A Polynomial Time Local Propagation Algorithm for General Dataflow Constraint Problems

Gilles Trombettoni

Artificial Intelligence Laboratory, E.P.F.L.
CH-1015 Lausanne, Switzerland
trombe@lia.di.epfl.ch

Abstract. The multi-way dataflow constraint model allows a user to describe interactive applications whose consistency is maintained by a local propagation algorithm. Local propagation applies a sequence of *methods* that solve the constraints individually. The local aspect of this solving process makes this model sensitive to cycles in the constraint graph. We use a formalism which overcomes this major limitation by allowing the definition of *general methods* that can solve several constraints simultaneously. This paper presents an algorithm called General-PDOF to deal with these methods which has a polynomial worst case time complexity. This algorithm therefore has the potential to tackle numerous real-life applications where cycles make local propagation unfeasible. Especially, general methods can implement "ruler and compass" rules to solve geometric constraints.

1 Introduction

Dataflow constraints are used in interactive systems, such as graphical user interfaces, graphical layout systems and animation. They simplify the programming task, are conceptually simple and easy to understand. They are capable of expressing relationships over multiple data types, including numbers, strings, bitmaps, fonts, and colors [Vander Zanden, 1996].
Local propagation was first applied to graphical layout systems [Sutherland, 1963], [Gosling, 1983] but is not used in this field anymore. Indeed, cycles in the constraint graph often make local propagation unfeasible for these applications which involve non-linear equations and dense constraint graphs. Instead, many researchers in the CAD field have designed more powerful algorithms reasoning at the geometric level, that is, considering relations between geometric objects (see [Bouma *et al.*, 1995], [Fudos and Hoffmann, 1997], [Dufourd *et al.*, 1998], [Kramer, 1992], [Hsu and Brüderlin, 1997]). Some of them use a *propagation of degrees of freedom* approach which can be viewed as a propagation mechanism working at the geometric level [Hsu and Brüderlin, 1997], [Kramer, 1992].

The model used in this paper remains at the algebraic level and generalizes the notion of *method* specific to local propagation: in the standard model, a method solves one constraint; here, a *general method* solves *a set* of constraints.

The paper describes General-PDOF, a simple, complete and polynomial time algorithm to planify the (general) methods to apply. The contribution of this generalization is twofold:

- For general-purpose interactive applications, this allows a designer to incrementally define new general methods whose constraints form a cycle when the current definitions led to a failure.
- In the CAD field, executing a general method (at the algebraic level) corresponds to applying a "ruler and compass" rule which is traditionnally a basic operation to solve geometric constraints. Thus, General-PDOF can maintain a system of geometric constraints while working at the algebraic level with a simple and general scheme.

2 Background

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

Definition 1 *A* **constraint** *c in C is a relation between a set of variables V_c in V. It has a predicate $\Re_c$ to check whether a valuation of V_c satisfies it.* (i.e., *$\Re_c(\overline{V_c}) = true$; $\overline{V_c}$ denotes a tuple of values, one for each variable in V_c.)*

Constraint c has a set M_c of methods that must be used to satisfy it[1].

A *method* satisfies a constraint by calculating values for its *output variables* in function of the other variables of the constraint.

Definition 2 *A* **method** *m in M_c is a function that can satisfy a constraint c. V_c can be partitioned into two disjoint subsets: the* **input variables** *$V^{in}_{c,m}$ and the non empty subset of* **output variables** *$V^{out}_{c,m}$. (m* **outputs to** *the variables of $V^{out}_{c,m}$.)*

Method m is defined by $\overline{V^{out}_{c,m}} = m(\overline{V^{in}_{c,m}})$ such that constraint c is satisfied. Method m is **executed** *when m is applied and the variables of $V^{out}_{c,m}$ are bound with the new values. Method m is* **free** *if no variable v in $V^{out}_{c,m}$ is connected to a constraint in C (except c). Thus, the execution of a free method does not violate other constraints.*

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

Definition 3 *A* **constraint graph** *is a bipartite graph where nodes are constraints and variables represented by rectangles and circles respectively. Each constraint is connected to its variables.*

[1] Hence the name *multi-way dataflow* constraint.

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

- The *planning phase* assigns one method to each constraint. The result of this phase is a *valid* directed graph G_m called *method graph* (see Definition 4 and Figure 1).
- When the method graph G_m contains no directed cycles, the *evaluation phase* executes the methods in some topological order. Otherwise, *strongly connected components*, *i.e.*, cycles, are collected and evaluated by an external solver.

