GRAPH AND COMBINATORIAL ALGORITHMS FOR GEOMETRIC CONSTRAINT SOLVING

By

ANDREW LOMONOSOV

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2004

Dedicated to H.

ACKNOWLEDGMENTS

I am mostly indebted to my advisor, Dr. Meera Sitharam, who has been supervising me with patience and everlasting interest. In the beginning she offered me a very extensive choice of research problems and was willing to spend considerable time explaining the nature and background of these problems, so I was able to choose the area that personally appealed to me. While all these problems were interesting and challenging, the problem statements were easy to understand, so one could start thinking about them right away, learning necessary technical skills along the way. My advisor has given me every opportunity to acquire these technical skills by spending a lot of her time meeting me, running relevant special topics seminars, introducing me to experts in the field and sending me to plenty of conferences and workshops.

While allowing for my independent growth, my advisor was trying to make the entire research process as efficient as possible. When working on one problem she always kept another one on "backburner", so if the progress on one problem slowed down somewhat then I could always turn to the second problem. This arrangement was made possible by her contagious enthusiasm about the research topics, enthusiasm that was highly motivating whenever I would feel temporarily bogged down.

Throughout the research process my advisor was mindful of my future career, and was steering me toward learning proper mix of theoretical and practical skills. Also in all the classes where I was her Teaching Assistant, she always took trouble to explain the intricacies and potential pitfalls of the teaching process.

To some advisors the dissertation is not the only final product, the individual growth of a student is at least as important. I am very happy that Dr. Meera Sitharam is one such advisor.

I would also like to thank my committee members Drs. Sartaj Sahni, Gerhardt Ritter, Tim Davis, Sanjay Ranka and Ravi Ahuja for discussions we had and their helpful suggestions regarding my research, especially concerning network flow related issues and relationships between various problems considered.

I would like to thank Dr. Christopher Hoffman for various assistance he has provided during our joint work.

I would like to thank Dr. Pardalos as well as Burak and Sandra Eksioglu for helping me with step-cost function approaches to obtaining approximate solution of minimum dense problem.

Finally I would like to thank people with whom I have been working on Geometric Constraint Solving Team: Jianjun Oung, Naganandhini Kohareswaran, Yong Zhou, Aditee Kumthekar, Ramesh Balasubramaniam and Heping Gao for helping me with various aspects of this large research area.

TABLE OF CONTENTS

LIST OF FIGURES

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

GRAPH AND COMBINATORIAL ALGORITHMS
FOR GEOMETRIC CONSTRAINT SOLVING

By

Andrew Lomonosov

May 2004

Chair: Meera Sitharam
Major Department: Computer and Information Science and Engineering

Geometric constraints are at the heart of CAD/CAM applications and also arise in many geometric modeling contexts such as virtual reality, robotics, molecular modeling, teaching geometry, etc.

Informally, a geometric constraint problem consists of a finite set of geometric objects and a finite set of constraints between them. The geometric objects are drawn from a fixed set of types such as points, lines, circles and conics in the plane, or points, lines, planes, cylinders and spheres in 3 dimensions. The constraints are spatial and include logical constraints such as incidence, tangency, perpendicularity and metric constraints such as distance, angle, radius. The spatial constraints can usually be written as algebraic equations whose variables are the coordinates of the participating geometric objects. A solution of a geometric constraint problem is a real zero of the corresponding algebraic system.

Currently there is a lack of effective spatial variational constraint solvers and assembly constraint solvers that scale to large problem sizes and can be used interactively by the designer as conceptual tools throughout the design process.

The requirement is a constraint solver that uses geometric domain knowledge to develop a plan for decomposing the constraint system into small subsystems, whose solutions can be recombined by solving other small subsystems. The primary aim of this decomposition plan is to restrict the use of direct algebraic/numeric solvers to subsystems that are as small as possible. Hence the optimal or most efficient decomposition plan would minimize the size of the largest such subsystem. Any geometric constraint solver should first solve the problem of efficiently finding a close-to-optimal decomposition-recombination (DR) plan, because that dictates the usability of the solver.

In this thesis we state this problem of finding a close-to-optimal solution as a problem that deals with weighted graphs and also identify several important subproblems. One class of such subproblem involves finding dense subgraphs - graphs such that sum of weights of its edges is greater than sum of weights of its vertices. Dense graphs that present interest for finding a DR-plan are (a) minimum (smallest possible dense graphs), (b) minimal (not containing any other dense subgraphs), (c) maximum (largest dense ones), (d) maximal (not contained in any other dense subgraph).

This thesis presents polynomial time algorithms for problems (b), (c) and (d). Problem (a) is shown to be NP-complete, and various approximation algorithms are suggested, as well as explicit solutions for special cases that arise from CAD/CAM applications.

CHAPTER 1
INTRODUCTION

1.1 Problem Definitions

1.1.1 Dense Graphs

Motivation. Suppose that we wanted to find a smallest subgraph of a given graph that has at least twice the number of edges as vertices. Or three times. Or 5 times more edges than vertices.

Given: an edge and vertex weighted graph $G = (V, E)$ and a constant K. Weights of vertices and edges are denoted by $w(v), w(e)$ respectively.

A graph G is called *dense* if $\sum_{e \in G} w(e) - \sum_{v \in G} w(v) \geq K$

Function $d(G) = \sum_{e \in G} w(e) - \sum_{v \in G} w(v)$ is called *density* of G.

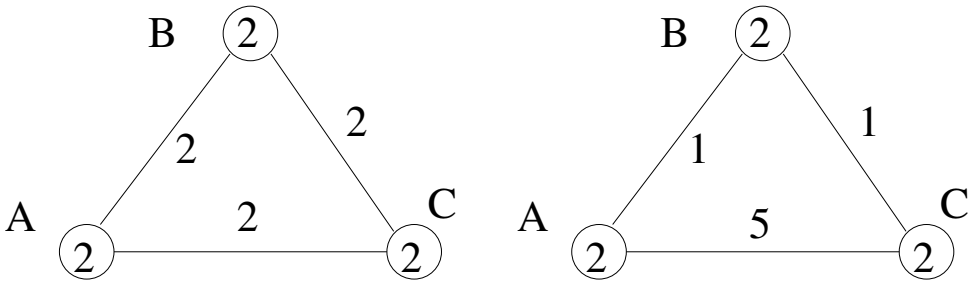


Figure 1–1: Nondense and dense graphs (K=1)

In many applications we are interested in finding a subgraph whose density is "uniform", i.e., not contributed by some small overly dense part. Density of this graph is "stable" or preserved even when some overly dense part is replaced by a barely dense part. Following definitions describe this concept.

1.1.2 Stably Dense Graph

A graph A that has $d(A) > K$ is *overconstrained*.

A graph G such that $d(G) = K$ and $\forall A \subseteq G, d(A) \leq K$ is *wellconstrained*.

A graph G is *stably dense* if $d(G) \geq K$ and after replacing any of it over-constrained nontrivial subgraph of by well-constrained subgraph G remains dense. For example graph ABCDEF in Figure 1–2 is dense but not stably dense $(K = -3, w(v) = 2, w(e) = 1, \forall v, w)$ because modified graph AFG is not dense anymore.

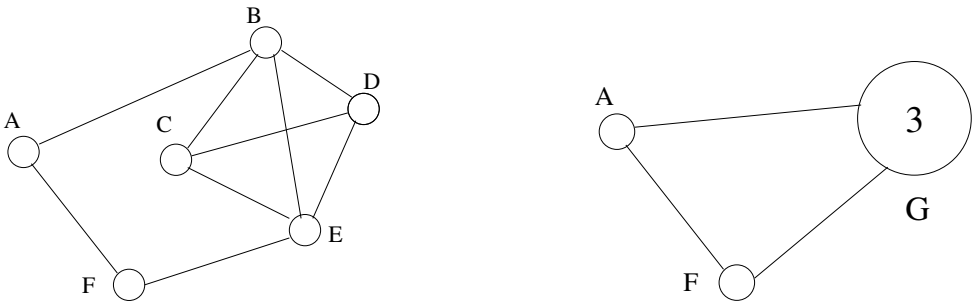


Figure 1–2: Graph ABCDEF is dense but not stably dense (K=-3, w(v)=2,w(e)=1)

Many applications require finding smallest or largest subgraph (unweighted) which has twice the number of edges as vertices. If we call graphs that have this property "dense" (by setting weights of vertices and edges appropriately) then the problem becomes that of finding "smallest" or "largest dense subgraph." Following definitions formalize these notions.

1.1.3 Minimal Stably Dense Subgraph Problem

A stably dense graph A is called *minimal stably dense* if $\nexists B \subset A$, s.t. B is stably dense. Given a graph G, a *minimal stably dense subgraph problem* involves locating a minimal stably dense subgraph $A \subseteq G$, if such A exists.

Note that A is minimal stably dense iff A is minimal stably dense.

1.1.4 Maximal Stably Dense Subgraph Problem

A stably dense graph A is called *maximal stably dense* if $\nexists B, A \subset B \subseteq G$, s.t. B is stably dense. Given a graph G, a *maximal stably dense subgraph problem* involves locating a maximal stably dense subgraph $A \subseteq G$, if such A exists.

1.1.5 Maximum Stably Dense Subgraph Problem

A largest (in terms of the number of vertices) maximal stably dense subgraph is called *maximum stably dense*. Given a graph G, a *maximum stably dense subgraph problem* involves locating a maximum stably dense subgraph $A \subseteq G$, if such A exists.

Note that a graph G can have several maximum stably dense subgraphs (that have the same size).

1.1.6 Minimum Stably Dense Subgraph Problem

A smallest (in terms of the number of vertices) minimal stably dense subgraph is called *minimum stably dense*. Given a graph G, a *minimum stably dense subgraph problem* involves locating a minimum stably dense subgraph $A \subseteq G$, if such A exists.

Note that a graph G can have several minimum stably dense subgraphs (that have the same size). Also note that subgraph A is minimum dense if and only if A is minimum stably dense, and if A is minimum dense, then A is also minimal dense.

Minimizing number of edges vs number of vertices

There is a modification of Minimum Stably Dense Subgraph Problem, where we want to minimize number of edges (and not vertices) of stably dense subgraph A. Following results demonstrate that optimal solution of modified problem will yield a constant factor approximate solution of the original problem.

Let $A \subseteq G$ be the dense subgraph $d(A) \geq 1$, such that $|E(A)|$ is minimum over all dense subgraphs of G.

Let $B \subseteq G$ be the dense subgraph $d(B) \geq 1$, such that $|V(B)|$ is minimum over all dense subgraphs of G.

