Geometric Constraint Decomposition

Christoph M. Hoffmann[*]
Department of Computer Science, Purdue University
West Lafayette, Indiana 47907-1398

Andrew Lomonosov Meera Sitharam[†]
Department of Computer Science, Kent State University
Kent, Ohio 44242

Abstract

We present a flow-based method for decomposing the graph of a geometric constraint problem. The method fully generalizes degree-of-freedom calculations, prior approaches based on matching specific subgraph patterns, as well as prior flow-based approaches. Moreover, the method generically iterates to obtain a decomposition of the underlying algebraic system into small subsystems.

Keywords: Extremal subgraph, dense graph, network flow, combinatorial optimization, geometric constraint solving, geometric constraint graph.

1 Introduction

Informally, a geometric constraint problem consists of a (finite) set of geometric elements and a (finite) set of constraints between them. The geometric elements are drawn from a fixed universe such as point, lines, circles and conics in the plane, or points, lines, planes, cylinders and spheres in 3-space. The constraints are logical constraints such as incidence, tangency, perpendicularity, etc., or metric constraints such as distance or angle.

The solution of a geometric constraint problem is an instantiation of the geometric elements such that all constraints are satisfied. Here, it is understood that such a solution is in a particular geometry, for example the Euclidean plane, the sphere, or Euclidean 3-space.

In general, every geometric constraint problem can be translated mechanically into a set of nonlinear equations. The equations are usually algebraic, and nonalgebraic formulations involving trigonometric functions can be avoided in nearly all cases. The equations express the constraints, the variables are the coordinates of the geometric elements, in a suitable coordinate system.

[*]Supported in part by NSF Grants CDA 92-23502 and CCR 95-05745, and by ONR Contract N00014-96-1-0635.

[†]Supported in part by NSF Grant CCR 94-09809.

This foundational perspective does not disclose that computing a solution of the nonlinear system is computationally challenging, and except for simple constraint systems, the problem cannot be solved in practice without further machinery. Direct approaches to algebraically processing the *entire* system include standard methods for ideal membership and locating solutions in algebraically closed fields, for example using Gröbner bases [27] or the Wu-Ritt method [4]; numerous algorithms and implementations for solving over the reals based on the methods of, for example, [3, 25, 5, 20] etc.; and algorithms for decomposing and solving sparse systems of polynomial equations based on [16, 28, 17] etc. They apply to general systems of polynomial equations, hence have at least exponential time complexity, are slow in practice, and do not exploit special properties of geometric constraints. They should be used primarily for preprocessing or solving small, compact subsystems. The problems would be further compounded if we allowed constraints that must be expressed as inequalities, such as "point P is to the left of the oriented line L in the plane." Such additions necessitate using cylindrical algebraic decomposition based techniques [5], such as in [7, 21, 29] which have a theoretical worst-case complexity of $O(2^{n^2})$, where n is the algebraic size of the problem, or nonlinear optimization techniques, all of which are slow enough in practice that they do not represent a viable option for large problem sizes.

1.1 Problem Decomposition

If we wish to solve a geometric constraint problem, without discarding the algebraic perspective outright, we will need an effective way in which to decompose the equation system. Such decomposition could be done purely algebraically, using symbolic computation techniques such as those for sparse systems discussed in the previous section, but doing so is inferior, in many cases, to a generalized degree-of-freedom analysis. The latter approach first translates the constraint problem into a graph in which the graph vertices are the geometric elements and the constraints between them are incident edges.

In the constraint graph, it is possible to abstract all geometric elements by numbers that characterize the number of generalized coordinates that must be determined in order to instantiate the element, i.e, the degrees of freedom of the geometric element. In planar Euclidean geometry, for example, a point and a line each have two degrees of freedom, a circle has three, an ellipse five, and so on. These numbers are affixed to the graph vertices as *weights*. Similarly, the constraints can be abstracted by the number of coordinates that are fixed by the constraint, usually the number of equations that express the constraint algebraically. Again, in the Euclidean plane, an incidence constraint between a point and a line determines one coordinate value, whereas an incidence constraint between two points determines two. Therefore, the respective number is affixed to the graph edge representing the constraint, as *weight* of the edge.

As was shown in [12], a key step in the decomposition of the constraint graph into generically solvable subsystems is to find minimal dense subgraphs. A *weighted undirected graph* is a graph where every vertex and every edge has a nonzero integer weight.

Let G be a weighted undirected graph, $G = (V, E, v_i, e_{ij})$, with n vertices V and

m edges E, where v_i, $1 \leq i \leq n$, are the weights of vertices and e_{ij}, $1 \leq i, j \leq n$, are the weights of edges. We want to find an induced subgraph $A \subseteq G$ such that

$$\sum_{e \in A} e - \sum_{v \in A} v > k \tag{1}$$

Such a subgraph A is called *dense*.

More precisely, we want to find a minimal dense subgraph, that is, a dense subgraph A such that A does not contain a proper dense subgraph B. The related problem of finding the **minimum** dense subgraph A is shown to be NP-hard.

Broadly speaking, a minimal dense subgraph corresponds to a subproblem of the geometric constraint problem that can be solved separately in the generic case, and therefore can be used to decompose the nonlinear system of equations. The constant k depends on the geometry, and, in some cases, on symmetries of the subproblem.

As soon as a subgraph of the appropriate density has been found, the corresponding geometric objects can be placed rigidly with respect to each other (or with respect to a global coordinate system) using only the constraints between them. The solver would then continue by condensing the constraint graph, coalescing the placed elements into a new graph vertex and suitably inducing edges to the other vertices. The description of this process is given in Section 5.

1.2 Prior Work on Constraint Graph Analysis

Prior attempts at a degree-of-freedom analysis for constraint graphs often concentrated on recognizing specific dense subgraphs of known shape, such as the triangles of [2, 22, 23] or the patterns of [2, 13, 14]. This approach has limited scope: certain constraint problems can be decomposed very efficiently, but many well-constrained problems cannot be decomposed and the solvers give up on them. The scope can always be extended by increasing the repertoire of patterns of dense subgraphs. However, doing so results in greater combinatorial complexity and eventually makes an efficient implementation too difficult.