Definition 4 *A* **method graph** *is a directed graph where nodes are the methods selected by the planning phase. There is an arc* (m_i, m_j) *iff at least one output variable of* m_i *is linked to a constraint satisfied by* m_j*.*

This definition differs from the usual one [Sannella, 1994]. It is more general [Trombettoni, 1997] and will be useful in the following.

Definition 5 *A method graph* G_m *is* **valid** *iff (1) every constraint has exactly one method associated with it in* G_m*, and (2)* G_m *has no variable conflicts, that is, each variable is an output variable of, at most, one method.*

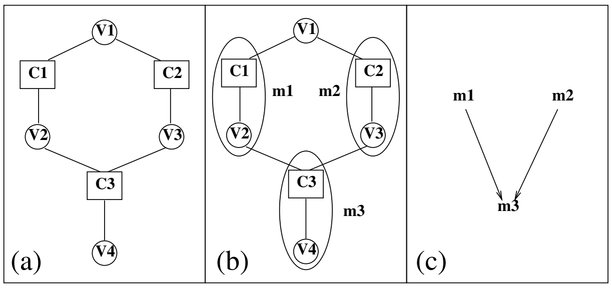

Fig. 1. (a) A (cyclic) constraint graph. (b) Methods selected during the planning phase. A method is represented by an ellipse that encloses both the output variables and the constraint. (c) Valid acyclic method graph formed by these methods. Method m_3, also denoted by m_{c3}^{v4}, is free.

Planning algorithms can be divided into three main categories.

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.

The propagation of degrees of freedom scheme (in short PDOF) selects the methods in reverse order, *i.e.*, first executing the methods that were chosen last. This algorithm has been used in SketchPad [Sutherland, 1963] and QuickPlan [Vander Zanden, 1996].

A third approach is related to the graph problem of bipartite maximum-matching [Gangnet and Rosenberg, 1992], [Serrano, 1987].

3 Definition and Use of General Methods

Most of the systems based on local propagation only allow *single-output* methods that solve *one* constraint by changing the value of *one* of its variables. For example, the constraint $x = y \times z$ has three single-output methods, *e.g.*, $y \leftarrow \frac{x}{z}$. SkyBlue and QuickPlan accept *multi-output* methods that solve *one* constraint (also called multi-output) by changing the value of *several* variables. [Trombettoni, 1995] introduces *general* methods that solve *one or more* constraints by changing the value of *one or more* output variables. These variables must be connected to at least one of the constraints.

Definition 6 *A* **general** *method m can satisfy a set of constraints C_m. Let V_m be the set of variables linked to a constraint in C_m. V_m is partitioned into two disjoint subsets V_m^{out} and V_m^{in}.*

Method m is defined by $\overline{V_m^{out}} = m(\overline{V_m^{in}})$ such that the constraints in C_m are satisfied[2]*.*

3.1 Example

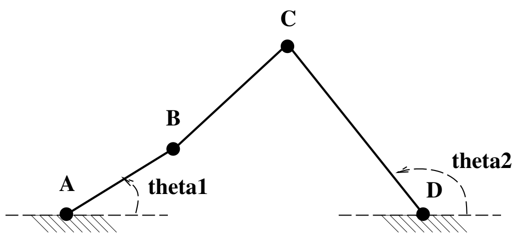

Fig. 2. A mechanism made of three bars AB, BC and CD in 2 D.

The example in Figure 2 is inspired from a linkage described in [Kramer, 1992]. It can be modeled as follows: Points $A\,(x_a, y_a)$ and $D\,(x_d, y_d)$ are fixed in the plane (gray constraints in Figure 3). Bars only impose distance constraints that are quadratic equations. The user can drive the mechanism by moving the bar AB (resp. CD) by an angle Θ_1 (resp. Θ_2).

The corresponding constraint graph is shown in Figure 3 (left), along with the defined methods: Method $m_{c1}^{\Theta_1}$ is a single-output method that can calculate Θ_1 when the location of B is known. Method $m_{c1,c2}^{xb,yb}$ is a general method which gives a new position for the point B. In a sense, this method intersects a line (constraint c_1) and a circle centered in A (constraint c_2) when the location of A and Θ_1 are known. Method $m_{c2,c3}^{xb,yb}$ is a general method giving a new position for the point B which, in a sense, intersects two circles centered in A and C when these points are known. The other methods are symmetric to those above.

[2] The definitions of Section 2 remain unchanged or can be trivially extended.

