sets*" that is reduced to the "*constraint-planning*" instance of Figure 3.

The planning reduction is based on the following intuition. One element i of X appears in exactly one 3-set of solution S. Thus, i appears *at most once* and *at least once* in a 3-set of S. This is translated into the planning problem as follows:

- (at most once) If the 3-set p belongs to solution S, then the *present* method of set_p is selected in the corresponding constraint planning problem. Thus, the variables determined by set_p ensure that no other 3-set than set_p covers them, as this would lead to variable conflicts.
- (at least once) As said above, a constraint *i-at-least* directed onto variable *i-p* ensures that element i of X is covered by the 3-set p in the solution. In fact, no method can be selected for constraint *i-at-least* (thus involving no solution) if *every* connected variable is the output of an *absent* method.

[1]This additional hypothesis discards trivial instances while keeping the problem NP-complete.

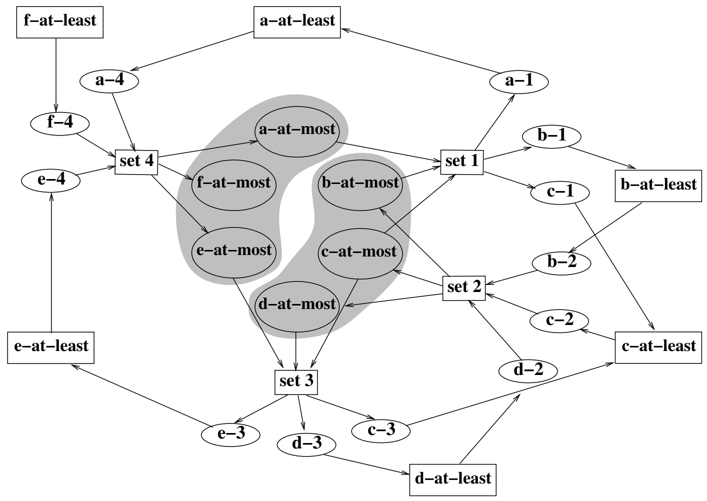

Figure 3: Valid method graph after transforming the instance of "*Exact Cover by 3-Sets*" (Figure 2).

4.2 Proof of Theorem 1

First, the planning reduction is $O(|X|+|E|)$. Indeed, one 3-set of E corresponds to one constraint, five methods, nine edges, and three variables in the planning problem. One element of X corresponds to one constraint and one variable.

Second, "*constraint-planning*" is in *NP* since verifying that a method graph issued from a planning reduction is valid is $O(|V|+|C|)$. Indeed, it just has to be verified that every variable is determined by at most one constraint, and that every constraint has one method selected for it.

Finally, the two following paragraphs prove the equivalence between (i) a solution S for any instance G of "*Exact Cover by 3-Sets*" and (ii) a solution S' for the instance G' of "*constraint-planning*" obtained by applying to G the planning reduction.

$(i) \rightarrow (ii)$ Based on S, S' is obtained by selecting present methods for the constraints set_p in *ConstE* when the 3-set p is in S. The absent method is selected for the 3-sets that are not in S. Every constraint *i-at-least* in *ConstX* is directed onto variable *i-p* of *VarE* when the 3-set p is in S. Variable *i-p* is an input of the present method selected for the constraint set_p in *ConstE*. By construction, every constraint has one method selected for it.

By hypothesis, every element of X belongs to exactly one 3-set of S. Since there is no intersection between any two 3-sets in S, this construction does not generate conflicts on variables *i-at-most* of *VarX*.

Every element i of X is covered by (at least) one 3-set p of S. By construction, set_p is activated with the present method that outputs to *i-at-most*. By construction, variable *i-p* is determined by constraint *i-at-least*, thus there is no variable conflict generated on *i-p*. The other variables of constraint *i-at-least* do not provide conflicts because they are linked only to two constraints and are not determined by *i-at-least*. Thus, for every element i of X, no corresponding variable in the constraint planning problem can cause a variable conflict.

$(ii) \rightarrow (i)$ Based on S', S is built by collecting a 3-set $\{a, b, c\}$ when the present method that outputs to variables *a-at-most*, *b-at-most*, and *c-at-most* is selected. Since the method graph is valid, the intersection of any two 3-sets in S is empty.

Let us consider every constraint *i-at-least* of *ConstX* in S'. Let *i-p* be the variable determined by *i-at-least*. *i-p* is necessarily an input variable of constraint set_p that determines variable *i-at-most*, otherwise a variable conflict would occur on *i-p*. By construction, i necessarily belongs to a 3-set in S. □

4.3 Complexity of 2-output constraint planning problems

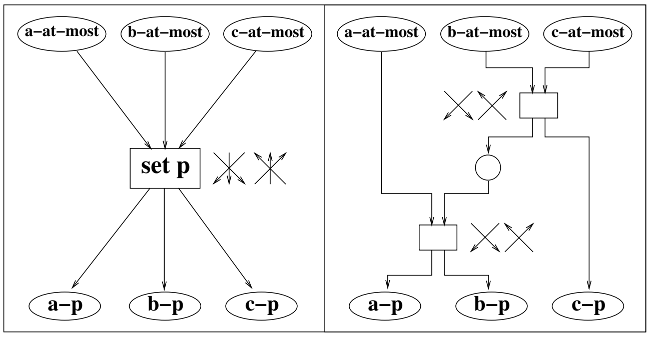

Figure 4: A 3-output constraint set_p transformed into two 2-output constraints. The motifs next to a constraint indicate the possible methods.