Lemma 1 *Number of edges* $|E(A)|$ *is at most* $3|V(A)|$*, similarly* $|E(B)|$ *is at most* $3|V(B)|$*.*

Proof Let k be an average degree of A, then $d(A) = |V(A)| * k/2 - 2 * |V(A)| \geq 1$. Since A is minimum dense, then removal of a vertex $v \in A$, such that v has smallest degree in A, will result in nondense graph $A \setminus v$. Degree of v could be at most k, so this forces $1 > d(A \setminus v) \geq |V(A)|(k/2 - 2 - 1)$ and therefore $k \leq 6$, and $|E(A)| = 3|V(A)|$. ∎

We can use this lemma to relate sizes of A and B

Claim 1 *For A and B defined above, $|V(A)| \leq 3/2|V(B)| - 1/2$.*

Proof

$$2|V(A)| + 1 \leq |E(A)| \leq |E(B)| \leq 3|V(B)|$$

First inequality is due to density of A, second inequality is due to A having smallest number of edges among all dense subgraphs and third inequality is due to Lemma 1. ∎

1.1.7 Examples of various Dense Subgraphs

Consider Figure 1–3. If $K = 0$ then BCDE is minimal (stably) dense, EDF is minimum (stably) dense, AGH is maximal stably dense and ABCDEF is maximum stably dense subgraph.

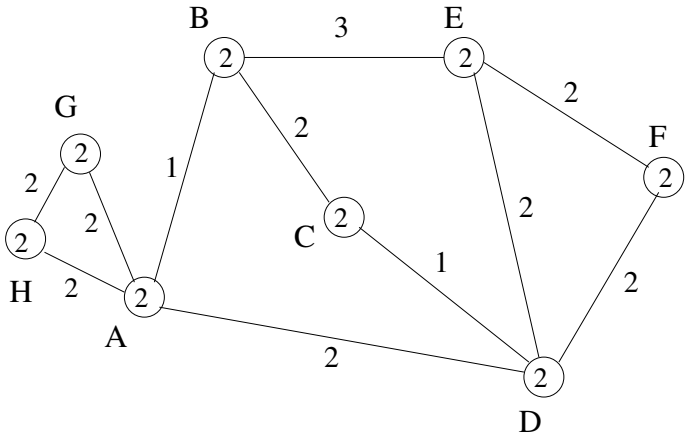


Figure 1–3: An edge/vertex weighted graph

1.1.8 Relationships between various Graph Problems

Here we will show that the problem of finding minimum (minimal) dense subgraph can be reduced in polynomial time to the problem of finding a maximum (maximal) dense subgraph.

Let X be a bipartite graph, comprised of two sets of vertices X_1 and X_2. Let A be a subset of X_1, a set of all vertices in X_2 that are connected to some vertex in A will be denoted by $I(A)$.

Now let $G = (V, E)$ be given graph. Let $BG = (V_B \cup E_B, F)$ be the bipartite graph corresponding to G, where V_B represents vertices V of G, E_B represents edges E of G, and in this graph BG, vertex $v \in V_B$ is connected to vertex $e \in E_B$ by an edge (in F) if and only if v is an endpoint of e in G.

Lemma 2 *Let A be a dense subgraph of G. Let $E_A \subseteq BG$ be the set corresponding to all edges of A. Then $A = I(E_A)$ and therefore $|E_A| - 3 * I(E_A) \geq K$ by definition of density of A. Hence finding largest E_A such that $|E_A| - 3 * I(E_A) \geq K$ would find maximum (number of edgeswise) dense graph.*

Lemma 3 *Let A be a dense subgraph of G. Let $C = V \setminus A$. Note that $I(C) = E \setminus E_A$. Hence $d(G) = |E_A| + I(C) - 3 * |V_A| - 3 * |C|$, and since $d(A) \geq K, d(G) - I(C) + 3*|C| \geq K$ and $3*|C| - I(C) \geq K - d(G)$. Therefore finding smallest dense subgraph, is equivalent to finding largest C such that $3*|C| - I(C) \geq K - d(G)$.*

Claim 2 *From previous two lemmas it follows that the problem of finding minimum (minimal) dense subgraph can be reduced in polynomial time to the problem of finding a maximum (maximal) dense subgraph. (Note that reverse reduction will not work due to the lack of symmetry in the bipartit graph - a vertex can have have any number of adjacent edges, but every edge has exactly two endpoints.)*

1.1.9 Optimal Complete Recursive Decomposition

A *recursive decomposition* of a graph G involves constructing a so-called *RD-dag* of G defined below.

A RD-dag of graph G is directed acyclic graph $R = (RE, RV)$ which has following properties.

The first property of R is that $OV \subseteq RV$, where OV is a copy of the entire set of vertices of G. Now let X be a node in R, then $U(X)$ denotes a set of vertices in MV such that there is an oriented path from v to X for every $v \in U(X)$. For example in Figure 1–4 $U(S_2) = \{C, D, E\}$ (in all examples in this subsection all edges of G have weight 1, all vertices have weight 2, constant $K = -3$). The second property of R is that for every sink vertex X of R, corresponding subgraph $U(X)$ is a maximal stably dense subgraph of G. The third property of R is that for every node S_i in R, with exception of nodes in OV, corresponding subgraph $U(S_i)$ is stably dense. The final property of R is that every S_i has a *cluster minimality property*, i.e., that the set $P(S_i)$ of all ancestors of S_i does not contain a proper nontrivial subset $P'(S_i) \subset P(S_i), |P'_{S_i}| \neq 1$ such that $\cup_{X \in P'(S_i)} U(X)$ is stably dense. For example in Figure 1–6 node S_2 does not have cluster minimality property (for proper subset $P' = S_1 \cup B, U(P') = BCDE$ is stably dense, hence Figure 1–6 is not a RD-dag.

Note that a graph G can have several different RD-dags, see Figure 1–4 and Figure 1–6.

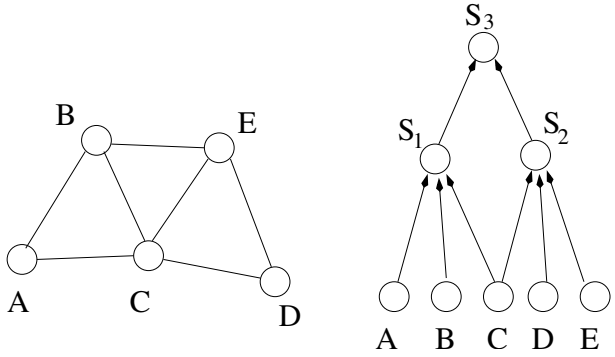


Figure 1–4: Original graph G and a corresponding RD-dag

A *complete recursive decomposition* of a graph G involves constructing a so-called *complete RD-dag* of G. A complete RD-dag of G is a RD-dag of G with

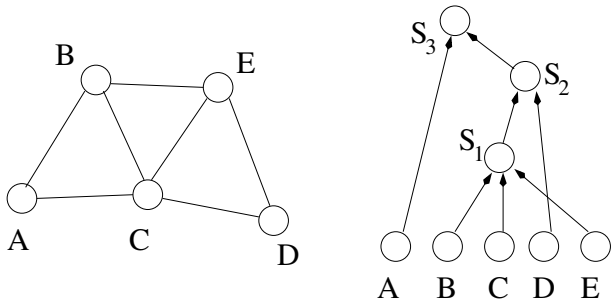


Figure 1–5: Another possible RD-dag

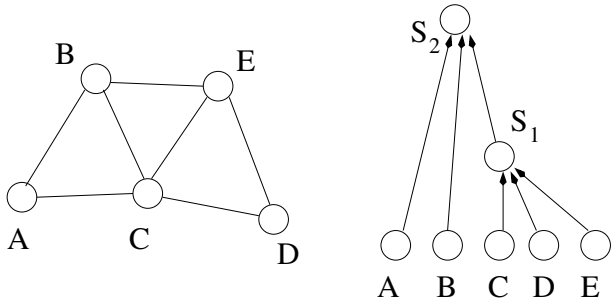


Figure 1–6: Original graph G and NOT a RD-dag

additional requirement that for every maximal stably dense subgraph M of G, there is a corresponding node $M' \in R$ such that $U(M') = M$. For example middle dag in Figure 1–6 is a RD-dag but not a complete RD-dag of G while the right dag is a complete RD-dag.

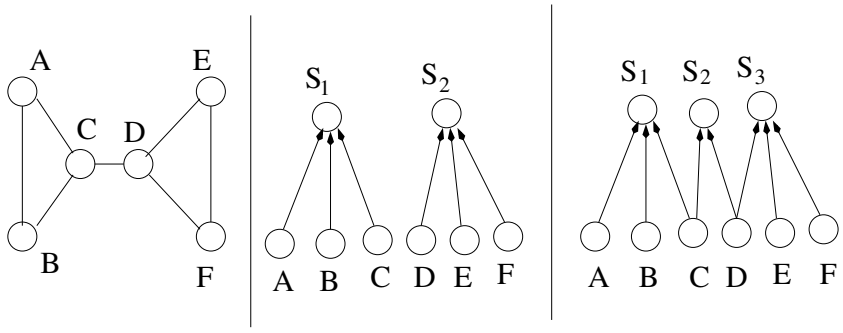


Figure 1–7: Original graph G, RD-dag and complete RD-dag

Since given graph G can have several possible RD-dags, an important measure of efficiency of a RD-dag R is it *maximum fan-in* defined as a maximum indegree among all vertices of R. We will provide motivation for this criteria in

Section 2.1.1. For example in Figure 1–4 maximum fan-in is 1. The problem of finding *optimal (complete) recursive decomposition* of a graph G involves finding (complete) RD-dag of G that has smallest maximum fan-in among all other (complete) RD-dags of G.

Nontrivial subgraphs. In practice we are only interested in finding maximal/minimal/minimum/maximum stably dense subgraphs of size 3 or more (i.e., not vertices or edges). Graphs of size 2 or less will be referred to as *trivial*.

1.2 Summary of Results

Numeric entries are sections where polynomial time algorithms (or NP-completeness) are demonstrated.

Problem	Bounded Parameter	Unbounded Version
Minimum (stably) dense	NP-complete, 4.2	NP-complete, 4.2
	special cases and appr 4.3, 4.4	
Maximal stably dense	3.2.3	3.2.3
Minimal (stably) dense	3.2.3	3.2.3
Maximum stably dense	3.4	3.4
Maximum dense	3.1	NP-complete, 3.1
Maximum number of edges, 4.1.1	N/A	NP-complete
Measures and Performance for GCS	2.1.6, 2.1.14	N/A
Complete recursive decomposition	3.3.10	Unknown

1.3 Related Work in Algorithms Community

Finding densest graph. The problem of finding subgraph $A \subseteq G$ of fixed size $|A| = k$ that maximizes $\sum w(e(A))$ is NP-Complete (could be shown by reduction from CLIQUE, see Asahiro and Iwama (1995)).

The problem of finding $A \subseteq G$ that maximizes $\frac{\sum w(e(A))}{\sum w(v(A))}$ can be solved in polynomial time using parametric flow techniques.

The problem of finding $A \subseteq G$, s.t $|A| \leq k$ for a given k, and A maximizes $\frac{\sum w(e(A))}{|A|}$ is NP-Complete (could be shown by reduction from CLIQUE).

Section 4.1.1 contains more details on the relationship between these and our graph problems.

Clique-like results. Let v and e be the number of vertices and edges of graph G respectively.