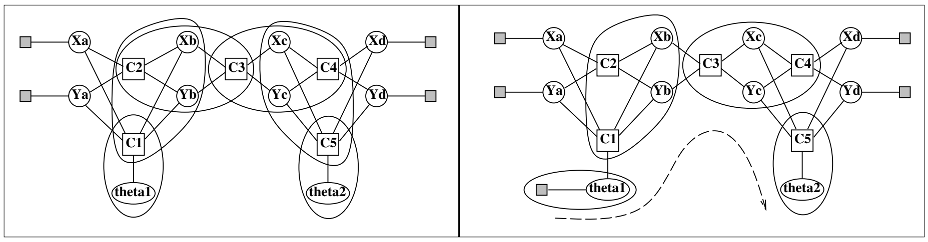

Fig. 3. Left: constraint graph and defined methods of the linkage example. A general method is depicted by an ellipsoid that includes the set of solved constraints and the output variables. Right: the sequence of methods to execute when the user changes the angle Θ_1: to recover the consistency of the linkage, point B is moved, next, point C is moved and finally the angle Θ_2 is modified.

In general, the methods may be either defined by the user or automatically defined from the corresponding constraints. Single-output methods can generally be automatically defined in graphical applications. This is also the case for the general methods of the example that are based on distance and angular constraints.

Remarks

Any numeric method could be used to perform method execution, such as the Newton-Raphson algorithm, or also continuous CSP solvers, such as Numerica [Hentenryck *et al.*, 1997]. It is important to note that a method execution must yield only one (partial) solution, while generally having several choices. In the example, the general methods yield one of the two possible locations for the corresponding point to move. The code of the method should indicate which one to choose. For example, method $m_{c1,c2}^{xb,yb}$ "keeps" only the point with a positive y_b when Θ_1 is comprised in $[0, \Pi]$. This refers to the general and crucial problem of solution predictibility in graphical systems [Kramer, 1992], [Bouma *et al.*, 1995], [Hsu and Brüderlin, 1997]. When one method execution fails, *e.g.,* in case of null intersection between two circles, we consider that the whole propagation process fails.

This problem cannot be modeled with multi-output constraints and shows that the multi-output formalism is a *strict* restriction of the general formalism. See [Trombettoni, 1997] for other examples. To model method $m_{c2,c3}^{xb,yb}$ as a multi-output method, we could consider the conjunction of c_2 and c_3. However, it is then impossible to model method $m_{c3,c4}^{xc,yc}$ in the same way. Indeed, these two methods cannot be defined together as multi-output methods because they both solve constraint c_3, but one solves c_2 while the other solves c_4. Also, methods $m_{c1,c2}^{xb,yb}$ and $m_{c1}^{\Theta_1}$ cannot be defined together in the multi-output method formalism. If only single-output methods are defined in the problem, it can be seen that there is no acyclic method graph able to solve the problem. This means that the PDOF algorithm cannot find a method graph and fails.

4 The General-PDOF Algorithm

We consider the problem of finding a valid acyclic method graph for a dataflow constraint problem with general methods. [Trombettoni, 1997] shows cases where this problem has no solution, whereas solutions exist when considering, not only the defined methods, but also certain implicit *submethods* that are deduced automatically from defined general methods. If these submethods are not known by the propagation algorithm, the solution may be missed. Thus, we would like to design an algorithm that can solve the problem of *finding a valid acyclic method graph by planning with both defined methods and their submethods.* It is important to note that the user does not really care with submethods which will only enable our algorithm to be complete.

Intuitively, the notion of a submethod has the following meaning. When a general method is defined, this may imply that other general methods, called submethods, also exist. A submethod satisfies a subset C' of the constraints of the method and outputs to a subset V' of its output variables. The projection of the result of the method execution on V' satisfies the constraints in C'. For example, consider the general method $m_{c2,c3}^{xb,yb}$ of the previous example. This method induces submethods $m_{c2}^{xb,yb}$ and $m_{c3}^{xb,yb}$. $m_{c2,c3}^{xb,yb}$ calculates the intersection between two circles centered in A and C, and chooses one of the two possible locations (if any). We can also select this point for method $m_{c2}^{xb,yb}$ which could theoretically yield any point on the circle centered in A. Figure 6 will show an example where such submethods are needed to find a valid method graph.

Since submethods are not defined explicitly and may be numerous, it cannot be guessed *a priori* whether a polynomial algorithm can be obtained. This problem is more general than finding a valid acyclic method graph to a system with multi-output constraints. [Vander Zanden, 1996] shows that PDOF remains polynomial when handling multi-output constraints. [Trombettoni, 1995] has presented an exponential algorithm to handle general methods. This section presents the General-PDOF algorithm that solves this problem in polynomial time.