More general attempts reduce the recognition of dense subgraphs in a degree-of-freedom analysis to a maximum weighted matching problem in bipartite graphs using methods from, e.g., [18]. A variation [1] of this approach does not use a degree-of-freedom analysis and directly deals with algebraic constraints. In this case, a maximum cardinality bipartite matching is used, since no weights are required. The approach then relies on decomposing into strongly connected components, and one can attempt to generalize it to a weighted version required for a general degree-of-freedom analysis either by replicating vertices, or by retaining weighted vertices, as in [**?**]. We discuss in Section 2.3 why all of these approaches are incomplete and less efficient than the approach presented here. In particular, having found the required matching, or maximum flow, finding a dense subgraph requires significant additional work; it becomes difficult to isolate minimal dense subgraphs, and the approaches only work for density 0, and do not generalize to arbitrary densities. The general approach of [15] appears to be exponential.

A different approach to constraint graph analysis uses rigidity theorems; e.g., [6, 11]. Corresponding decomposition steps may be nondeterministic or require difficult symbolic computations when computing a solution.

2 Dense Subgraphs

In decomposing a geometric constraint system, one would ideally like to locate the smallest possible subsystems. This corresponds generically to finding a minimum dense subgraph of the weighted constraint graph. A provable bound on the size of the minimum dense subgraph, for example, would permit a feasible search for one. Unfortunately (even with the common assumption that the weights of the vertices and edges are bounded) it is easy to construct graphs that have arbitrarily large minimum dense subgraphs. In fact, the next section renders a feasible search unlikely, by showing that the problem (when the weights of the vertices and edges is unbounded) is NP hard.

2.1 Finding a Minimum Dense Subgraph is NP-hard

For a weighted undirected graph G and a constant p, let *SMALL-DENSE* be the problem of deciding whether there is a dense subgraph (without loss, for $K = 0$) in G with at most p vertices.

For an input undirected graph G, let *CLIQUE* be the problem of deciding whether there is a complete subgraph in G with at least p vertices. Recall that *CLIQUE* is NP-complete. A polynomial time reduction of *CLIQUE* to *SMALL-DENSE* shows that *SMALL-DENSE* is NP-hard:

Proposition

SMALL-DENSE is NP-complete.

Proof

That *SMALL-DENSE* is in NP is obvious. Let $G' = (V, E, w)$ be a weighted undirected graph. Every vertex has the weight $w(v) = p(p-1)/2$, and every edge has the weight $w(e) = p$. Let $G = (V, E)$ be the corresponding undirected graph, and (G, p) an instance of *CLIQUE*. If G has a clique of size p, then the corresponding subgraph of G' is dense. Conversely, let S be a dense subgraph of G' of size s at most p. Because of the weights, $s < p$ is not possible. Therefore, since $s = p$, there must be $p(p-1)/2$ edges in S; that is, there is a clique of size p in G. $\square$

2.2 A Greedy Algorithm

Next we give the straightforward greedy algorithm for finding dense (not necessarily minimum, or even minimal), subgraphs and show that it runs for exponential time in the worst case.

Definition

For $A \subseteq G$ define the *density* function $d(A)$ as

$$d(A) = \sum_{e \in A} w(e) - \sum_{v \in A} w(v)$$

The idea of the greedy algorithm is to start with an empty list of subgraphs L. Subgraphs $A \in L$ are built by adding one vertex at a time and computing the density function $d(A \cup \{v\})$. One could expect that L is simply the list of all possible subgraphs considered so far and that its size would increase exponentially. However, it turns out that it is sometimes possible to ignore $A \cup v$ or A, so that the size of L does not always increase when adding a vertex.

Let B be the set of all vertices that were already considered. If $A \cup \{v\}$ is dense then we are done. If not, then if $d(A \cup \{v\}) \geq d(A)$ we can replace A by $A \cup \{v\}$, thus keeping the size of L unchanged. If

$$d(A \cup \{v\}) < d(A)$$

and

$$d(A \cup \{v\}) \ + \sum_{e=(u,u'), u \in B, u' \notin B} w(e) \ \leq \ d(A)$$

then we will not add $(A \cup \{v\})$ to L and keep A in L, so the size of L is unchanged. Therefore, we only increase the size of L by adding $A \cup \{v\}$ when $d(A \cup \{v\}) < d(A)$ and $d(A \cup \{v\}) + cut(v, G - B) > d(A)$

Pseudocode for the Greedy Algorithm

```
1.  $L = \{\}$
2.  $B = \{\}$
3.  for every $v$ in $G$
4.      $B = B \cup \{v\}$
5.      for $A \in L$
6.          if $d(A \cup \{v\}) \geq 0$ then
7.              return $A \cup \{v\}$
8.          else if $d(A \cup \{v\}) \geq d(A)$ then
9.              replace $A$ by $A \cup \{v\}$
10.         else if $d(A \cup \{v\})+$ cut$(v, G - B) > d(A)$ then
11.             $L = L \cup (A \cup \{v\})$
12.         endif
13.     endfor
14. endfor
```

Performance of the Greedy Algorithm

When G is not dense, the algorithm may require exponential time. For example, if G is the rectangular grid, the weight of all vertices is 2, and the weight of all the edges is 1, then the algorithm creates a list L of exponential length.