It was shown by (Asahiro and Iwama, 1995) that finding a subgraph of G that has at most a vertices and at least b edges is

NP-Complete for $b = a(a-1)/2$

NP-Complete for $a = v/2, b \leq e/4(1 + O(1/\sqrt{v}))$

NP-Complete for $a \leq v/2, b \leq a^{1+\epsilon}, b \leq e/4(1 + 0.45 + O(1))$

can be done in polynomial time for $a = v/2, b \leq e/4(1 - O(1/v)$ (by greedy algorithm)

Semidefinite programming based approaches. For a maximum number of edges problem (Feige and Seltzer, 1997) gives a semidefinite programming relaxation approximation algorithm with approximation ratio roughly $O(n/k)$.

(Goemans, 1996) studied a linear relaxation of the problem, getting approximation ratio of $O(n^{-1/2})$ when $k = n/2$.

(Srivastav and Wolf, 1999) used semidefinite programming relaxation to get approximation factor within .5 for $k = n/2$ and (Ye and Zhang, 1997) were able to design .5866 approximation factor SDP based algorithm for the case of $k = n/2$.

CHAPTER 2
APPLICATIONS TO GEOMETRIC CONSTRAINT SOLVING

2.1 Geometric Background

2.1.1 Geometric Constraint Problems

Geometric constraints are at the heart of computer aided engineering applications (see, e.g., Hoffmann (1997); Hoffmann and Rossignac (1996)), and also arise in many geometric modeling contexts such as virtual reality, robotics, molecular modeling, teaching geometry, and so on. In this dissertation we will be looking at geometric constraint systems primarily within the context of product design and assembly. Figure 2–3 illustrates those (boldface) components within a standard CAD/CAM/CAE master model architecture (Bronsvoort and Jansen, 1994; Kraker et al., 1997; Hoffmann and Joan-Arinyo, 1998) where our graph problems are relevant.

Informally, a *geometric constraint problem* consists of a finite set of geometric objects and a finite set of constraints between them. The geometric objects are drawn from a fixed set of types such as points, lines, circles and conics in the plane, or points, lines, planes, cylinders and spheres in 3 dimensions. The constraints are spatial and include logical constraints such as incidence, tangency, perpendicularity and metric constraints such as distance, angle, radius. The spatial constraints can usually be written as algebraic equations whose variables are the coordinates of the participating geometric objects.

A *solution of a geometric constraint problem* is a real zero of the corresponding algebraic system. In other words, a solution is a class of valid instantiations 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 dimensional space. For recent reviews of the extensive literature on geometric constraint solving see Hoffmann et al. (1998); Kramer (1992); Fudos (1995).

2.1.2 The Main Reason to Decompose Constraint Systems

Currently there is a lack of effective spatial variational constraint solvers and assembly constraint solvers that scale to large problem sizes and can be used interactively by the designer as *conceptual* tools throughout the design process. Almost all current CAD/CAM systems primarily use a non-variational, history-based 3 dimensional constraint mechanism. This basic inadequacy in spatial constraint solving seriously hinders progress in the development of intelligent and agile CAD/CAM/CAE systems.

One governing issue is efficiency: computing the solution of the nonlinear algebraic system that arises from geometric constraints is computationally challenging, and except for very simple geometric constraint systems, this problem is not tractable in practice without further machinery. The so-called constraint propagation based solvers (e.g. Gao and Chou (1998a); Klein (1998)) generally suffer from a drawback that cannot be easily overcome: they find it difficult to decompose cyclically constrained systems, an essential feature of variational problems. Direct approaches to algebraically processing the *entire* system include the following:

1) standard methods for polynomial ideal membership and locating solutions in algebraically closed fields, for example using Gröbner bases or the Wu-Ritt method;

2) numerous algorithms and implementations for solving over the reals based on the methods of, for example, Canny (1993); Renegar (1992); Collins (1975); Lazard (1981) and

3) algorithms for decomposing and solving sparse polynomial systems based on Canny and Emiris (1993); Sturmfels (1993); Khovanskii (1978); Sridhar et al. (1993, 1996). These direct algebraic solvers deal with general systems of polynomial equations: that is, they do not exploit geometric domain knowledge; partly as a result, they have at least exponential time complexity and they are slow in practice as well. In addition, they do not take into account design considerations and as a result, cannot assist in the conceptual design process. These drawbacks apply to direct numerical solvers as well, including those that use homotopy continuation methods, see, for example, Durand (1998). The problem would be further compounded if we allowed constraints that are natural in the design process, but which must be expressed as inequalities, such as "point P is to the left of the oriented line L in the plane." Such additions would necessitate using cylindrical algebraic decomposition based techniques (Collins, 1975), such as Grigor'ev and Vorobjov (1988); Lazard (1991); Wang (1993) which have a theoretical worst-case complexity of $O(2^{n^2})$, where n is the algebraic size of the problem; or alternatively require the use of nonlinear optimization techniques, all of which are slow enough in practice that they do not represent a viable option for large problem sizes.

With regard to efficiency, the following rule of thumb has therefore emerged from years of experimentation with geometric, spatial constraint solvers in engineering design and assembly: the use of direct algebraic/numeric solvers for solving large subsystems renders a geometric constraint solver practically useless: (see Durand (1998) for a natural example of a geometric constraint system with 6 primitive geometric objects and 15 constraints, which has repeatedly defied attempts at tractable solution). The overwhelming cost in geometric constraint solving is directly proportional to the size of the largest subsystem that is solved using a direct algebraic/numeric solver. This size dictates the practical utility of

the overall constraint solver, since the time complexity of the constraint solver is at least exponential in the size of the largest such subsystem.

2.1.3 Decomposition Recombination (DR) Plans

Therefore, the constraint solver should use geometric domain knowledge to develop a plan for decomposing the constraint system into small subsystems, whose solutions can be recombined by solving other small subsystems. The primary aim of this decomposition plan is to restrict the use of direct algebraic/numeric solvers to subsystems that are as small as possible. Hence the *optimal* or most efficient decomposition plan would minimize the size of the largest such subsystem. Any geometric constraint solver should first solve the problem of efficiently finding a close-to-optimal *decomposition-recombination (DR) plan*, because that dictates the usability of the solver. Finding a DR-plan can be done as a pre-processing step by the constraint solver: a robust DR-plan would remain unchanged even as minor changes to numerical parameters or other such on-line perturbations to the constraint system are made during the design process.

In addition to optimality (efficiency), other equally important (and sometimes competing) issues arise from the fact that a DR-plan is a key conceptual component of the CAD model and should *aid* the the overall design or assembly process. These issues will be discussed under "Desirable characteristics" later in this section.

A clean and precise formulation of the DR-planning problem is therefore a fundamental necessity. To our knowledge, despite its longstanding presence, the DR-problem has not yet been clearly isolated or precisely formulated, although there have been many prior, specialized DR-planners that utilize geometric domain knowledge (Bouma et al., 1995; Owen, 1991, 1993; Hoffmann and Vermeer, 1994, 1995; Latham and Middleditch, 1996; Fudos and Hoffmann, 1996b, 1997; Crippen and Havel, 1988; Havel, 1991; Hsu, 1996; Ait-Aoudia et al., 1993; Pabon, 1993; Kramer, 1992; Serrano and Gossard, 1986; Serrano, 1990). See Hoffmann et al.

(1998) for an exposition of two primary classes of existing methods of decomposing geometric constraint systems; representative algorithms from these two classes are extensively analyzed in Section 2.1.8.

In the next two subsections we informally describe both the basic requirements of a DR-plan(ner) that dictate its overall structure, as well as desirable characteristics of the DR-plan(ner) that improve efficiency and assist in the design process.

2.1.4 Basic Requirements of a DR-plan

Recall that a DR-plan specifies a plan for decomposing a constraint system into small subsystems and recombining solutions of these subsystems later. Therefore the first requirement of a DR-plan is that the solutions of the small subsystems in the decomposition can be recombined into a solution of the entire system. In other words, it should be possible to substitute the (set of) solution(s) of each subsystem into the entire system in a natural manner, resulting in a simpler system. Secondly, we would like these intermediate subsystems that are solved during the decomposition and recombination to be geometrically meaningful.

Together, these two requirements on the intermediate subsystems translate to a requirement that the subsystems be geometrically rigid. A *rigid or solvable* subsystem of the constraint system is one for which the set of real-zeroes of the corresponding algebraic equations is discrete (that is, the corresponding real-algebraic variety is zero dimensional), *after* the local coordinate system is fixed arbitrarily, that is, after an appropriate number of degrees of freedom D, are fixed arbitrarily. The constant D is usually the number of (translational and rotational) degrees of freedom available to any rigid object in the given geometry (3 in 2 dimensions, 6 in 3 dimensions, and so on.) and in some cases, D depends on other symmetries of the subsystem. An *underconstrained* system is not solvable, that is, its set of real zeroes is not discrete (non-zero-dimensional). A

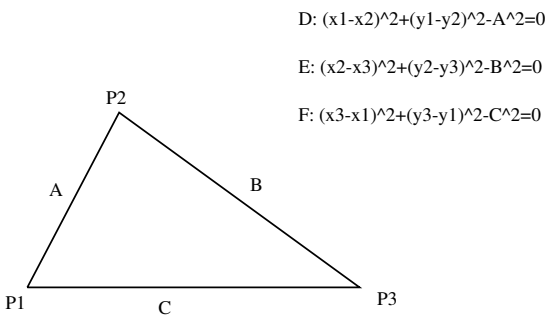


Figure 2–1: A solvable system of equations

wellconstrained system is a solvable system where removal of any constraint results in an underconstrained system. An *overconstrained* system is a solvable system in which there is a constraint whose removal still leaves the system solvable. Solvable systems of equations are therefore wellconstrained or overconstrained. I.e, the constraints force a finite number of isolated real solutions, so one solution cannot be obtained by an infinitesimal perturbation of another. For example, Figure 2–1 is a solvable subsystem of three points and three distances between pairs of points. A *consistently* overconstrained system is one which has at least one real zero.

Note. It is important to distinguish "solvable" from "has a real solution." Although (inconsistently) overconstrained (or even certain wellconstrained) systems may have no real solutions at all, by our definition, since their set of real zeroes is discrete, they would still be considered "solvable." In general, whenever a subsystem of a constraint system is detected that has no real solutions, then the solution process would have to immediately halt and inform the designer of this fact. ♢

Informally, a geometric constraint solver which solves a large constraint system E by using a DR-planner – to guide a direct algebraic/numeric solver capable of solving small subsystems – proceeds by repeatedly applying the following three steps at each iteration i.

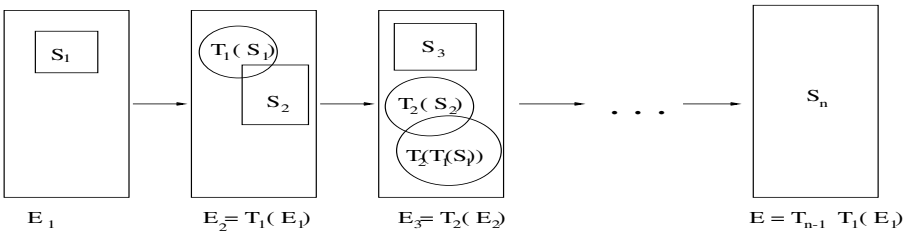


Figure 2–2: Step 1 - rectangles, Step 3 - ovals

1. Find a small solvable subsystem S_i of the (current) entire system E_i (at the first iteration, this is simply the given constraint system E, i.e, $E_1 = E$). This step is indicated by a rectangle in Figure 2–2. The subsystem S_i could be also chosen by the designer.