4.1 Submethods

A general method m, given as input of a problem, may induce a method m' that satisfies a subset of the constraints in m and that is not an input of the problem. Figure 4 shows an example.

Definition 7 *Let m be a general method that solves the constraints C_m, with input variables V_m^{in} and output variables V_m^{out}.*

m', a **submethod** *of method m, is defined by m, a subset C'_m of C_m, and a subset $V_m^{out'}$ of V_m^{out}: m' is a function from V_m^{in} to $V_m^{out'}$ such that $m'(\overline{V_m^{in}}) = \Pi_{V_m^{out'}}(m(\overline{V_m^{in}}))$, where Π is the projection on $V_m^{out'}$ of the values returned by m. The submethod m' is* **correct***, i.e., m' is a method, if calling m' yields values for $V_m^{out'}$ that satisfy the constraints in C'_m.*

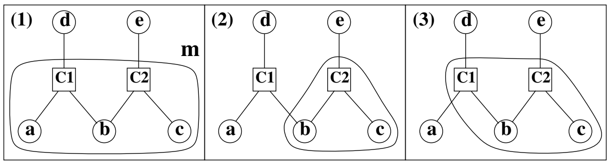

Fig. 4. (1) A general method m that solves constraints c_1 and c_2 and outputs to variables a, b and c. (2) Correct submethod of m. (3) Incorrect submethod of m. Indeed, m changes the value of variable a to satisfy c_1. Thus, there is no guarantee that, for every possible input values for d and e, the projection of the computation of m, only on variable b, satisfies c_1.

This example highlights sufficient syntactical conditions for which a method is correct:

Proposition 1 *Let m be a general method that solves the constraints C_m and outputs to variables V_m. Let m' be a submethod of m that handles constraints $C'_m \subset C_m$ and outputs to variables $V'_m \subset V_m$.*

Submethod m' is correct if no constraint in C'_m is linked to an output variable of V_m which is not in V'_m.

A correct submethod m' means that there exists at least one procedure to perform it, that is, the one given by the defined method m. For instance, the submethod $m_{c2}^{xb,yb}$ previously mentionned in the example can use the result of $m_{c2,c3}^{xb,yb}$ (intersecting the two circles centered in A and C), even if c_2 does not constrain A. However, using method m is not necessary and another procedure can be generated from scratch for m' according to the constraints implied. For instance, another procedure can be used for $m_{c2}^{xb,yb}$ that chooses an arbitrary point on the circle centered in A or that chooses on this circle the closest point from the old position of B.

4.2 Presentation of the Algorithm

Algorithm 1 describes General-PDOF. Its completeness is not trivial and the proof can be found in Section 6. Section 4.3 shows examples. Let us first recall the classical PDOF algorithm.

The key idea behind PDOF is to focus on how to terminate the propagation process, instead of propagating the initial perturbations. It is performed by iteratively selecting a free method, that will be executed after those not yet selected in the remaining constraint graph, and removing its constraints and output variables from the current constraint graph. The process ends when the constraint graph does not contain anymore constraints, and the (directed) method graph formed by the selected methods, *i.e.*, *MG*, is valid and acyclic.

Except that our algorithm can take into account general methods, it is very close to the classical PDOF scheme: only the *Connect* procedure has been added. In the classical PDOF algorithm, the set of defined methods that can currently be selected, including the free ones, evolves as follows when a method is selected

```
algorithm General-PDOF (G: a constraint graph): a valid method graph
    let free be the set of free methods in G
    let MG be an empty acyclic method graph
    while G contains a constraint do
        if free = ∅ then
            exit and return ∅ /* no solution */
        else
            – choose a method m from free and remove it from this set
            – add m to MG (along with the corresponding arcs)
            – remove from G the constraints and the output variables of m
            – Connect (G, m)
            – add to free the free methods of G that previously output to an input
            variable of m
        end
    end
    return MG
end.
procedure Connect (G, m)
    let C_m be the set of constraints solved by m
    let V_m be the set of output variables of m
    for every general method m_i that satisfies a constraint in G connected to an
    input variable of m do
        m'_i ← SubMethod (m_i, C_m, V_m)
        if m'_i ≠ m_i and m'_i has non empty sets of constraints and output variables
        then
            M_i ← Generate-connected-methods (m'_i)
            remove m_i and add the set M_i of connected methods to the set of meth-
            ods (see Definition 8)
        end
    end
end.
```