2.3 Prior Work Using Flow-Based Approaches

Suppose that we want to find a *most* dense subgraph $A \subseteq G$, i.e, one for which $d(A)$ is maximum. We could maximize, over subgraphs A of G, the expression

$$d(A) + \sum_{v \in G} w(v) = \sum_{e \in A} w(e) + \sum_{v \notin A} w(v) \tag{2}$$

or, equivalently, minimize

$$\min_{A \subseteq G} (\sum_{e \notin A} w(e) + \sum_{v \in A} w(v)) \tag{3}$$

To do this, consider a bipartite graph $\tilde{G} = (M, N, \tilde{E}, w)$ associated with the given graph $G = (V, E, w)$. The vertices in N are the vertices in V and the vertices in M are the edges in E. Moreover, the edges of $\tilde{G}$ are $\tilde{E} = \{(e, u), (e, v) \mid e = (u, v), e \in E\}$. The weights w now appear on the vertices of $\tilde{G}$. Maximizing the expression (2) reduces to finding a maximum weighted independent set in the bipartite graph $\tilde{G}$, or, equivalently, the minimum weight vertex cover.

There are two ways to try to find the minimum weight vertex cover. The minimum *cardinality* vertex cover in a bipartite graph can be identified with a maximum cardinality matching and can be found using network flow in $O(\sqrt{n}m)$ time [8]. To take advantage of this algorithm, however, we need to replicate edges and vertices by the corresponding weights. For example, a vertex of weight 3 is tripled. In this larger graph, we find a minimum cardinality vertex cover, and then we try to locate a corresponding minimum weight vertex cover in $\tilde{G}$ and the corresponding dense subgraph in the graph G. No efficient method is known for the latter part.

The unweighted version of bipartite matching in $\tilde{G}$ can be used naturally when variables are directly represented as vertices in G; the algebraic equations are assumed to kill only one degree of freedom, and are represented as edges in G (instead of analyzing general degrees-of-freedom). This again results in a constant factor increase in the size of the graph. The matching naturally induces a direction on the edges of the original graph G, and the strongly connected components provably yield a decomposition into minimal dense subgraphs in the case when the density is chosen to be exactly 0 ($K = -1$). This approach was used in [1], and in fact, gives a natural way of decomposing the entire graph into minimal dense subgraphs. However, it is not clear how to extend the algorithm for general K, or for general, weighted, degree of freedom analysis, with constraints that kill more than one degree of freedom.

An attempt to directly extend this method to general degree of freedom analysis, i.e, to weighted graphs can be found in [24], although it preceded [1]. The method is also a flow-based method and is superficially similar to ours, however, it differs in crucial aspects. Create a source and sink in $\tilde{G}$ corresponding to the M and the N sets of vertices respectively, assigning capacities to M and N corresponding to the weights of the edges and vertices in G respectively. Now a *maximum* flow is found, which corresponds to a "generalized" matching and induces a natural direction on the edges of the graph G as in [1]. Unlike in the unweighted case, however, the strongly connected components are neither guaranteed to correspond to minimal dense, or even dense subgraphs of G, nor do they provide a natural decomposition method, *even for the zero-density case*, i.e, $K = -1$.

A second method for dealing with weights is to search for a minimum weighted vertex cover in $\tilde{G}$ by solving the maximum (vertex) weighted bipartite matching problem. A maximum (edge) weighted bipartite matching problem can be solved in $O(\sqrt{n}m \log n)$ time for bounded weights, [26]. This trivially gives a solution to the maximum (vertex) weighted bipartite matching problem. The catch is that, unlike in the unweighted case, a minimum weighted vertex cover does *not* correspond directly to a maximum weighted matching. Having found a maximum weighted matching, a significant amount of work is needed to obtain the minimum weight vertex cover, and, from it, the corresponding dense subgraph in G.

Summarizing, the above approaches have the following disadvantages.

1. The maximum (weighted) matching or maximum flow in $\tilde{G}$ does not directly yield dense subgraphs in G.

2. We need only *some* subgraph of a specific density, not necessarily a *most* dense one. Hence, in the flow based approaches, it is *not* necessary to find the maximum flow.

3. The above approaches provide no natural way of finding a minimal dense subgraph for arbitrary weighted graphs, for arbitrary densities or values of K.

We develop a more efficient method analogously based on a different optimization problem (see [19]), but which will be seen to address all of these drawbacks.

3 Finding a Dense Subgraph

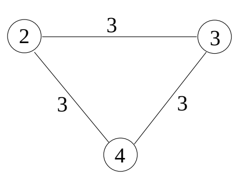
We devise a flow-based algorithm for finding dense subgraphs assuming that $K = 0$ in Equation (1). We discuss the case $K \neq 0$ in Section 4.

3.1 Construction of the Network

From the graph G, construct a bipartite *directed* network $G^* = (M, N, s, t, E^*, w)$, where M, N and E^* are as in $\tilde{G}$. The source s is connected by a directed edge to every node in M, and every node in N is connected by a directed edge to the sink t. The capacity of the network edge (s, e), $e \in M$, is the weight $w(e)$ of the edge e in G. The capacity of the network edge (v, t), $v \in N$, is equal to the weight $w(v)$ of the vertex v in G. The capacity of the network edge (e, v), $e \in M,\ v \in N$, is infinite. There are no other network edges. See also Figure 1.

Notice that the construction of the network extends to hypergraphs representing ternary or other constraints, where each hyperedge involves an arbitrary number of vertices.

A minimum cut in G^* *directly* defines a subgraph A that minimizes Expression (3). It can be found as the max flow using a netflow algorithm. Now we are only interested in finding a dense subgraph and not necessarily the *most* dense one. So, we are interested in a small enough cut in G^*, not necessarily the smallest one. Thus, to find a dense subgraph,



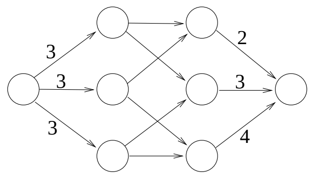
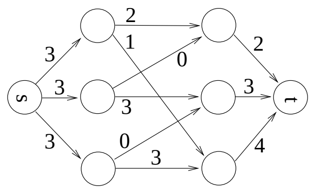


Figure 1: Constraint graph (left) and associated network (right).