2. Solve S_i using the direct algebraic/numeric solver.

3. Using the output of the solver, and perhaps using the designer's help to eliminate some solutions, replace S_i by an *abstraction or simplification* $T_i(S_i)$ thereby replacing the entire system E_i by a simplification $T_i(E_i) = E_{i+1}$. This step is indicated by an oval in Figure 2–2. Some informal requirements on the simplifiers T_i are the following: we would like E_i to be (real algebraically) *inferable* from E_{i+1}; i.e, we would like any real solution of E_{i+1} to be a solution of E_i as well. A constraint solver that fits the above structural description, – which we shall refer to as $\mathcal{S}$ in future discussions – is called a *decomposition-recombination (DR) solver*. (Formal definition imposes further requirements on simplifier T_i – see the next section). This solver terminates when the small, solvable subsystem S_i found in Step 1 is the entire polynomial system E_i. An optimal DR-plan will minimize the size of the largest S_i. If the whole system is underconstrained, the solver should still solve its *maximal* solvable subsystems.

For the purpose of *planning* a solution sequence apriori, we would like to execute Steps 1 and 3 alone without access to the algebraic solver, and get a DR-plan, without actually solving the subsystems. I.e, we would like the constraint solver to

look as in Figure 2–3, with the DR-planner driving the direct algebraic/numeric solver.

To generate a DR-plan apriori, one would have to locate a solvable subsystem S_i, and without actually solving it, find a suitable abstraction or simplification of it that is substituted into the larger system E_i to obtain an overall simpler system E_{i+1} in Step 3. On the other hand, such a DR-plan would possess the advantage of being robust, or generically independent of particular numerical values attached to the constraints, and of the solution to the S_i, and usually only depends on the degrees of freedom of the relevant geometric objects and geometric constraints.

2.1.5 Desirable Characteristics of DR-planners for CAD/CAM

We enumerate and informally describe a set $\mathcal{C}$ of natural characteristics desirable for a DR-planner. We begin with four criteria that directly follow from the overall structural description of a typical DR-planner $\mathcal{S}$ in the previous subsection.

(i) The DR-planner should be general, i.e, it should not be artificially restricted to a narrow class of decomposition plans; it should output a DR-plan if there is one; and it should be able to decompose underconstrained systems as well. Furthermore, if a DR-plan exists, the planner should run to completion irrespective of how and in what order the solvable subsystems S_i are chosen for the plan (a Church-Rosser property).

(ii) The planner should potentially output a close-to-optimal DR-plan (i.e, where the size of the largest solvable subsystem S_i is close-to-minimal). This dictates efficiency of solution of the constraint system.

(iii) The DR-planner should be fast and simple; the time complexity should be low, the planner should be fast in practice, easily implementable, and compatible with existing algebraic solvers, CAD systems, constraint models and manipulators.

(iv) The planner should utilize and take advantage of geometric domain knowledge, as well as other special properties of geometric constraints arising from the relevant design or assembly application or, in some situations, from a downstream manufacturing application.

Besides critically affecting the speed of constraint solving, a good DR-plan is a key component of the constraint model which participates in the overall process of design/assembly. This is especially so, since the constraint system is a component of the CAD model which the designer directly interacts with, and moreover, a DR-plan is nothing but a *hierarchical, structural* decomposition of the geometric constraint system. Therefore, maintaining a robust DR-plan - which reflects *design intent* at every level of refinement - is invaluable in improving efficiency, flexibility and transparency in the overall design process. These properties are crucial for intelligent CAD systems to facilitate designer interaction at a conceptual, early-design phase; in fact, an effective CAD system based on spatial, variational constraints to be effective must facilitate early-design interaction. This motivates adding the following desirable characteristics to the set $\mathcal{C}$.

(v) The DR-plan should be consistent with the design intent: in particular, the designer often has a multi-layered or hierarchical conceptual, design decomposition in mind, reflecting features or conglomerates of features in the case of product design and parts or subassemblies in the case of assembly. See e.g, Klein (1996); Mantyla et al. (1989); Middleditch and Reade (1997). The designer would typically wish to further decompose the components of her design decomposition, as well as treat these components (recursively) as units that can be independently manipulated. For example, the geometric objects within a component are manipulated with respect to a local coordinate system, and a component as a whole unit is manipulated with respect to a global (next level) coordinate system. The DR-plan should

therefore be a consistent extension and/or refinement of this conceptual design decomposition.

(vi) While the DR-plan is used to guide the algebraic solver, it should remain unaffected or adapt easily to the designer's intervention in the solution process, which is valuable for pruning combinatorial explosion: the designer can throw out meaningless or undesirable solutions of subsystems at an early stage. Such designer interference is also crucial for avoiding the use of inequality constraints: for example, instead of adding on a constraint that point P is to the left of line L, the designer can simply throw out any partial solutions that put P to the right of line L.

(vii) The DR-plan for solving a geometric constraint system should remain meaningful in the presence of other than geometric constraints, such as equational constraints or parametric constraints expressing design intent.

Finally, the CAD system and the CAD model do not stand alone. In standard client-server based architectures (see e.g, Bronsvoort and Jansen (1994); Kraker et al. (1997); Hoffmann and Joan-Arinyo (1998)), the CAD model is just one client's view of the product master model (Newell and Evans, 1976; Semenkov, 1976), with which it has to be continually coordinated and made consistent. The master model in turn coordinates with other downstream production client systems which maintain other consistent views. These clients include geometric dimensioning and tolerancing systems (GD& T), and manufacturing process planners (MPP) for automatically controlled machining or assembly. Each client view contains potentially proprietary information that must be kept secure from the master model. Figure 2–3 illustrates this architecture and those parts that we deal with directly.

Each client view contains its own internal view of the constraint model and therefore coordination and consistency checks between the various views

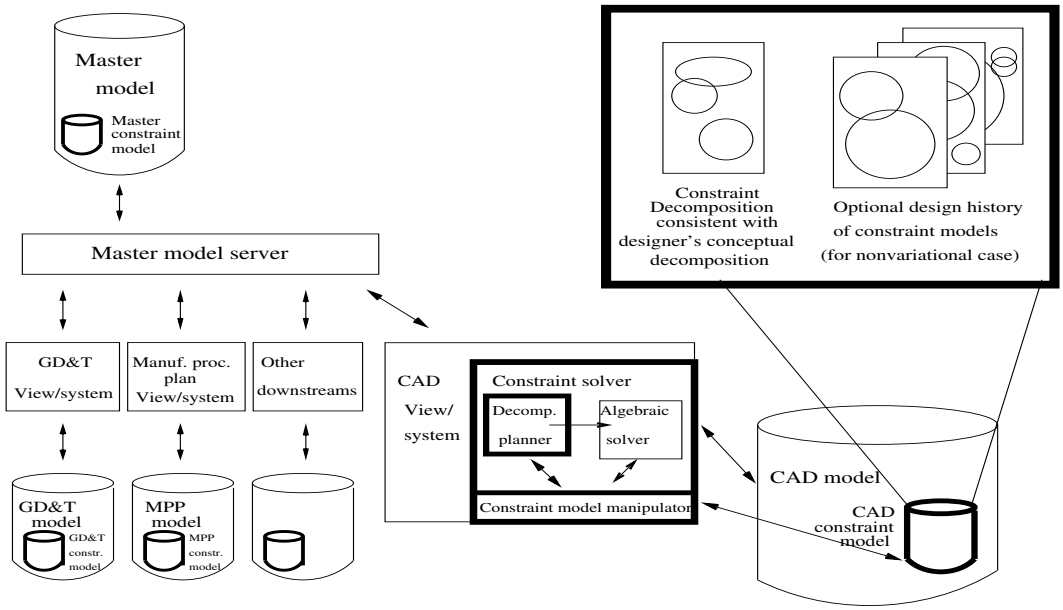


Figure 2–3: CAD/CAM/CAE master model architecture

crucially involve, and viceversa may affect the DR-plan. This leads to the following additions to the set $\mathcal{C}$ of desirable characteristics for DR-planners.

(viii) The DR-plan should be as robust as possible (see e.g. Fang (1992)) to on-line changes made to the constraint system, and the DR-planner should be able to quickly alter the DR-plan in response to such changes. In particular, the DR-plan should ideally not change *at all* with numerical perturbations (within large ranges) to the constraints, thereby permitting the DR-plan to be computed as a pre-processing step. Addition, deletion, and modification of constraints and geometric objects to the constraint system occurs in numerous circumstances during the design process. For example:

(a) In the process of solving the system using the DR-plan, a subsystem S_i deemed solvable by a degree of freedom analysis may be found to have no real solutions, or may, in a degenerate case, turn out to be underconstrained or have infinitely many solutions, preventing a continuation of the solution process $\mathcal{S}$;

(b) In history-based CAD systems, the specification of the product takes the form of a progressive sequence of changes being made to successive partial specifications and the associated partial constraint systems;

(c) Inferred or understood constraints are often added to an incomplete constraint specification at the discretion of the CAD system;

(d) In underconstrained situations, as might occur in designing flexible or moving parts and assemblies, the mapping of the configuration space of the resulting mechanism would require the repeated addition of constraints at the discretion of the constraint solver – the solutions to the resulting wellconstrained systems would represent various configurations of the mechanism;

(e) For a variational spatial constraint solver to be effective or even usable, it would have to rely on extensive interactive constraint changes being made by the designer, sometimes in the course of solving; and

(f) Finally, when one of the various clients in Figure 2–3 makes a change to its own view of the constraint model, this will result in consistency updates to the master constraint model which will, in turn, result in updates to the other views of the constraint model.

(ix) The DR-plan should isolate overconstrained subsystems which arise in assembly problems; furthermore, with multiple (possibly proprietary) views of the constraint model being kept by various clients as in 2–3, constraint reconciliation often takes place and the DR-plan should facilitate this process, and viceversa, should be robust against this process. For a precise description of the constraint reconciliation problem, see Hoffmann and Joan-Arinyo (1998). The problem requires isolation of overconstrained subsystems and is compounded in the case of non-variational, history-based design.

2.1.6 Formal Definition of DR-solvers using Polynomial Systems

We will formally state the decomposition-recombination (DR) problem for polynomial systems arising from geometric constraints. This requires formalizing the notion of a decomposition-recombination solution sequence, and of decomposition-recombination solvers that fit the description $\mathcal{S}$ given in the previous section. In addition, we define the performance measures - that capture some of the desirable properties $\mathcal{C}$ given in the last section - for comparing such sequences and solvers.