Algorithm 1: The General-PDOF algorithm.

and its constraint c is removed: the set of methods that satisfy c is not available anymore and the remaining set is unchanged. We also find these two basic cases in General-PDOF, but an additional non trivial case may occur. When a method (general or not) is selected and its constraints C_m and output variables V_m are removed, certain general methods may be neither rejected, nor kept in the current set: those that have a part of their constraints and output variables removed. Lemma 2 will show that their submethods are still correct: *SubMethod* (m_i, C_m, V_m), called by the *Connect* procedure, adds a new (sub)method m'_i to the whole set by removing from m_i the constraints in C_m and the output variables in V_m. Moreover, if m'_i is not *connected*, it is split into the set of its connected subparts by the function *Generate-connected-methods*. Note that since method connectivity is maintained during the planning phase, this requires that defined methods are initially connected.

Definition 8 *Let m be a general method that solves constraints C_m and that outputs to variables V_m. Let g_m be the bipartite graph that represents the dependencies between V_m and C_m in m.*

The method m is **connected** *if g_m is connected, i.e., there exists a path between any two nodes in g_m.*

4.3 Examples

The reader can easily check that General-PDOF can build the method graph of the example in Section 3.1: method $m_{c5}^{\Theta_2}$ is first selected, which frees method $m_{c3,c4}^{xc,yc}$. The selection of that method frees method $m_{c1,c2}^{xb,yb}$. Selecting $m_{c1,c2}^{xb,yb}$ finally frees the input method modifying Θ_1 which is selected at the end.

The example in Figure 5 illustrates the subtleties of the *Connect* procedure and their importance in guaranteeing completeness.

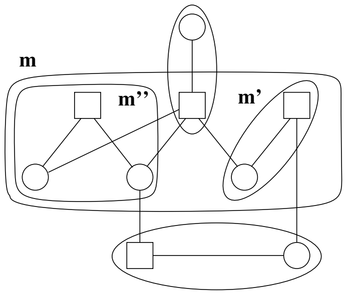

Fig. 5. Role of the *Connect* procedure. At first, the only free method is the single-output method at the top of the figure. When it is selected and removed from the graph, the general method m is replaced by its two connected submethods m' and m''. The free method m' is then selected and removed. This frees the single-output method at the bottom of the figure which is selected next. Finally, the method m'' is selected. If m had not been split into m' and m'', no solution would have been found. Indeed, the "conjunction" of m' and m'' would not have been free (because of m'').

Figure 6 details a geometric example solvable by General-PDOF which highlights the importance of taking into account submethods.

The submethod $m_{c1}^{xb,yb}$ is executed first by placing B in the location calculated by $m_{c1,c2}^{xb,yb}$. This corresponds to one of the two possible locations coming from the intersection between the two circles centered in the new position of A and the old position of C. Using this old position for C, it will be given B a new position close to the old one, in case of a small move of A. Then C will be moved by executing the method $m_{c2,c3}^{xc,yc}$, using the new positions of B and D.

5 Time Complexity

The worst-case time complexity of General-PDOF is $O(n \times dc \times dv \times r \times (g \times dc + g^2))$. n is the number of constraints, r is the maximum number of methods per

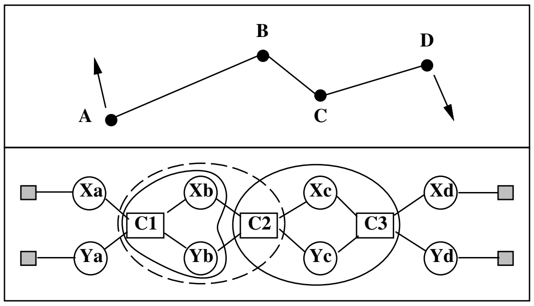

Fig. 6. Another linkage example where the user freely moves A and D simultaneously. All general methods are defined which compute one point given the location of two adjacent ones (intersection of two circles). The methods selected by General-PDOF are shown on the constraint graph. In particular, no solution could be found if the submethod $m_{c1}^{xb,yb}$ of $m_{c1,c2}^{xb,yb}$ was not taken into account.

constraint, dc and dv are the maximum degrees of respectively a constraint and a variable in the constraint graph, and g is the maximum number of constraints and output variables involved in a general method. This complexity is a polynomial function of the input parameters: $dv \leq n$; $g \leq n$; $dc \leq |V|$; $r \leq |M|$.