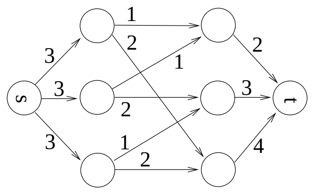


Figure 2: Two different flows for the constraint graph of Figure 1

there should be an algorithm that is faster than a general maximum flow (or minimum cut) algorithm.

The algorithm given in the next section relies on a subtle, but crucial modification of the incremental max flow algorithm which seems tailormade for the current application in that it simultaneously addresses all the drawbacks of the previous algorithms mentioned in the previous section.

3.2 The Dense Algorithm

The idea of the algorithm (Algorithm Dense below) is to start with the empty subgraph G' of G and add to it one vertex at time. When a vertex v is added, consider the adjacent edges e incident to G'. For each e, (for ease of exposition, we assume the edges are binary) try to distribute the weight $w(e)$ to one or both of its endpoints without exceeding their weights; see also Figure 2. As illustrated by Figure 3, we may need to redistribute some of the flow later.

If we are able to distribute all edges, then G' is not dense. If no dense subgraph exists, then the algorithm will terminate in $O(n(m+n))$ steps and announce this fact. If there is a dense subgraph, then there is an edge whose weight cannot be distributed even with

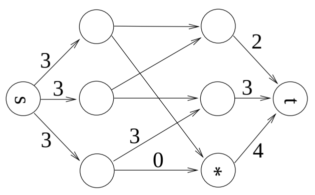


Figure 3: Initial flow assignment that requires redistribution later

redistribution. The last vertex added when this happens can be shown to be in all dense subgraphs $A \subseteq G'$.

Distributing an edge e in G now corresponds to pushing a flow equal to the capacity of (s, e) from s to t in G^*. This is possible either directly by a path of the form $\langle s, e, v, t\rangle$ in G^*, or it might require flow redistribution achieved by a standard search for augmenting paths [9], using network flow techniques, see Figures 4, 5. Note that the search for augmenting paths takes advantage of the fact that the flow through each vertex in M is distributed to exactly 2 vertices in N (lines 4-7) in Algorithm Distribute. While this decreases running time by a constant factor, it doesn't affect complexity.

If there is an augmenting path, then the resulting flows in G^* provide a distribution of the weight of each edge e in the current subgraph G' consisting of the examined vertices and edges of the original graph G as follows: the weight $w(e)$ of each edge e connecting the vertices a and b is split into two parts f_e^a and f_e^b such that $f_e^a + f_e^b = w(e)$ and, for each vertex $v \in G'$, $\sum_{e=(v,*)} f_e^v \leq w(v)$.

If there is no augmenting path for the residual flow on (s, e), i.e, the flow $w(e)$ is undistributable, then a dense subgraph has been found and is identified based on the flows in G^* starting from e.

Algorithm Dense

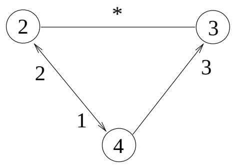
```
1.  $G' = \emptyset$.
2.  **for every** vertex $v$ **do**
3.      **for every** edge $e$ incident to $v$ and to $G'$ **do**
4.          Distribute the weight $w(e)$ of $e$
5.          **if** not able to distribute all of $w(e)$ **then**
6.              $A$ = set of vertices labeled during Distribute
7.              **goto** Step 12
8.              **endif**
9.          **endfor**
10.     add vertex $v$ to $G'$
11.     **endfor**
12.     **if** $A = \emptyset$ **then** no dense subgraph exists
13.     **else** $A$ is a dense subgraph
```

Algorithm Distribute searches for augmenting paths in G^* to achieve the required flow and the labeling. It repeats a Breadth First Search for augmentation until all of $w(e)$ has been distributed or until there is no augmenting path. The technique is similar to the one used in the max-flow algorithm in [19].

Algorithm Distribute

Input: $(G^*, f, edge)$, where $G^* = (N, M, s, t, E^*, w)$, f is a set of flows f_e^v
and $edge$ is the edge that is being distributed.

```
0.  Initialize $scan(v) = 0, label(v) = 0, scan(e) = 0, label(e) = 0$ for all $v \in N, e \in M$
```



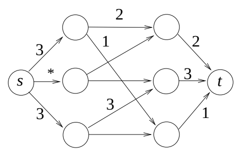
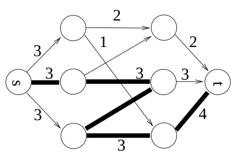


Figure 4: Current graph G' and corresponding network G^*, the edge marked by asterisk is currently being distributed



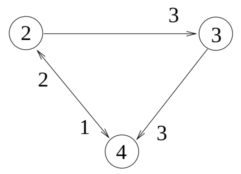


Figure 5: The augmenting path and the distribution of edges in original graph G'

```
1.  $vert = 0,\ capvert = 0$
2.  $label(edge) = 1,\ pathcap(edge) = w(edge)$
3.  **while** $(w(edge) > \sum_v f^v_{edge})$ **or** not all labeled nodes have been scanned
4.      **for** all labeled $e \in M$, with $scan(e) = 0$
5.          label unlabeled neighbors of $e$ (i.e $v \in N$)
6.          $scan(e) = 1,\ pred(v) = e,\ pathcap(v) = pathcap(e)$
7.      **endfor**
8.      **for all** labeled $v \in N$ with $scan(v) = 0$
9.          **if** $\min(w(v) - \sum_e f^v_e,\ pathcap(v)) > capvert$ **then**
10.             $vert = v,\ capvert = min(w - \sum_e f^v_e,\ pathcap(v))$
11.             **else**
12.                 label all unlabeled $e' \in M$ s.t $f^v_{e'} > 0$
13.         **endif**
14.             $scan(v) = 1$
15.     **endfor**
16.     **if** $vert > 0$ **then**
17.         An augmenting path from $s$ to $t$ has been found: backtrack from
        $vert$ using $pred()$ and change the values of $f^v_e$ as requirted.
18.         **for all** $e \in M$, $v \in N$
19..            $label(e) = 0, scan(e) = 0, label(v) = 0, scan(v) = 0$
20.         **endfor**
21.         $vert = 0, capvert = 0, label(edge) = 1$
22.         $pathcap(edge) = w(edge) - \sum_v f^v_{edge}$
23.     **endif**
24. **endwhile**
```

Lemma 1

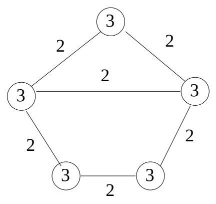


Figure 6: This subgraph is dense for $K = -4$, so is upper triangle

Let G^* be the bipartite network constructed from G, and $e \in M$. If, after checking all possible augmenting paths originating at e, the flow through (s, e) is less than the capacity of (s, e), and $A = (E_A, V_A)$ is the set of edges and vertices labeled after the search for an augmenting path, then $d(A) > 0$.