Note. As was mentioned previously, for the sake of gradual exposition, we will first assume that the constraint system is being solved even as the decomposition is being generated – i.e, the DR-solver exactly fits the structural description $\mathcal{S}$ given in the previous section. Furthermore, we will define the DR-solver in the general context of polynomial equations that arise from the geometric constraints.
In Section 2.1.7 we will shift our attention to the DR-planning problem. The DR-planner generates a decomposition plan apriori, which then drives the direct algebraic/numeric solver – together, they form a DR-solver. The DR-planner, however, will be defined in the context of constraint graphs that incorporate geometric degrees of freedom. The DR-planner and its performance measures will be analogous to the *italics* terms defined here.

In order to formally define DR-solvers and their performance measures, we need to specify the model of real-algebraic computation used by the algebraic/numeric solver in Figure 2–3. However, the issue of the model or how real numbers are represented is entirely outside the focus of this manuscript for the following reason: our DR-solvers and performance measures are robust in that they are conceptually independent of the algebraic/numeric solver being used or how real numbers are represented. The definition of DR-solvers and performance measures adapts straightforwardly to other natural models of computation. In other words,

our conceptual definition of DR-solvers and performance measures are such that if DR-solver A performs better than DR-solver B with respect to one of our performance measures, then it continues to do irrespective of the (natural) model of computation being used by the algebraic/numeric solver.

Further, as pointed out in an earlier note, we only emphasize the boldface components of Figure 2–3, specifically the DR-planner, whose operation defined in Section 2.1.7 will be seen to be purely combinatorial. In particular, no emphasis is placed on the non-boldface components, which includes the algebraic/numeric solver, and the model of computation it uses. ♢

Having noted the above, the model of computation we assume - for the the sake of completeness and formality of definitions in this manuscript - is the Blum-Shub-Smale model of real algebraic computation Blum et al. (1989). Briefly, in this model, real numbers are assumed to be representable as such as entities, without any recourse to rational approximations and finite precision or interval arithmetic. Real arithmetic operations such as multiplication and addition and division can be performed in constant time, and finding unambiguous representation of each real zero of a univariate polynomial p can also be achieved in time polynomial in the degree of p. Furthermore, these unambiguous representations of real zeroes of univariate polynomials are treated as entities just like any other real number, and can for instance, be used as coefficients of other polynomials.

A *system of equations* E is a pair (P, X) where P is a set of polynomial equations with real coefficients, and X is a set of formal indeterminates. The *union of two systems* $E_1 = (P_1, X_1)$ and $E_2 = (P_2, X_2)$ is the system $(P_1 \cup P_2, X_1 \cup X_2)$. An *intersection* $E_1 \cap E_2$ is the system $(P_1 \cap P_2, X_1 \cap X_2)$.

Within the geometric context, a solved subsystem is a rigid or solvable system where all the variables have already been "solved for," i.e, they have typically been expressed *explicitly* as polynomials of D free variables, where D represents the

number of degrees of freedom of the rigid body (within its *local* coordinate system) in the prevailing geometry. In some cases, for example when the rigid body is a line in 3 dimensions, the number of free variables, including redundant ones, may be greater than D.

Let $E = (P, V)$ with $V = \{y_1, \ldots, y_D\} \cup X$ be such that every equation in P is of type $x_j = F_j(y_1, ..., y_D)$, for all $x_j \in X$ and F_j is a rational function (of the form Q_1/Q_2 where Q_1 and Q_2 are, as usual, polynomials with real coefficients) that can be evaluated in time polynomial in $(|V|, |P|)$. Such a system of equations, E, is called a *solved system*. The variables $y_1, ..., y_D \in V$ of the solved system are *free variables* of E, and the variables $x_1, ..., x_k \in V$ are *explicitly fixed variables*. Note that all the variables in a solved system are fixed or free, whereas a general solvable system may have free, explicitly fixed and other, *implicitly fixed* variables.

A typical DR-solver - that follows the overall structural description $\mathcal{S}$ of Section 2.1.4 - obtains a sequence $E_1, \ldots, E_m$, consisting of successively simpler solvable systems. These are general solvable systems and have successively fewer implicitly fixed variables; $E = E_1$; and E_m is a solved system. New variables y_i may be introduced at intermediate stages which represent free variables *within* subsystems S_i that are solved with respect to these variables. These solved subsystems represent various rigid bodies located and fixed with respect to their local coordinate systems. However, these local coordinate systems are still constrained with respect to each other, and hence in fact only D of the newly introduced variables are, in effect, free and the remainder are implicitly fixed. Some of these newly introduced variables may be removed at later stages. Eventually, in the solved system E_m, all the original variables and those newly introduced variables that might remain become explicitly fixed with respect to D free variables as well. The set of real solutions to E_{i+1} should also be a subset of solutions to E_i, to ensure that the final solutions to E_m actually represent solutions to the original system E.

We formally define real algebraic equivalence and real algebraic inference of two geometric constraint systems as follows. Given two systems $E_1 = (P_1, X \cup Y_1)$ and $E_2 = (P_2, X \cup Y_2)$ - where the set X represent the original variables that are currently implicitly or explicitly fixed, and the sets Y_i represent the newly introduced variables (that are "free" *within* the solved subsystems S_i), we say that the system $E_1 = (P_1, X \cup Y_1)$ is *real algebraically inferable* (in short, inferable) from the system $E_2 = (P_2, X \cup Y_2)$, if for any real solution $Y_2 = A_2; X = B$ that satisfies the equations in P_2, there is a corresponding assignment A_1 of real values to Y_1 such that $Y_1 = A_1; X = B$ satisfies P_1. Two systems E_1 and E_2 are *real algebraically equivalent* if E_1 is real algebraically inferable from E_2 and E_2 is real algebraically inferable from E_1.

Now we are ready to define the notion of a DR-solution sequence. Let E be a system of equations.

A *DR-solution sequence* of E is a sequence of systems of equations $E_1, ..., E_m$ such that $E = E_1$, E_m is a solved system (so E_m has a real solution), every E_i is solvable, each E_i is inferable from E_{i+1}. Any solvable system E which has a real solution in fact has a DR-solution sequence. A trivial DR-solution sequence E, E^* - where E^* is a solved system equivalent to E - will do. (Note that by E_i we denote abstract algebraic systems, rather than their computer representations that could only have rational coefficients and therefore may only have approximate DR-solution sequences).

The *DR-problem* is the problem of finding a DR-solution sequence of a given constraint system. A *DR-solver* is a constraint solver or algorithm that solves the DR-problem. A DR-solver is *general* if it always outputs a DR-solution sequence when given a solvable system as input. Aside from being general, we would also like a DR-solver to have the *Church-Rosser property*, i.e, the DR-solver should terminate irrespective of the order in which the solvable subsystems S_i are chosen.

In other words, at each step, a solvable subsystem S_i can be chosen greedily to satisfy the (easily checkable) requirements of the algorithm. This prevents exhaustive search.

Performance measures. Next, we formally define a set of *performance measures* for the DR-solution sequences and DR-solvers. These performance measures are designed to capture the characteristics $\mathcal{C}$ of constraint solvers given in Section 2.1.5, that are desirable for engineering design and assembly applications. (Note that many of these measures are Boolean, i.e, either a certain desirable condition is met or not.) In particular, we would like a DR-solution sequence $Q = E_1, \ldots, E_m$ of a solvable system E to have several properties. To describe these properties we formalize a simplifying map from each system E_i to its successor E_{i+1}. In fact, for generality, we choose these maps T_i - called *subsystem simplifiers* - to map the set of subsystems of E_i onto the set of subsystems of E_{i+1}.

First, in order to reflect the Church-Rosser property in (i), and points (iv) and (vi) of $\mathcal{C}$, we would like these subsystem simplifiers T_i to be natural and well-behaved, i.e, to obey the following simple and non-restrictive rules.

(1) If A is a subsystem of B then $T_i(A)$ is a subsystem of $T_i(B)$

(2) $T_i(A) \cup T_i(B) = T_i(A \cup B)$

(3) $T_i(A) \cap T_i(B) = T_i(A \cap B)$

Second, in the description of the typical DR-solver $\mathcal{S}$ given in Section 2.1.4, each system E_{i+1} in the DR-solution sequence is typically to be obtained from E_i by replacing a solvable subsystem S_i in E_i (located during Step 1 of $\mathcal{S}$), by a simpler subsystem (during Steps 2 and 3). For a manipulable DR-solution sequence (again satisfying the points (i), (iv), (v) and (vi) of $\mathcal{C}$), we would like E_{i+1} to look exactly like E_i outside of S_i. This leads to another set of properties desirable for the subsystem simplifiers T_i.

(4) Each $E_i = S_i \cup R_i \cup U_i$ $1 \leq i \leq m$ where S_i is solvable, R_i is a maximal subsystem such that S_i and R_i do not share any variables, S_i, R_i, U_i do not share any equations, and all variables of U_i are either variables of S_i or R_i. For any $A \subseteq R_i$, $T_i(A) = A$.

Thirdly, in order to address the points (iv)-(vi), and (i) of $\mathcal{C}$ simultaneously, i.e, permitting generality of the subsystem S_i that is replaced by a simpler system during the i^{th} step, while at the same time making it convenient for the designer to geometrically follow and manipulate the decomposition, we would like the subsystem simplifiers to satisfy the following property.

(5) For each i, all the pre-images of S_i, $T_j^{-1}...T_{i-1}^{-1}(S_i), 1 \leq j \leq i-1$ are algebraically inferable from S_i, and furthermore, they are solvable or rigid subsystems for the given geometry (recall that the definition of "solvable" depends on the geometry). It follows from (2) and (3) above that the *inverse* $T_i^{-1}(A) = \bigcup B$ where the union is taken over all $B \subseteq E_i$, such that $T_i(B) \subseteq A$.

The above property says that while the subsystem simplifiers enjoy a high degree of generality and are completely free to map solvable subsystems into solvable systems, they should never map (convert) subsystems that are originally *not* solvable into one of the chosen, solvable systems S_i, at any stage i. In other words, in the act of simplifying, the subsystem simplifiers should not *create* one of the solvable subsystems S_i out of subsystems that were originally not solvable. A DR-solution sequence that satisfies the above properties is called *valid*. Thus a valid DR-solution sequence for a geometric constraint system E is specified as a sequence of $E_1, \ldots, E_m$ such that $E_1 = E$, E_m is a solved system, every E_i is solvable and inferable from E_{i+1}, every $E_i = S_i \cup R_i \cup U_i$ as described above.

The motivation given for each of the properties above makes it clear that valid DR-solution sequences encompass highly general but geometrically meaningful solutions

of the original constraint system E. Next we turn to point (ii) of $\mathcal{C}$, i.e, optimality, which also competes with generality (point (i)). For optimality, we would like to minimize the size of the largest solvable subsystem S_i in the DR-solution sequence. Formally, the *size* of the DR-sequence Q is equal to $\max_{1\leq i\leq m} |S_i|$, where the size $|S_i|$ is equal to the total number of it variables less the number of its explicitly fixed variables. The *optimal DR-size* of the algebraic system E is a minimum size of Q, where the minimum is taken over all possible DR-solution sequences Q of E. An *optimal DR-solution sequence* Q of E is a solution sequence such that the size of Q is equal to the optimal DR-size of E. The *approximation factor* of DR-solution sequence Q of the system E is defined as the ratio of the optimal DR-size of E to the size of Q.