Details: At each step of General-PDOF, at least one constraint is removed, so that the maximum number of calls to the while loop is n. The other factors correspond to the calls at each iteration, *i.e.,* within the while loop. The complexity of an iteration is dominated by the two last items (see Algorithm 1). In both cases, $dc \times dv \times r$ methods are visited, that is, all of the methods satisfying a constraint at distance 2 of the removed constraints in the bipartite constraint graph[3]. The *Connect* procedure is $O(g \times dc)$ because a method m_i' can be split in at most g connected methods. Checking that a visited method is newly free is $O(g^2)$. These two results are obtained by managing marks for the constraints and output variables of the visited method.

The same analysis gives a more precise time complexity for PDOF which is $O(n \times dv \times dc^2)$ and $O(n \times dv \times dc^2 \times r)$ for multi-output constraints.

Note that this polynomial complexity only concerns the planning phase. The execution phase complexity cannot be evaluated without considering specific types of constraints.

5.1 Discussion

PDOF is $O(n)$ when dc and dv are considered constant, which is the case for the sparse constraint graphs encountered in real user interfaces [Sannella, 1994].

PDOF applied to multi-output methods is $O(n)$ when r is also considered constant. The number r of multi-output methods per constraint could theoretically be very large, but it is a small constant for user interfaces [Sannella *et al.*, 1993].

[3] The data structures follow the existing implementations where entities are implemented as records [Vander Zanden, 1996], [Sannella, 1994].

Also, [Vander Zanden, 1996] shows that PDOF is linear in the number of constraints using benchmarks and existing user interfaces.

General-PDOF is also O(n) when dc, dv, g and r are considered constant. Considering r constant follows the same argument as for multi-output constraints. In CAD applications, the number of ruler and compass rules per constraint should be small. The factor g could theoretically be equal to n, but methods that solve the entire problem should not be defined.

To conclude this discussion, General-PDOF is polynomial. We believe that, in practice, it is linear and very close to the classical PDOF scheme.

6 Completeness

The following completeness property validates General-PDOF.

Proposition 2 *Let $P = (V, C, M)$ be a dataflow constraint problem that contains general methods.*

If there exist valid acyclic method graphs that correspond to P (that may contain submethods of methods in M), then General-PDOF can find one of them.

The proof uses the following two lemmas which are proven in [Trombettoni, 1997].

Lemma 1 *Let P be a dataflow constraint problem that contains general methods and let m be one of the methods of P.*

If m is connected then m has no submethod that is both correct and free.

Lemma 2 *Let us consider any step of General-PDOF applied to a given problem. Let m_1 be a free method that solves the constraints C_1 and outputs to the variables V_1. Let m_2 be another method of the problem.*

After having removed m_1, the submethod m'_2 = SubMethod(m_2, C_1, V_1) is correct.

Proof of Proposition 2 (sketch). The completeness proof of General-PDOF is based on: (1) a confluence property coming from PDOF that remains valid for General-PDOF, and (2) the fact that all of the existing free methods are available at each step of the algorithm.

A confluence property of PDOF states that, considering two free methods m_1 and m_2 at a given step, removing a free method m_1 does not prevent the further selection of m_2. This property remains true for General-PDOF. Indeed, Lemma 2 applied to m_2 proves that m'_2 is correct, and a submethod of a free method is also free. Thus, when method m_1 is selected and removed, m_2 (more precisely m'_2) remains available for a further selection.

The two lemmas are necessary to prove that all of the existing free methods are available at each step of the algorithm. Lemma 1 shows that General-PDOF does not need to consider the submethods of the current connected methods because they cannot be selected. This justifies the connectivity condition maintained during the process. Lemma 2 shows that the submethod m'_i, calculated by the procedure *Connect* of General-PDOF, is correct. This explains why the algorithm adds all of the connected submethods deduced from m'_i since these are all correct and may be free.$\square$

7 Qualitative comparison with Maximum-matching

In the dataflow constraint field, the maximum-matching algorithm (MM) can be used as follows [Gangnet and Rosenberg, 1992]. The planning phase applies MM on the constraint graph. Each pair (c, v) in the matching corresponds to the selection of the single-output method m_c^v. A method graph is thus obtained. The evaluation phase collects the strongly connected components, topologically sorts the *condensed* graph and solves each component in this order: the corresponding method is executed or an external solver is called when a component is made of several methods (a cycle).

MM is close to General-PDOF in that a strongly connected component may correspond to a general method. We highlight here several differences.

7.1 Advantages of MM

First, MM does not require to define general methods, but only single-output methods, which is simpler.