We know that when "*constraint-planning*" is restricted to single-output constraints, the problem complexity comes down to P (cp_1). The planning reduction shows that "*constraint-planning*" is NP-complete with 3-output constraints. A natural question is therefore, whether the 2-output constraint restriction would yield a polynomial problem or not.

Theorem 2 *Let G be a dataflow constraint system for which the method restriction holds. G contains methods that have at most two outputs.*

Then proving the existence of a valid method graph (cyclic or not) corresponding to G is NP-complete.

Proof. Every 3-output constraint set_p can easily be transformed into two 2-output constraints and a

problem	method restriction	single-output	complexity	proof	algorithms
acp_1	yes	yes	P	[Sutherland, 1963]	PDOF, DeltaBlue
acp_2	yes	no	P	[Vander Zanden, 1996]	QuickPlan
acp_3	no	yes	NPC	[Maloney, 1991] 3	–
acp_4	no	no	NPC	[Maloney, 1991] 1 and 2	–
cp_1	yes	yes	P	[Gangnet and Rosenberg, 1992]	Maximum-matching
$\boldsymbol{cp_2}$	**yes**	**no**	NPC	**Theorems 1 and 2**	**SkyBlue**
$\boldsymbol{cp_3}$	**no**	**yes**	P	**Theorem 3 and $\boldsymbol{cp_1}$**	**Maximum-matching**
$\boldsymbol{cp_4}$	**no**	**no**	NPC	$\boldsymbol{cp_2}$	**SkyBlue**

Table 2: Computational complexity of constraint planning problems. The contributions of this paper are bold-faced.

"dummy" variable, as shown in Figure 4. The global behavior remains exactly the same[2]. □

5 Influence of the method restriction on problems with cyclic solutions

The following theorem states that the method restriction has no influence on the computational complexity of the constraint planning problem when cyclic solutions are allowed.

Theorem 3 *Let C be a class of dataflow constraint systems and let P_C be the problem of existence of a valid method graph (cyclic or not) for any instance in the class C. Let P'_C be the restriction of P_C to constraint systems that satisfy the method restriction.*

Then P_C and P'_C are polynomially (actually LOG-space) equivalent.

The proof is based on the *method transformation.*

Definition 5 *Let $G_1 = (V_1, C_1, M_1)$ be a constraint system. Based on G_1, the method transformation provides a constraint system $G_2 = (V_2, C_2, M_2)$ such that: (1) $V_1 = V_2$, (2) $C_1 = C_2$, (3) methods in M_1 for which the method restriction holds, occur unchanged in M_2, and (4) every method m_1 in M_1 for which the method restriction does not hold is replaced by a method m_2 in M_2 for which the method restriction holds: m_2 has the same output variables as m_1 and has* all *of the other variables of the associated constraint as input.*

Note that this trivial transformation is LOG-space. Thus, Theorem 3 can be applied to problems that are in P or are NP-complete.

Proof of Theorem 3. First, P'_C can be reduced to P_C since P'_C is a restriction of P_C. Second, P_C can be reduced to P'_C thanks to the method transformation that reduces a constraint system G_1 into a constraint system G_2 for which the method restriction holds. We prove the equivalence between (i) a solution of G_1 and (ii) a solution of G_2.

$(ii) \rightarrow (i)$ A valid method graph of G_2 can be transformed into a valid method graph of G_1 since G_1 is the same as G_2 without certain edges. This does not induce variable conflicts.

$(i) \rightarrow (ii)$ A valid method graph of G_1 can be transformed into a valid method graph of G_2 since every edge added by the method transformation connects a constraint and one of its *input* variable. This does not generate variable conflicts. □

Note that if directed cycles are not allowed in the solution, the last implication is false because adding an input edge could introduce a directed cycle.

6 Synthesis

These three theorems allow us to deduce the three missing computational complexity results, as shown in Table 2. Since cp_2 (which is NP-complete) is a restriction of cp_4, and cp_4 is in NP, cp_4 is also NP-complete[3]. Theorem 3 proves that cp_1 and cp_3 have the same computational complexity.

Since the table is now complete, we can highlight interesting points about the constraint planning problems handled by existing algorithms.

SkyBlue Problem cp_2 assumes that the constraints must be *required*, whereas SkyBlue [Sannella, 1994] can handle one type of *preferential* constraints. Therefore, the SkyBlue problem is NP-hard. The NP-completeness result given by Theorem 1 makes the exponential worst case time complexity of SkyBlue less surprising. However, [Sannella, 1994] has proven that SkyBlue could reach this worst case complexity even on problems acp_2 and cp_1 that are in P.