As a general rule for optimality, it is clear that the larger the choice for solvable subsystems S_i of E_i available at any stage i, the more likely that one can find a *small* solvable S_i in E_i. In other words, we would like to make sure that the subsystem simplifier does *not destroy* solvability of too many *sub*systems starting from the original system E. Note that while the definition of DR-solution sequence makes sure that the entire system E_m is solvable if E_1 is, it does not require the same for subsystems of E_i. (In fact, even if all the E_i in a DR-solution-sequence are algebraically equivalent, this would still not imply that solvability is preserved for the subsystems). In addition, while the definition of a DR-solution sequence ensures that E_{i+1} has a real solution if E_i has one, it does not ensure the same for subsystems of E_i. The next two definitions capture properties of subsystem simplifiers that preserve subsystem solvability (resp. solutions) to varying degrees, thereby helping the optimality of the DR-solver, i.e, (ii) of $\mathcal{C}$. The DR-solution sequence $Q = E_1, ..., E_m$ is *solvability preserving* if and only if for all $A \subset E_i$, A is solvable (resp. has a real solution) and $(A \cap S_i = \emptyset$ or $A \subset S_i) \Longleftrightarrow T_i(A)$ is solvable (resp. has a real solution). A DR-sequence $Q = E_1, ..., E_m$ is *strictly solvability*

preserving if and only if for all $A \subset E_i$, A is solvable (resp. has a real solution) $\iff$ $T_i(A)$ is solvable (resp. has a real solution). Requiring such a solvability preserving simplifier places a weak restriction on the class of valid DR-solution sequences, but on the other hand, this restriction cannot eliminate all optimal DR-solution sequences. Furthermore, solvability preservation helps to ensure the Church-Rosser property in (i) of $\mathcal{C}$.

Note. To be more precise, "solvability preservation" should be termed "solvability and solution-existence preservation," but we choose the shorter phrase. $\diamondsuit$

In fact, for DR-solution sequences to be optimal, we would prefer that for all $i \geq 1$, S_i does not contain any strictly smaller subsystem, say B that is solvable, and has not been found (during Step 1 of the description $\mathcal{S}$ in Section 2.1.4) and simplified/replaced at an earlier stage $j < i$. The DR-sequence $Q = E_1, ..., E_m$ is *complete* if and only if for every nontrivial solvable $B \subset S_i$, $B = T_{i-1}T_{i-2}...T_j(S_j)$ for some $j \leq i-1$. While the completeness requirement restricts the class of valid DR-solution sequences, it *only eliminates* sequences that either have size greater than optimal or the same size as some optimal sequence that does simplify B. In addition to affecting optimality, i.e, (ii) of $\mathcal{C}$, completeness also strongly reflects (ix): completeness prevents a DR-solver from overlooking an overconstrained subsystem inside a wellconstrained subsystem, which is also useful for constraint reconciliation (see Hoffmann and Joan-Arinyo (1998)).

Performance measures. So far, we have discussed performance measures that measure the desirability of DR-solution sequences. Next we formally define directly analogous performance measures for DR-solvers which generate DR-sequences. A DR-solver A is said to be *valid, solvability preserving, strictly solvability preserving, complete* if and only if for every input constraint system E, every DR-solution sequence produced by A is valid, solvability preserving, strictly solvability preserving or complete respectively.

Note. Purely as a tool to help analysis and exposition, often we assume that DR-solvers for producing DR-sequences are randomized or nondeterministic in a natural way, i.e those steps in the algorithm where *arbitrary* choices of equations or variables are made (for example, to be the lowest numbered equation or variable) are now taken to be randomized or nondeterministic choices. ♢

The next definition formalizes performance measures related to the characteristic (ii) of $\mathcal{C}$ that differentiates randomized DR-solvers for which *some* random choice leads to an optimal DR-solution sequence, from inferior DR-solvers where *no* choice would lead to an optimal DR-solution sequence. The *worst-choice approximation factor* of a DR-solver A on input system E is the minimum of the approximation factors of all DR-solution sequences Q of E obtained by the algorithm A over all possible random choices. The *best-choice approximation factor* of the algorithm A on input E is maximum of the approximation factors of all the DR-solution sequences Q of E obtained by the algorithm A over all possible random choices.

2.1.7 Formal Definition of a DR-planner via Constraint Graphs

The DR-solution sequence defined in the previous section embeds a DR-plan that is intertwined with the actual solution of the system. Therefore, a DR-solver that simply outputs a DR-solution sequence (even one that is strictly solvability preserving, complete, and so on..), may not be modular as shown in Figure 2–3.

How to construct a DR-solver that first generates a DR-plan before using it to drive the general algebraic/numeric solver? First notice that repeatedly applying Steps 1 and 3 of the DR-solver in the description $\mathcal{S}$ in Section 2.1.4 could potentially generate a DR-plan without actually solving any subsystem (and without applying Step 2), provided that the following can be done:

(a) solvability of a subsystem S_i of the system E_i in Step 1 can be determined generically, without actually solving it; and

(b) a solvable subsystem S_i can be replaced in Step 3 without actually solving S_i - i.e, by a hypothetical solved subsystem - to give the new system E_{i+1}. For this we need a consistent and adequate *abstraction* of the systems E_i, and of their subsystems, as well as a way to abstract a hypothetical solved subsystem. In other words, we need to adapt the subsystem simplifier maps described in the last section, so that the iteration can proceed with Step 1 again.

By thus modifying the description $\mathcal{S}$ of the DR-solver, we obtain a DR-planner which can be employed as a preprocessing step to output a DR-plan instead of a DR-solution sequence. This DR-plan can *thereafter* be used to direct a series of applications of Step 2, leading to a complete DR-solution sequence. This modularity helps, for example, towards maneuverability by the designer (point (vi)), and towards compatibility of the DR-solver with existing solvers (point (v)).

In order to formally define such a DR-plan, we follow a common practice and view the constraint system as the constraint hypergraph: this abstraction permits us to build the DR-planner on the foundation of generalized degree of freedom analysis which is known to work well in estimating generic solvability of constraint systems *without* actually solving them. (This is explained more precisely after the formal graph-theoretic definitions are in place). Hence using constraint graphs facilitates *both* the tasks (a) and (b) described above.

In other words a DR-planner that is based on a generalized degree of freedom analysis is robust in the following sense: changing the numerical values of constraints is not likely to affect the DR-plan. Thus the motivation for using constraint graphs includes all of the desirable characteristics (iii) to (vi) of the set $\mathcal{C}$ in Section 2.1.5. In addition, geometry is more visibly displayed via constraint graphs than via equations, thereby helping interaction with the designer.

Note. DR-plans and planners and their performance measures could also be defined directly in terms of the algebraic constraint systems, just as we defined

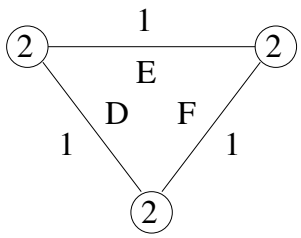


Figure 2–4: A constraint graph

DR-solvers. In this thesis, however, due to the reasons mentioned above, we define DR-plans and planners entirely within the context of constraint graphs and degree of freedom analysis. ♢

Constraint graphs and solvability. First we define the abstraction that converts geometric constraint system into a weighted graph. Recall that a geometric constraint problem consists of a set of geometric objects and a set of constraints between them.

A geometric constraint graph $G = (V, E, w)$ corresponding to a geometric constraint problem is a weighted graph with n vertices (representing geometric objects) V and m edges (representing constraints) E; $w(v)$ is the weight of vertex v and $w(e)$ is the weight of edge e, corresponding to the number of degrees of freedom available to an object represented by v and number of degrees of freedom removed by a constraint represented by e respectively. For example, Figure 2–4 is a constraint graph of a constraint problem shown in Figure 2–1. Note that the constraint graph could be a *hypergraph*, each hyperedge involving any number of vertices.

Now we introduce definitions that will help us to relate the notion of solvability of the geometric constraint system to the corresponding geometric constraint graph.

A subgraph $A \subseteq G$ that satisfies

$$\sum_{e \in A} w(e) + D \geq \sum_{v \in A} w(v) \tag{2.1}$$

is called *dense*, where D is a dimension-dependent constant, to be described below. The function $d(A) = \sum_{e \in A} w(e) - \sum_{v \in A} w(v)$ is called *density* of a graph A. The constant D is typically $\binom{d+1}{2}$ where d is the dimension. The constant D captures the degrees of freedom associated with the *cluster* of geometric objects corresponding to the dense graph. In general, we use words "subgraph" and "cluster" interchangeably. For planar contexts and Euclidean geometry, we expect $D = 3$ and for spatial contexts $D = 6$, in general. If we expect the cluster to be fixed with respect to a global coordinate system, then $D = 0$.

A dense graph with density strictly greater than $-D$ is called *overconstrained*. A graph that is dense and all of whose subgraphs (including itself) have density at most $-D$ is called *wellconstrained*. A graph G is called *welloverconstrained* if it satisfies the following: G is dense, G has at least one overconstrained subgraph, and has the property that on replacing all overconstrained subgraphs by wellconstrained subgraphs, G remains dense. A graph that is wellconstrained or welloverconstrained is said to be *solvable*. A dense graph is *minimal* if it has no proper dense subgraph. Note that all minimal dense subgraphs are solvable, but the converse is not the case. A graph that is not solvable is said to be *underconstrained*. If a dense graph is not minimal, it could in fact be an underconstrained graph: the density of the graph could be the result of embedding a subgraph of density greater than $-D$.

In order to understand how solvable constraint graphs relate to solvable constraint systems it is important to remember that a geometric constraint problem has two aspects - combinatorial and geometric. The geometric aspect deals with actual parameters of the geometric constraint problem, while the combinatorial aspect deals with only the abstractions of objects and constraints. Unfortunately at the moment it is not known how to completely separate the two aspects, except for some special cases. In this thesis we will limit ourselves to the geometric constraint problems where generally there is a correspondence between solvable constraint

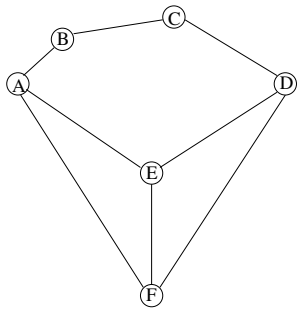


Figure 2–5: Generically unsolvable system

systems and solvable constraint graphs. In order to do that, we need to introduce and formalize the notion of generic solvability of constraint systems. Informally, a constraint system is generically (un)solvable if it is (un)solvable for most choices of coefficients of the system. More formally we use the notion of *genericity* of e.g, Cox et al. (1998). A property is said to hold *generically* for polynomials $f_1, \ldots, f_n$ if there is a nonzero polynomial P in the coefficients of the f_i such that this property holds for all $f_1, \ldots, f_n$ for which P does not vanish.