Second, MM finds the same decomposition as General-PDOF when tackling a "structurally well-constrained" problem of equations[4], *i.e.*, for which there exists a perfect matching. Indeed, [König, 1916] has shown that, in this case, the decomposition in strongly connected components is unique. One can check this on the example presented in Section 3.1.

Third, MM always terminates giving a method graph, whereas General-PDOF may fail if there is no acyclic method graph with the defined methods. Indeed, General-PDOF can use only the defined methods and their submethods.

7.2 Advantages of General-PDOF

First, when the problem is under-constrained, such as in most of graphical applications, the result of König does not hold anymore, so that MM may generate (strongly connected) components of arbitrary "semantics" and size. Indeed, MM may generate larger components than General-PDOF would do. *In particular, MM may create a cyclic method graph which needs to be evaluated by an external solver even though an acyclic method graph exists which would lead to a less time-consuming solution.* Moreover, the components created by MM may not correspond to a "real" general method.

For instance, suppose that an additional degree of freedom is allowed in the problem described in Section 3.1: point A can now "roll" horizontally, *i.e.,* x_a is free. In this case, after having changed Θ_1, MM may create the component which corresponds to the general method $m_{c1,c2}^{xa,xb}$. However, this method has no sense geometrically and corresponds to a contradictory system of equations. General-PDOF cannot make this bad choice if $m_{c1,c2}^{xa,xb}$ is not defined.

Second, and even if the problem is well-constrained, MM builds only components which contain as many constraints as variables since there exists a perfect

[4] except if non-square general methods are defined (see below).

matching of the corresponding subgraph. Thus, MM is not able to planify non-square components[5], such as the ones solving inequalities or method $m_{c1}^{xb,yb}$ of the example in Figure 6.

Finally, a component built arbitrarily by MM may yield a solution which is not intuitive for the user, whereas the definition of a general method handled by General-PDOF should precise which solution to choose, according to the semantics of the corresponding ruler and compass rule for example.

7.3 Taking the Best of the Two Algorithms

The confrontation between these two algorithms is not necessary in fact because MM and PDOF can easily be brought together. This can be done as follows:

1. Apply MM on the constraint graph. Let A be the set of arcs in the matching.
2. Apply PDOF on the constraint graph. Two cases may occur at each step:
   (a) There exists a free method m_c^v: m_c^v is selected by PDOF and c and v are removed from the graph. Arcs sharing v or c are removed from A.
   (b) There exists no free method (PDOF is blocked):
      i. Topologically sort the (condensed) directed graph corresponding to A. One obtains a DAG D of components.
      ii. Choose a leaf of D as next free method to select with PDOF.

Thus, this MM-PDOF algorithm can find an acyclic method graph with defined methods if possible (because of PDOF) but never fails and builds new components when no free method is available (because of MM).

The correctness of this algorithm is trivial to check.

We believe that MM-PDOF can be extended to a MM-GPDOF algorithm which makes collaborate MM and General-PDOF. MM-GPDOF will be described in a future work.

8 Conclusion

This paper has described a new local propagation algorithm, called General-PDOF, that can take into account methods solving several constraints simultaneously. This formalism especially allows a system to solve geometric constraints at the algebraic level while keeping a bridge between the two levels.

General-PDOF is simple, complete and has a polynomial time complexity. From a theoretical point of view, it shows that the constraint planning problem with general methods is in P when acyclic solutions are sought. There is no computational jump from simpler problems (with only single-output or multi-output constraints [Trombettoni and Neveu, 1997]). In practice, this complexity should be linear in the number of constraints for user interfaces or CAD applications and General-PDOF should be almost as efficient as PDOF.

[5] These components generally correspond to general methods which choose a solution among an infinite set...

The algorithm given in [Trombettoni, 1995] solves the same problem. However, it is complicated and has an exponential time complexity. The comparison between the two algorithms suggests that the *propagation of conflicts* scheme is not suitable to handle general methods [Trombettoni, 1997].

9 Future Works

Many systems based on local propagation, such as DeltaBlue, SkyBlue and QuickPlan, allow the user to define both *required* constraints that must be satisfied and *preferential* constraints that are satisfied if possible [Borning *et al.*, 1992]. QuickPlan, based on PDOF, removes preferential constraints with a low priority until an acyclic solution is obtained. However, this technique may fail when cycles only contain required constraints. General-PDOF could easily replace the PDOF procedure of QuickPlan in order to take into account both general methods and *constraint hierarchies* [Borning *et al.*, 1992]. A brief analysis shows that the *locally-graph-better* criterion still holds for this hybrid algorithm [Trombettoni, 1997].