Proof: A is a subgraph of G because for every labeled edge $e \in E$ both of its vertices will be labeled. For all $v \in V_A$, the network edges (v, t) are saturated, otherwise there is an augmenting path from e to v and the flow through (s, e) can be increased. Let f be the maximum flow through (E_A, V_A). Since all (v, t) are saturated, $f = \sum_{v \in A} w(v)$, but since at least one edge (s, e) is not distributed $f < \sum_{e \in A} w(e)$; therefore $d(A) = \sum_{e \in A} w(e) - \sum_{v \in A} w(v) > 0$. □

The correctness of Algorithm Dense follows from the above Lemma, since if the graph contains any dense subgraph, Algorithm Dense will find it.

Complexity Analysis

In the worst case, constructing an augmenting path labels at most $m + n$ nodes. Since the algorithm stops when the total edge weight exceeds the total vertex weight, the total edge weight that is distributed is at most the total vertex weight times a constant bound b. Each augmentation increases the flow by least 1 unit. Therefore, the number of augmentations cannot exceed $O(n)$. Hence, Algorithm Dense has complexity $O(n(m + n))$.

3.3 Finding a Minimal Dense Subgraph

Let $G' = (V', E')$ be the subgraph already examined by Algorithm Dense. That is, assume that the vertices V' have been examined and the weight $w(e)$ of all induced edges e has been distributed. Let v be the first vertex that is about to be examined next, such that the weight of one of its incident edges e adjacent to G' cannot be distributed. Let $V_A \subseteq V'$ be the set of vertices labeled while trying to distribute $w(e)$, (which includes the vertex v), and let A be the subgraph induced by V_A. By Lemma 1, A is a dense subgraph.

Lemma 2
Every dense subgraph of A contains v.

Proof: Let A' be a dense subgraph of A not containing v. Then there should be an edge $e \in A'$ such that e was not distributed before v was considered. However, this contradicts our assumption that all edges in G' have been distributed. □

Remark
Similarly, if $(v, v_1), (v, v_2), ..., (v, v_k)$ are undistributed edges of v then every dense subgraph of A contains at least one edge from this list. If $k = 1$ then every dense subgraph of A contains (v, v_1).

Proposition 3
If the amount of undistributable flow, i.e, the density of A is $d(A)$ and A' is a dense proper subgraph of A, then $0 < d(A') < d(A)$ (in general, $K < d(A') < d(A)$).

Proof: Note that the excess flow comes from the edges incident to v. Suppose $A' \subseteq A$ is dense and $d(A') \geq d(A)$. By Lemma 2, A' contains v. Consider the relative complement A^* of A' with respect to A. Then $d(A^*) \leq 0$, which implies that the vertices of A^* could not have been labeled after distributing the flow of the edges of v. Since all vertices in V_A are labeled, we know that $A = A'$. □

Corollary 4
If $d(A) = 1$ then A is minimal. In general, if $d(A) = K + 1$, then A is minimal.

In particular, when $K = 0$, well-constrained or underconstrained problems have $d(A) \leq 1$. Then, by Corollary 4, we know that the subgraph found by Algorithm Dense is minimal. Moreover, if overconstrained problems are rejected, then a first test for overconstrained would be to determine $\sum_{e \in G} w(e) - \sum_{v \in G} w(v) > 1$ in linear time. This test would reject many overconstrained problems. The remaining cases would be found by noting whether $d(A) > 1$ when Algorithm Dense terminates.

We may accept consistently overconstrained problems. In that case, the graph A may have to be analyzed further to extract a minimal dense subgraph. We now develop a method for performing this extraction, once a dense subgraph A has been found by Algorithm Dense and $d(A) > 1$. The algorithm to be developed post-processes only the subgraph A.

Without loss of generality, assume that A contains the vertices $\{v_1, \ldots, v_l, v_{l+1}\}$, and v_{l+1} was the last vertex examined when A was found. The density $d(A)$ is the total undistributed weight of the edges between v_{l+1} and $\{v_1, \ldots, v_l\}$. We begin with the knowledge of a subgraph B of A that is contained in *every* dense subgraph of A. By Lemma 2, B contains initially the vertex v_{l+1}. The algorithm to be developed is to determine either an enlargement of the graph B, or else a reduction of the graph A.

We perform the following step iteratively. Choose a vertex $v_k \notin B$ from A. Determine the quantity $c = d(A) - w(e') + f_{e'}^{v_k} + f_{e'}^{v_{l+1}}$ where e' is the edge (v_k, v_{l+1}). That is, c is the undistributed weight of edges in A without v_k. Remove the vertex v_k from A along with its edges. This would create unutilized capacity in the set of vertices adjacent to v_k (that are in A) through the set E_k of incident edges. The excess vertex capacity is

$$\sum_{e \in E_k} w(e) - w(v_k) - w(e') + f_{e'}^{v_k} + f_{e'}^{v_{l+1}}$$

where e' is the edge between v_k and v_{l+1}. This quantity is the total flow on the edges of v_k, distributed away from v_k. We now attempt to distribute the previously undistributed weight of the edges between v_{l+1} and $\{v_1, \ldots, v_l\} - \{v_k\}$, using redistribution if necessary. We use

Algorithm Distribute on the modified network, setting the capacity of (v_k, t) to zero. There are two outcomes possible:

1. If we distribute all of c successfully into the newly created holes, or excess capacity on the vertices adjacent to v_k, then no subgraph of $A - v_k$ is dense, so v_k belongs in every dense subgraph of A, and hence gets restored into A and, moreover, gets added to B.