Thus the constraint system E is generically (un)solvable if there is a nonzero polynomial P in the parameters of the constraint system - such that E is (un)solvable when P does not vanish. Consider for example Figure 2–5. Here the objects are 6 points in 2d and the constraints are 8 distances between them. This system is unsolvable since the edge BC can be continuously displaced, unless the length of AD (induced by the lengths of AE, DE, DF, AF and EF) is equal to the $|AB| + |BC| + |CD|$. Since we can create an appropriate nonzero polynomial $P(AB, BC, \ldots, EF)$, such that $P() = 0$ if and only if $|AD| = |AB| + |BC| + |CD|$, this system is generically unsolvable.

While a generically solvable system always gives a solvable constraint graph, the converse is not always the case. In fact, there are solvable, even minimal dense graphs whose corresponding systems are not generically solvable, and are in fact generically not solvable (note that the position of the 'not' changes the meaning,

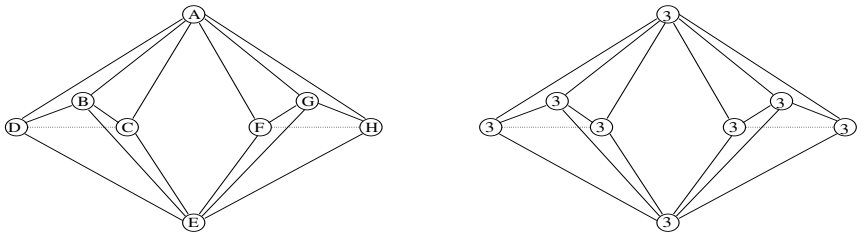


Figure 2–6: Generically unsolvable system that has a solvable constraint graph

the latter being stronger than the former). Consider for example Figure 2–6. The constraint system is shown on the left, it consists of 8 points in 3d and 18 distances between them. The corresponding constraint graph is shown on the right, weight of all the edges is 1, weight of the vertices is 3, geometry dependent constant $D = 6$. Note that this system is generically unsolvable since rigid bodies $ABCDE$ and $AFGHE$ can rotate about the axis passing through AE. This example can also be transformed into a constraint problem in 2d, where the objects are circles and the constraints are angles of intersections Saliola and Whiteley (1999).

Another example is the graph $K_{7,6}$ that in 4 dimensions represents distances between pairs of points. The constraint graph is minimal dense but it does not represent a generically solvable system.

It should be noted that in 2 dimensions, according to Laman's theorem Laman (1970) if all geometric objects are points and all constraints are distance constraints between these points then any minimal dense subgraph represents a generically solvable system. Also there exists a purely combinatorial characterization of solvable systems in 1 dimension based on connectivity of the constraint graphs. However, as examples above indicate, the generalization of Laman's theorem fails in higher dimensions even for the case of points and distances.

There is a matroid based approach to verifying whether solvability of the constraint graph implies solvability of the constraint system. It begins by checking whether a submodular function $f(E) = 2|V| - 3$, defined on sets of edges of the constraint graph creates a matroid Whiteley (1992, 1997), i.e checking whether

the created function is positive on single edges. Thereafter this approach checks whether the corresponding geometric structure is generically rigid. Some matroid based approaches to determining generic solvability are fast in practice, for example those to be discussed later under "generalized maximum matching". However, so far no match has been established (for all cases) between the created matroid and generic rigidity of the particular geometric problem.

There are several attempts at characterization of generic solvability in dimension 3 and higher for the case of points and distances Tay (1999); Graver et al. (1993). One such characterization, the so called Henneberg construction, checks whether a given constraint graph can be constructed from the initial basic graph by applying a sequence of standard replacements. A characterization due to Dress checks whether the constraint graph satisfies a certain inclusion-exclusion type rule. A characterization due to Crapo examines whether the constraint graph can be decomposed into a union of certain edge-disjoint trees. All of these characterizations though interesting and useful, are so far unproven conjectures.

Note. Due to the above discussion, we restrict ourselves to the class of constraint systems where solvability of the constraint graph in fact implies the generic solvability of the constraint system. (As pointed out earlier, the converse is always true, with no assumptions on the constraint system).

As was indicated above, this class is far from empty, it contains all constraint problems involving points and distances in 2d, problems resulting from Cauchy triangulations of the polyhedra in 3d as well as body-and-hinge structures in 3d. Moreover, it should be emphasized that while existing applications stop at finding subgraphs representing solvable constraint systems, we are interested in the entire problem of decomposition and recombination, optimizing the size of the largest subsystem to be solved, i.e we are interested in finding an optimal DR-plan. Also note that already for the class of generically solvable constraint systems and

Figure 2–7: Original geometric constraint graph G_1 and simplified graph G_2

corresponding graphs, the DR-planning problem is, in general, difficult (see later Note about NP-hardness, after the definition of the optimal DR-plan). ♢

Formal definition of DR-plans using constraint graphs. Informally, stated in terms of constraint graphs, the DR-planning problem involves finding a sequence of graphs G_i - a DR-plan - such that the original constraint graph $G = G_1$ and every G_i contains a minimal solvable subgraph S_i, which is simplified or abstracted into a simpler subgraph $T_i(S_i)$ and substituted into G_i to give an overall simpler graph $G_{i+1} = T_i(G_i)$. (While $T_i(S_i)$ should be simpler than S_i, it should also be somehow equivalent to S_i, for example by having same density value.) If the original graph G_1 is wellconstrained, then the process terminates when $G_m = S_m$. (If not, the process terminates with the decomposition of G_m into a maximal set of minimal solvable subgraphs).

Consider for example Figure 2–7 which shows one simplification step. On the left is the constraint graph G_1, the weight of all vertices is 2, weight of all edges is 1, geometry dependent constant $D = 3$. Then $S_1 = \{A, B, C\}$ is a solvable subgraph. On the right is the simplified graph $T_1(G_1) = G_2$, after subgraph S_1 is replaced by a vertex $\{V\} = T_1(S_1)$. Since density of the subgraph S_1 is -3, the weight of the vertex $\{V\}$ could be set to 3.

A sequence of simplification steps is shown in Figure 2–8. The top part depicts a geometric constraint graph G, where the weight of each edge is 1, the weight of each vertex is 2, and the dimension dependent constant D is equal to 3 (this

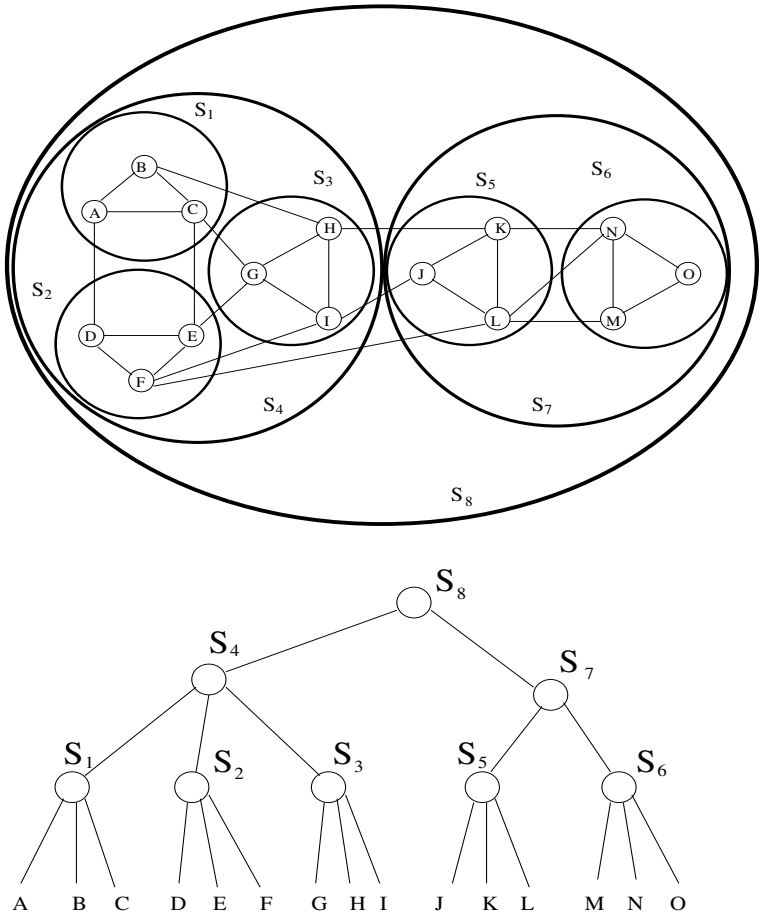


Figure 2–8: Geometric constraint graph and a DR-plan

corresponds to points and distances in 2d). One of the plans for decomposing and recombining G (and the geometric constraint system that G represents) into small solvable subgraphs (representing solvable subsystems), is to decompose G into dense subgraphs $S_1 = \{A, B, C\}, S_2 = \{D, E, F\}, S_3 = \{G, H, I\}, S_5 = \{J, K, L\}, S_6 = \{M, N, O\}$, represent their solutions appropriately in a simplified graph so that they can be recombined, one possibility is to represent them as vertices P, Q, R, S, T of weight 3 each; recursively decompose the simplified graph into $S_4 = \{P, Q, R\}, S_7 = \{S, T\}$; represent and recombine their solutions as vertices U, W of weight 3; and so on until the entire graph is represented and recombined as a single vertex.

A corresponding DR-plan is shown at the bottom part of Figure 2–8. Note that there could be more than one DR-plan for a given constraint graph. For example, another possible DR-plan for a constraint graph described above is shown in Figure 2–9.

An optimal DR-plan will minimize the size of the largest solvable subgraph S_i found during the process. I.e, it will *minimize the maximum fan-in of the vertices in the DR-tree* shown in Figure 2–8 and Figure 2–9, where by fan-in of a vertex we mean the number of immediate descendants of the vertex. With this description, it should be clear, that DR-plans obtained using the weighted, constraint graph model are generically robust with respect to the changes made to the geometric constraints; as long as the number of degrees of freedom attached to the objects and destroyed by the constraints remains the same, the same DR-plan will work for the changed constraint system as well. Thus, such DR-plans satisfy the initial robustness requirements of the characteristic (viii) of the set $\mathcal{C}$ described in Section 2.1.5.

Next we formally define a DR-plan for constraint graphs, and the various performance measures that capture desirable properties of DR-plans and DR-planners.