We intend to design a MM-GPDOF algorithm which will allow a simple and pertinent collaboration between maximum-matching and General-PDOF: MM-GPDOF would planify defined (general) methods if possible and would build non-defined general methods (corresponding to strongly connected components) only when necessary.

The main contribution of this paper is "theoretical", formally describing a new general-purpose local propagation algorithm and its properties. The potential of General-PDOF to maintain systems of geometric constraints must be validated. We intend to develop a prototype suitable for such applications and especially compare it to the propagation of (geometric) degrees of freedom approach [Hsu and Brüderlin, 1997], [Kramer, 1992]. It will be desirable to allow this tool to automatically define the set of ruler and compass general methods based on the given geometric constraint graph.

In this paper, local propagation has been presented as a technique for incrementally maintaining a set of constraints. However, it could also be used to decompose a set of numeric constraints before satisfaction. [Bliek *et al.*, 1998] presents a distance problem made of tetrahedra and additional bars. In that example, General-PDOF could be used to find the decomposition in small blocks.

Acknowledgments

Special thanks to Christian Bliek and Bertrand Neveu for many useful discussions. Also thanks to Nicolas Chleq and Steven Wilmott for comments on earlier versions of the paper.

References

[Bliek *et al.*, 1998] Christian Bliek, Bertrand Neveu, and Gilles Trombettoni. Using Graph Decomposition for Solving Continuous CSPs. In *these proceedings*, 1998.

[Borning *et al.*, 1992] Alan Borning, Bjorn Freeman-Benson, and Molly Wilson. Constraint hierarchies. *Lisp and Symbolic Computation*, 5(3):223–270, September 1992.

[Bouma *et al.*, 1995] William Bouma, Ioannis Fudos, Christoph Hoffmann, Jiazhen Cai, and Robert Paige. Geometric constraint solver. *Computer Aided Design*, 27(6):487–501, 1995.

[Dufourd *et al.*, 1998] Jean-François Dufourd, Pascal Mathis, and Pascal Schreck. Geometric Construction by Assembling Solved Subfigures. *Artificial Intelligence*, 99(1):73–119, 1998.

[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.

[Fudos and Hoffmann, 1997] Ioannis Fudos and Christoph Hoffmann. A graph-constructive approach to solving systems of geometric constraints. *ACM Transactions on Graphics*, 16(2):179–216, 1997.

[Gangnet and Rosenberg, 1992] Michel Gangnet and Burton Rosenberg. Constraint programming and graph algorithms. In *Second International Symposium on Artificial Intelligence and Mathematics*, January 1992.

[Gosling, 1983] James Gosling. *Algebraic Constraints*. PhD thesis, Carnegie–Mellon University, 1983.

[Hentenryck *et al.*, 1997] Pascal Van Hentenryck, Laurent Michel, and Yves Deville. *Numerica : A Modeling Language for Global Optimization*. MIT Press, 1997.

[Hsu and Brüderlin, 1997] Ching-Yao Hsu and Beat Brüderlin. A degree-of-freedom graph approach. In , editor, , pages 132–155. Springer Verlag, 1997.

[König, 1916] D. König. Über Graphen und ihre Anwendung auf Determinantentheorie und Mengenlehre. In *Math Ann 77*, pages 453–465, 1916.

[Kramer, 1992] Glenn Kramer. *Solving Geometric Constraint Systems*. MIT Press, 1992.

[Sannella *et al.*, 1993] Michael Sannella, John Maloney, Bjorn Freeman-Benson, and Alan Borning. Multi-way versus one-way constraints in user interfaces. *Software – Practice and Experience*, 23(5):529–566, May 1993.

[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. Also available as Technical Report 94-09-10.

[Serrano, 1987] D. Serrano. *Constraint Management in Conceptual Design*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, October 1987.

[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. In *International Joint Conference on Artificial Intelligence, IJCAI'97*, pages 358–363, 1997.

[Trombettoni, 1995] Gilles Trombettoni. Formalizing local propagation in constraint maintenance systems. In 7^{th} *Portuguese Conference on Artificial Intelligence, EPIA'95*, pages 83–94, 1995. Lecture Notes in Artificial Intelligence 990.

[Trombettoni, 1997] Gilles Trombettoni. *Solution Maintenance of Constraint Systems Based on Local Propagation*. PhD thesis, University of Nice-Sophia Antipolis, 1997. In french.

[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.