2. If we were unable to distribute c, then by Lemma 1, we have found a smaller dense subgraph of $A - v_k$. This new subgraph consists only of the vertices labeled by the Algorithm Distribute in the process of distributing one of the undistributed edges adjacent to v_{l+1}. This outcome reduces the size of A. Note that, by Proposition 3, the density (and size) of the new graph A must drop by at least 1.

We repeat this process for the remaining vertices in $A - B$. We stop either when $d(B) > 0$, because then B is minimal dense, or when $d(A) = 1$, because then A is minimal dense.

Complexity Analysis

The complexity of each iteration described above is $O(n(m + n))$, since c represents the undistributed weight on at most n edges adjacent to v_{l+1} are distributed at each iteration. We can assume that the sum of capacities of the edges is constant, thus the determination of a minimal dense subgraph takes $O(n^2(m + n))$ steps. Note, however, that by Proposition 3 the actual complexity rarely reaches this upper bound.

The complexity of the iteration is reduced to $O(m + n)$ if the constraint graph has bounded valence or if $d(A)$ has an a-priori constant bound. The latter situation means that the constraint problem has a bound on the "overconstrainedness" of subgraphs, a natural assumption if the constraint problem is specified interactively and we keep track of the density of the full constraint graph. In those cases, the complexity reduces to $O(n(m + n))$ steps.

Algorithm Minimal

Comment: The input is the output of Dense, a dense (sub)graph A of G, and the distribution of edge weights f_e^a and f_e^b for each edge $e = (a, b)$. Note that v_{l+1} is the last vertex added that caused A to be found, and e' is the edge between v_k and v_{l+1}.

```
1.  $B = \{v_{l+1}\}$
2.  **while** $d(B) \leq 0$ and $d(A) > 1$ **do**
3.      choose $v_k \in A - B$
4.      $c = d(A) - w(e') + f_{e'}^{v_k} + f_{e'}^{v_{l+1}}$
5.          **for** all $v \in N$ (Removing $\{v_k\}$ from $A$)
6.              Let $e = (v, v_k)$
7.              remove $e$ from $M$
8.          **endfor**
```

```
 9.         remove $v_k$ from $N$
10.     Distribute (in $A$) excess $c$ from the edges of $v$
11.     **if** there are some undistributed edges left **then**
12.         set $A =$ new labeled graph
13.     **else**
14.         set $A = A \cup \{v_k\}$ (as well as restoring edges of $v_k$)
15.         set $B = B \cup \{v_k\}$
16.     **endif**
17. **endwhile**
18. **output** $B$, **if** $d(B) > 0$, **else output** $A$.
```

4 The Case of $K \neq 0$

In the context of geometric constraint solving, $K = 0$ represents the case where the minimal dense subgraph corresponds to elements that can be placed rigidly with respect to each other and to a global coordinate system. The case of $K > 0$ means that the subgraph is overconstrained, usually by K constraints, and the case of $K < 0$ means that the resulting geometric configuration has residual motion with respect to a global coordinate system.

The most important case is when $K = -3$ in the planar case or $K = -6$ in the spatial case, signaling that the resulting geometric configuration can move as a rigid body with respect to the coordinate system. In [10], this property has been exploited recursively for the purpose of cluster combination. The case $K < 0$ is also the reason that symbolic algebraic decomposition methods fail to succeed as they implicitly assume $K = 0$.

As presented, our algorithms satisfy Inequality (1) for $K = 0$: keep adding vertices until we are unable to distribute the edge weight/capacity. The first undistributable edge signals a dense graph A, for $K = 0$, and $d(A) > d(A - v)$, where v was the last vertex examined. We now explain how to modify the algorithm to accommodate different values of K.

The modification for $K > 0$ is trivial. Instead of exiting Algorithm Dense when an edge cannot be distributed, exit when the total undistributable edge capacity exceeds K. The computation of the total undistributable edge capacity so far is based only on the weights of the labeled edges and vertices, thus ensuring that the resulting dense graph is connected. An analogous change is in order for Algorithm Minimal. Clearly this modification does not affect the performance complexity of the algorithms.

When $K < 0$ the algorithms can also be modified without increasing the complexity. Suppose, therefore, that $K < 0$, and consider Step 4 of Algorithm Dense. If $w(v) + K \geq 0$, simply reduce the capacity of the network edge (v, t) to $w(v) + K$ and distribute $w(e)$ in the modified network. If the edge cannot be distributed, then the subgraph found in Step 6 has density exceeding K. If every incident edge can be distributed, then restore the capacity of the network edge when adding v to G'.

If the weight $w(v)$ of the added vertex v is too small, that is, if $w(v) + K < 0$, then a more complex modification is needed. We set the capacity of (v, t) to zero. Let e be an new edge to be distributed, and do the following.

```
1.  Distribute the edge weight $w(e)$ in the modified network.
2.  if $w(e)$ cannot be distributed then
3.      we have found a dense subgraph for $K$; exit.
4.  else
5.      save the existing flow for Step 10.
6.      increase the flow of $e$ by $-(w(v)+K)$
7.      if the increased flow cannot be Distributed then
8.          we have found a dense subgraph for $K$; exit.
9.      else
10.         restore the old flow. No dense subgraph found
11.     endif
12. endif
```

In worst case the algorithm saves and restores the flows for every edge added, which requires $O(m)$ operations per edge. Distributing the edge flow however dominates this cost since it may require up to $O(m+n)$ operations per edge; so the modification does not adversely impact asymptotic performance.