QuickPlan QuickPlan [Vander Zanden, 1996] cannot be extended without making the corresponding problem NP-complete. Indeed, the gap between acp_2 (in P) and acp_4 (NP-complete) is due to the method restriction, as described in [Vander Zanden, 1996]. Moreover, the gap between acp_2 (in P) and cp_2 (NP-complete) is due also to cyclic solutions being accepted or not.

[2] Figure 3 illustrates that every variable is determined by a constraint. Thus, no solution can be found for 2-output constraint problems when the two 2-output constraints are selected in opposite directions.

[3] Theorem 3 applied to cp_2 and cp_4 also proves this result.

The companion paper [Trombettoni and Neveu, 1997] proves that there exist instances of "*constraint-planning*" issued from the planning reduction which only have cyclic solutions (otherwise P would be equal to NP).

Maximum-matching The Maximum-matching problem is in P [Gangnet and Rosenberg, 1992]. Since the gap between cp_1 and cp_2 lies in the *single-output constraint* restriction, Maximum-matching cannot be extended to multi-output constraints.

Note that Maximum-matching can also solve problem cp_3. Indeed, Theorem 3 can easily be extended to the problem of *finding* a solution, thanks to a reverse transformation. So one needs to (1) transform an instance of cp_3 into one of cp_1 with the *method transformation*, (2) call Maximum-matching on the cp_1 instance and (3) retrieve the solution (if any) with the reverse method transformation.

7 Complexity of problems with acyclic constraint graphs

We know that an acyclic constraint graph cannot yield a method graph with directed cycles. The restriction of the two problems acp_i and cp_i ($i \in \{1..4\}$) to acyclic constraints graphs is then a unique problem p'_i.

Problems p'_1 and p'_2 are in P because they are restrictions of acp_1 and acp_2. In the same way, p'_3 is in P since it is a restriction of cp_3. p'_2 and p'_4 can be seen as problems where cyclic solutions are allowed (in fact, all solutions are acyclic and one does not need to disallow cyclic solutions). They satisfy the conditions of Theorem 3 so that p'_4 is in P. We can then conclude that all of the restrictions to acyclic constraint graphs are in P.

8 Conclusion

We have proven new computational complexity theorems. First, the constraint planning problem handled by SkyBlue is NP-hard. We do not know yet whether it is in NP, when handling *constraint hierarchies* that are a widely used type of preferential constraints. Second, the computational complexity is insensitive to the method restriction when cyclic solutions are allowed. Based on the theoretical results presented in this paper, the following simple rule gives sufficient conditions to determine if a given constraint planning problem is in P.

if the constraint graph contains no cycle **then**
  the problem is in P
else if an acyclic solution is expected **then**
  the problem is in P if the method restriction is imposed
else
  the problem is in P if it only contains single-output constraints

This rule highlights the importance of the "cyclic/-acyclic solution" condition. When directed cycles are not allowed in the solution, the gap between problems in P and NP-complete problems comes from the method restriction, and not from the single-output constraint restriction. The problem complexity has exactly the opposite behavior when cyclic solutions are allowed. Finally, the companion paper shows that the polynomial complexity of problem acp_1, acp_2, cp_1, or cp_3 is not lost when handling constraint hierarchies.

We believe that these results will help designers to conceive multi-way constraint systems that provide a good balance between expressiveness and performance.

Acknowledgements

Special thanks to N. Chleq and T. Schiex whose comments were very helpful. Also thanks to C. Bliek, C. Lottaz, N. Prcovic, and R. Stalker.

References

[Freeman-Benson *et al.*, 1990] Bjorn Freeman-Benson, John Maloney, and Alan Borning. An incremental constraint solver. *Communications of the ACM*, 33(1):54–63, January 1990.

[Gangnet and Rosenberg, 1992] Michel Gangnet and Burton Rosenberg. Constraint programming and graph algorithms. In 2^{nd} *International Symposium on Artificial Intelligence and Mathematics*, January 1992.

[Hoover, 1987] Roger Hoover. *Incremental Graph Evaluation*. PhD thesis, Cornell University, Ithaca, 1987.

[Maloney, 1991] John Maloney. *Using Constraints for User Interface Construction*. PhD thesis, Department of Computer Science and Engineering, University of Washington, Seattle, 1991. Published as Technical Report 91-08-12.

[Papadimitriou, 1994] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[Sannella, 1994] Michael Sannella. *Constraint Satisfaction and Debugging for Interactive User Interfaces*. PhD thesis, Department of Computer Science and Engineering, University of Washington, Seattle, 1994.

[Sutherland, 1963] Ivan Sutherland. *Sketchpad: A Man-Machine Graphical Communication System*. PhD thesis, Department of Electrical Engineering, MIT, 1963.

[Trombettoni and Neveu, 1997] Gilles Trombettoni and Bertrand Neveu. Computational complexity of multi-way, dataflow constraint problems. Technical Report 97–86, CERMICS, January 1997.

[Vander Zanden, 1996] Bradley Vander Zanden. An incremental algorithm for satisfying hierarchies of multi-way, dataflow constraints. *ACM Transactions on Programming Languages and Systems*, 18(1):30–72, January 1996.