5 Graph Decomposition

The flow-based degree-of-freedom analysis must be applied repeatedly to constraint graphs. Two different conceptual steps are involved. When a minimal subgraph of the appropriate density has been found, then the vertices form a *cluster* of geometric elements that can be placed rigidly with respect to each other, using only the constraints represented by the edges of the subgraph. This cluster can be extended under certain circumstances by adding more geometric elements that are determined by constraints involving the cluster. After a cluster has been so extended, it must then be abstracted into a single geometric entity, and the rest of the constraint graph can be searched for another minimal dense subgraph.

5.1 Extended Clusters

Consider the constraint graph G of Figure 7. We assume that all vertices have weight 2 and all edges weight 1. Then the vertex set $\{a,b\}$ induces a minimal dense subgraph of G. After solving this subgraph, i.e., after assigning coordinates to the geometric elements a and b so that the constraint between them is satisfied, we can place c because its two coordinates (the weight) are determined by the two incident constraints with a and b. After so placing c, we can place, in sequence, d, e and f. Thus, all six geometric elements can be placed rigidly relative to each other.

The structure of the solution is as follows: Beginning with a simultaneous system of equations, the elements of the minimal dense subgraph are placed with respect to each other (and the global coordinate system in the case of $K=0$). In our example, two geometric elements are placed. Then, a number of geometric elements are placed, one by one, solving

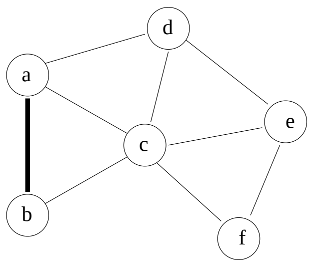


Figure 7: Constraint graph with vertices of weight 2 and edges of weight 1. The minimal dense subgraph {a,b} can be extended sequentially by the other elements, in alphabetic order.

for each individually from as many equations as there are incident constraints on them with the previously placed elements. In our example, each element requires solving two equations in two variables.

We call a minimal dense subgraph that has been enlarged by sequential extensions an *extended dense subgraph*.

5.2 Recursive Application

After an extended dense subgraph has been found, the set of geometric elements that are its vertices form a rigid geometric structure. For $K = 0$, this structure is fixed with respect to the coordinate system, otherwise the structure can be moved with $K > 0$ degrees of freedom.

We extend the decomposition and find other dense subgraphs, abstracting this subgraph into a geometric construct with K degrees of freedom. This can be done applying *natural graph reduction*: Replace the dense subgraph G_0 by a vertex u of weight K and combine all edges from a vertex in G_0 to a vertex w not in G_0. The weight of the induced edge (u, w), in the reduced graph, is the sum of weights of the edges that have been combined. After the reduction, another dense subgraph is found. Reduction ends when the graph has been reduced to a single vertex.

For example, consider the graph of Figure 8. All vertices have weight 2, all edges have weight 1. The four vertices connected by the heavy edges constitute an extended dense subgraph. After reduction of the extended dense subgraph we obtain the graph of Figure 9. Again, an extended cluster is found and indicated by the heavy edges in the graph. Reducing this cluster, we obtain the graph of Figure 10. In this graph, a minimal cluster is found as indicated, and further reduction yields the graph of Figure 10. Here, the entire graph is minimal dense, so the last reduction yields a graph that consists of a single vertex.

5.3 Church-Rosser Property

A constraint graph G is *well-constrained* if the density of G is $d(G) = -K$ and there is no vertex-induced subgraph of density greater than $-K$. We want to prove that a well-constrained graph can be reduced iteratively to a single vertex, no matter how the dense

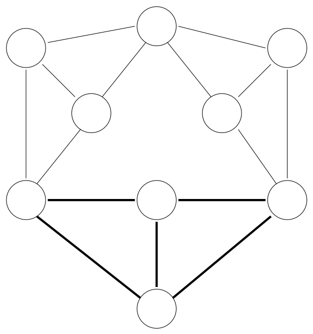


Figure 8: Constraint graph with an extended dense subgraph. Vertex weight is 2, edge weight is 1.

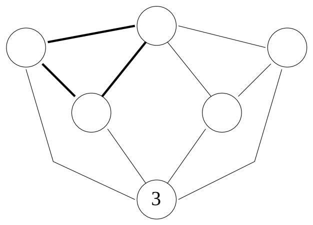


Figure 9: Constraint graph after reducing the subgraph. The cluster vertex has weight 3.

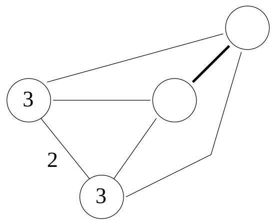


Figure 10: Constraint graph after two subgraph reductions. The cluster vertices have weight 3 each, the edge between them has weight 2.

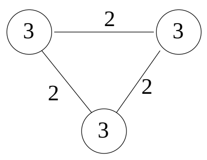


Figure 11: Constraint graph after three subgraph reductions.

subgraph is chosen by our algorithms. Intuitively this fact is a consequence of the subgraph reduction preserving the density of the graph.

If a well-constrained geometric constraint graph can be reduced, by a sequence of the reduction steps described before, to a single vertex, then the geometric constraint problem is solved by the corresponding decomposition of the nonlinear equation system. We want to show the converse: If a generic geometric constraint system is well-constrained and has no overconstrained subsystems, then *any* sequence of reduction steps described before will reduce the constraint graph to a single vertex.

The reduction algorithm considers a graph G_i, finds a well-constrained subgraph H_{i+1} and reduces G_i to G_{i+1}. We denote such a reduction step by

$$G_i \longrightarrow_{H_{i+1}} G_{i+1}$$

The reduction terminates when no dense subgraph can be found.

Theorem Let G be a well-constrained subgraph; that is $d(G) = -K$, and every vertex-induced subgraph A has density $d(A) \leq -K$. Consider any complete reduction sequence

$$G = G_0 \longrightarrow_{H_1} G_1 \longrightarrow_{H_2} \ldots \longrightarrow_{H_m} G_m$$

produced by our decomposition method, where the reduction halts with G_m. The subgraphs H_i are nontrivial and have been located by the Algorithm Dense. Then $|G_m| = 1$.

Proof. The proof follows immediately from the fact that the process of condensing does not change the density. In the reduction

$$G \longrightarrow_H G'$$

we replace H with a vertex h whose weight is $w(h) = d(H)$. Moreover, we remove the edges (r, u), where $r \in H$ and $u \notin H$ and add the edge weight to the edge (h, u) of G'. Thus $d(G) = d(G')$, and therefore $d(G) = d(G_0) = d(G_1) = \ldots = d(G_m)$. Since G is well-constrained, it follows that G_m is also well-constrained. Since the decomposition method halts only when the current (reduced) graph does not have a well-constrained subgraph of size greater than 1 (by the correctness of Algorithm Dense), we can assume that G_m has no well-constrained subgraph (including itself) of size greater than 1. Therefore, $|G_m| = 1$, thus proving the theorem. □

References

[1] S. Ait-Aoudia, R. Jegou, and D. Michelucci. Reduction of constraint systems. In *Compugraphics*, pages 83–92, 1993.

[2] W. Bouma, I. Fudos, C. Hoffmann, J. Cai, and R. Paige. A geometric constraint solver. *Computer Aided Design*, 27:487–501, 1995.

[3] J. Canny. Improved algorithms for sign determination and existential quantifier elimination. *Computer Journal*, 36(5), 1993.

[4] S. C. Chou, X. S. Gao, and J. Z. Zhang. A method of solving geometric constraints. Technical report, Wichita State University, Dept. of Computer Sci., 1996.

[5] G.E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Lect. Notes in CS, Vol 33*, pages 134–183. Springer-Verlag, 1975.

[6] G. Crippen and T. Havel. *Distance Geometry and Molecular Conformation.* John Wiley & Sons, 1988.

[7] N. Vorobjov D. Grigoriev. Solving systems of polynomial inequalities in subexponential time. *J. Symbolic Computation*, 5, 1988.

[8] S. Even and R. Tarjan. Network flow and testing graph connectivity. *SIAM journal on computing*, 3:507–518, 1975.

[9] L.R. Ford and D.R. Fulkerson. *Flows in Networks.* Princeton Univ. Press, 1962.

[10] I. Fudos. *Geometric Constraint Solving.* PhD thesis, Purdue University, Dept of Computer Science, 1995.

[11] T. Havel. Some examples of the use of distances as coordinates for Euclidean geometry. *J. of Symbolic Computation*, 11:579–594, 1991.

[12] Christoph Hoffmann, Andrew Lomonosov, and Meera Sitharam. Finding solvable subsets of constraint graphs. In *Principles and Practice of Constraint Programming – CP97*, pages 463–477. Springer LNCS 1330, 1997.

[13] Christoph M. Hoffmann and Pamela J. Vermeer. Geometric constraint solving in R^2 and R^3. In D. Z. Du and F. Hwang, editors, *Computing in Euclidean Geometry.* World Scientific Publishing, 1994. second edition.

[14] Christoph M. Hoffmann and Pamela J. Vermeer. A spatial constraint problem. In *Workshop on Computational Kinematics*, France, 1995. INRIA Sophia-Antipolis.

[15] Ching-Yao Hsu. *Graph-based approach for solving geometric constraint problems.* PhD thesis, University of Utah, Dept. of Comp. Sci., 1996.

[16] I. Emiris J. Canny. An efficient algorithm for the sparse mixed resultant. In O. Moreno ed.s G. Cohen, T. Mora, editor, *Proc. 10th Intern. Symp. on Applied Algebra, Algebraic Algorithms, an d Error Correcting Codes*, volume 263, pages 89 – 104. Lecture Notes in Computer Science, Springer-Verlag, 1993.

[17] A.G. Khovanskii. Newton polyhedra and the genus of complete intersections. *Funktsional'nyi Analiz i Ego Prilozheniya*, 12(1):51–61, 1978.

[18] R. Latham and A. Middleditch. Connectivity analysis: a tool for processing geometric constraints. *Computer Aided Design*, 28:917–928, 1996.

[19] E. Lawler. *Combinatorial optimization, networks and Matroids.* Holt, Rinehart and Winston, 1976.

[20] D. Lazard. Résolution des systèmes d'équations algébriques. *Theoretical Computer Science*, 15:77–110, 1981.

[21] D. Lazard. A new method for solving algebraic systems of positive dimension. *Discrete Applied Mathematics*, 33(1):147–160, 1991.

[22] J. Owen. Algebraic solution for geometry from dimensional constraints. In *ACM Symp. Found. of Solid Modeling*, pages 397–407, Austin, Tex, 1991.

[23] J. Owen. Constraints on simple geometry in two and three dimensions. In *Third SIAM Conference on Geometric Design.* SIAM, November 1993. To appear in Int J of Computational Geometry and Applications.

[24] J.A. Pabon. Modeling method for sorting dependencies among geometric entities. In *US States Patent 5,251,290*, Oct 1993.

[25] J. Renegar. On the computational complexity and the first order theory of the reals, part i. *Journal of Symbolic Computation*, 13:255–299, 1992.

[26] T.L. Magnanti R.K. Ahuja and J.B. Orlin. *Network Flows.* Prentice-Hall, 1993.

[27] O. E. Ruiz and P. M. Ferreira. Algebraic geometry and group theory in geometric constraint satisfaction for computer-aided design and assembly planning. *IIE Transactions on Design and Manufacturing*, 28:281–294, 1996.

[28] B. Sturmfels. Sparse elimination theory. In *Proc. Computational Algebraic Geometry and Commutative Algebra*, pages 377 –396. Cambridge University Press, 1993.

[29] D. Wang. An elimination method for polynomial systems. *J. Symbolic Computation*, 16:83–114, 1993.