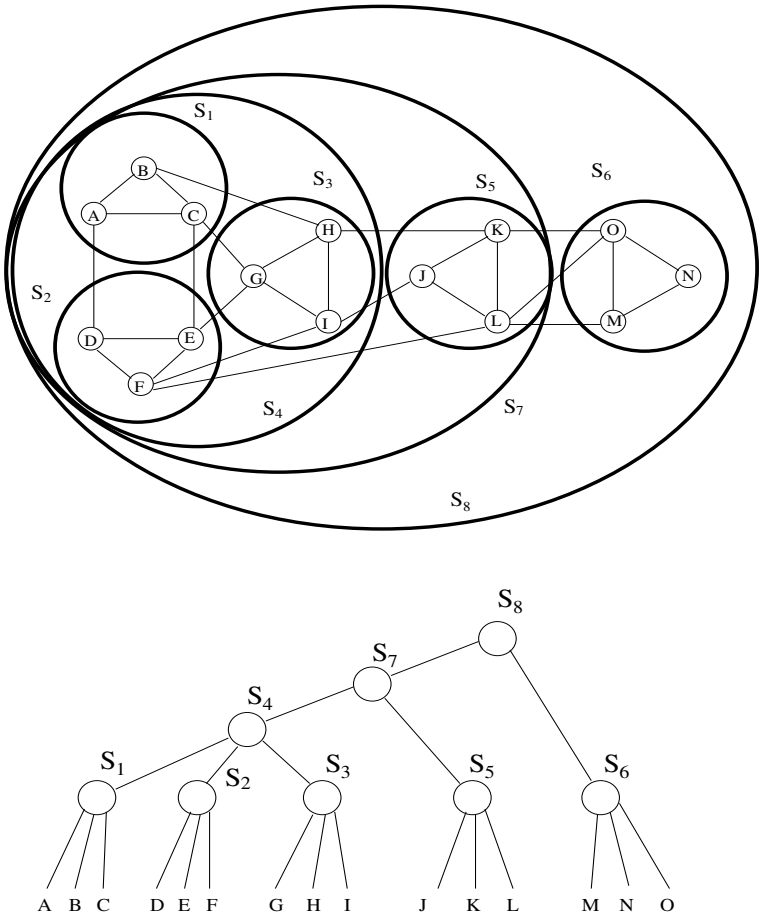


Figure 2–9: Another possible DR-plan

The development is parallel to that in Section 2.1.6, where a DR-solution sequence and various performance measures for these sequences and for DR-solvers were motivated and defined in terms of polynomial systems.

Note. Due to the strong analogy of DR-planners to DR-solvers defined in the previous section, the discussion here is more terse. The following are useful generic correspondences to keep in mind *after* reading the Note at the end of Section 2.1.7, and the paragraphs preceding it:

Solvable subgraphs $\rightleftharpoons$ solvable subsystems;

DR-plans $\rightleftharpoons$ DR-solution sequences;

DR-planner $\rightleftharpoons$ DR-solver;

subgraph simplifier $\rightleftharpoons$ subsystem simplifier. $\diamondsuit$

Let G be a solvable constraint graph. A *DR-plan* Q for G is a sequence Q of graphs $G_1, ..., G_m$ such that $G_1 = G$, G_m is a minimal solvable graph, and every G_i is solvable. An algorithm is a *general DR-planner* if it outputs a DR-plan when given a solvable constraint graph as input.

The *union (resp. intersection) of two subgraphs* A and B is the graph induced by the union (resp. intersection) of sets of vertices of A and B. All subgraphs are understood to be vertex induced.

The mapping from the graph G_i to G_{i+1} is called a *subgraph simplifier* and is denoted by T_i. This mapping should have the following properties.

(1) If A is a subgraph of B then $T_i(A)$ is a subgraph of $T_i(B)$.

(2) $T_i(A) \cup T_i(B)$ is the same as the graph $T_i(A \cup B)$.

(3) $T_i(A) \cap T_i(B)$ is the same as $T_i(A \cap B)$.

As in the case of DR-solution sequences, assume that every constraint graph G_i in the DR-plan can be written as $S_i \cup R_i \cup U_i$, where S_i is minimal solvable, R_i is a maximal subgraph such that S_i and R_i do not have common vertices, S_i, R_i, U_i do not have common edges and all vertices of U_i are either vertices of S_i of vertices of

R_i. Analogous to properties (4) and (5) of the subsystem simplifier in the previous section, we would like the subgraph simplifier to have the following additional properties.

(4) For every $A \subseteq R_i$, $T_i(A) = A$

(5) All the pre-images of S_i, i.e $T_j^{-1}T_{j+1}^{-1}...T_{i-1}^{-1}(S_i)$ for all $1 \leq j < i-1$, are solvable.

A DR-plan that satisfies the above rules is called *valid*. The *size* of the DR-plan Q of G is the maximum of the sizes of S_i. The size of an arbitrary subgraph $A \subseteq G_i$ is computed as follows.

$Size(A) = 0$

For $1 \leq j \leq i-1$

$B = T_{i-1}T_{i-2}...T_j(S_j)$

If $A \cap B \neq 0$

then $Size(A) = Size(A) + D, A = A \setminus B$

end if

end for

$Size(A) = Size(A) + \sum_{v \in A} w(v)$

In other words, the image of any of the S_j contributes D to the size of A, where D is geometry dependent constant, and the vertices of A that are not in any such image contribute their original weight. The *optimal size* of the constraint graph G is the minimum size of Q, where the minimum is taken over all possible DR-plans of G. An *optimal DR-plan* of G is the DR-plan that has size equal to the optimal size of G. The *approximation factor* of DR-plan Q of the graph G is defined as the ratio of the optimal size of G to the size of Q.

Note. The problem of finding the optimal DR-plan for a constraint graph with unbounded vertex weights is NP-hard. This follows from a result in the authors' paper Hoffmann et al. (1997) showing that the problem of finding a *minimum*

dense subgraph is NP-hard, by reducing this problem to the CLIQUE. The CLIQUE problem is extremely hard to approximate Hastad (1996), i.e, finding a clique of size within a $n^{1-\epsilon}$ factor of the size of the maximum clique cannot be achieved in time polynomial in n, for any constant ϵ (unless P=NP). However our reduction of CLIQUE to the optimal DR-planning problem is not a so-called gap-preserving reduction (or $L-$reduction); thus how well this problem could be approximated is still an open question. ♢

The definition of solvability preservation is largely analogous to the case of DR-solvers, but is additionally motivated by the following. In the case of DR-solvers, one condition on a solvability preserving simplifier is that it preserves the existence of real solutions for certain subsystems. Here, in the case of DR-plans, a natural choice is to correspondingly require that the simplifier does not map wellconstrained subgraphs or overconstrained subgraphs to underconstrained and viceversa. The DR-plan Q of G is *solvability preserving* if and only if for all $A \subset G_i$, A is solvable and $(A \cap S_i = \emptyset$ or $A \subset S_i) \Longleftrightarrow T_i(A)$ is solvable. The DR-plan Q of G is *strictly solvability preserving* if and only if for all $A \subset G_i$, A is solvable $\Leftrightarrow T_i(A)$ is solvable. The DR-plan Q of G is *complete* if and only if for all nontrivial solvable $B \subset S_i$, $B = T_{i-1}T_{i-2}...T_j(S_j)$ for some $j \leq i-1$.

Next, we formally define DR-planners and their performance measures. An algorithm is said to be a *DR-planner* if it outputs a DR-plan when given a solvable constraint graph as input. As before, we consider DR-planners to be randomized algorithms. A randomized DR-planner A is said to be *valid, solvability preserving, strictly solvability preserving, complete* if and only if for every G *every* DR-plan produced by A is valid, solvability preserving, strictly solvability preserving or complete accordingly. The *worst-choice approximation factor* of a DR-planner A on input graph G is the minimum of the approximation factors of all DR-plans Q of G obtained by the DR-planner A over all possible random choices. The *best-choice*

approximation factor of the algorithm A on input graph G is the maximum of the approximation factors of all the DR-plans Q of G obtained by the DR-planner A over all possible random choices.

In addition to the above performance measures, we define three others that reflect the Church-Rosser property, the ability to deal with underconstrained subsystems, as well as the ability to incorporate an input, design decomposition provided by the designer.

A DR-planner is said to have the *Church-Rosser* property, if the DR-planner terminates with a DR-plan irrespective of the order in which the dense subgraphs S_i are chosen.

A DR-planner A *adapts to underconstrained constraint graphs* G if every (partial) DR-plan produced by A terminates with a set of solvable subgraphs Q_i such that each solvable subgraph Q_i has no supergraph that is solvable, and moreover, no subgraph of G that is disjoint from all of the Q_i's is solvable.

A *conceptual design decomposition* P is a set of solvable subgraphs P_i, which are partially ordered with respect to the subgraph relation. A DR-planner A is said to *incorporate a design decomposition* P, if for every DR-plan Q produced by A, the corresponding sequence of solvable subgraphs S_i embeds a topological ordering of P as a subsequence - recall that a topological ordering is one that is consistent with the natural partial order given by the subgraph relation on P.

When a DR-plan incorporates a design decomposition P, the level of a cluster P_i in the partial ordering of P can now be viewed as a priority rating which specifies which component of the design decomposition has most influence over a given geometric object. In other words, a given geometric object K is first fixed/manipulated with respect to the local coordinate system of the lowest level cluster $P_i \in P$ containing K. Thereafter, the entire cluster P_i can be treated as a

unit and can be independently fixed/manipulated in the local coordinate system of the next level cluster P_j containing (the simplification of) P_i, and so on.

Finally, we summarize how the above formal performance measures capture the informally described characteristics in the set $\mathcal{C}$ given in the Section 2.1.5. The property of being a general DR-planner refers to whether the method successfully terminates with a DR-plan in the general case, and reflects characteristic (i); since we use constraint graphs which yield robust DR-plans that can be obtained efficiently, the property of being a general DR-planner also reflects (iii) and (viii); dealing with underconstrained graphs also reflects (i); incorporating input, design decompositions reflects (v); validity influences the Church-Rosser property and reflects (i) as well as (iv),(v), (vi); solvability preservation, strict solvability preservation influence the Church-Rosser property and reflect (i), (ii), (iv), (v); completeness is based on the criteria (ii) and (ix); worst and best choice approximation factors are based on (ii) and complexity directly reflects (iii).

2.1.8 Two old DR-planners

We concentrate on two primary types of prior algorithms for constructing DR-plans using constraint graphs and geometric degrees of freedom.

Note. Due to reasons discussed in Section 2.1.7, we leave out those graph rigidity based methods for distance constraints in dimensions 3 or greater such as Tay and Whiteley (1985); Hendrickson (1992) as well as methods such as Crippen and Havel (1988); Havel (1991); Hsu (1996) since they are nondeterministic or rely on symbolic computation, or they are randomized or generally exponential and based on exhaustive search. Graph rigidity and matroid based methods for more specific constraint graphs are discussed in Section 2.1.10 under the so-called Maximum Matching based algorithms. Also, for reasons discussed in the introduction, we leave out methods based on decomposing sparse polynomial systems. ♢

The first type of algorithms, which we call SR for *constraint Shape Recognition* (e.g.Bouma et al. (1995); Owen (1991, 1993); Bouma et al. (1995), Hoffmann and Vermeer (1994, 1995); Fudos and Hoffmann (1996b, 1997)), concentrates on recognizing specific solvable subgraphs of known shape, most commonly, patterns such as triangles. The second type, which we call MM for *generalized Maximum Matching* (e.g. Ait-Aoudia et al. (1993); Pabon (1993); Latham and Middleditch (1996); Kramer (1992)), is based on first isolating certain solvable subgraphs by transforming the constraint graph into a bipartite graph and finding a maximum generalized matching, followed by a connectivity analysis to obtain the DR-plan. In this section, we give a formal performance analysis of SR, and MM based algorithms – choosing a representative algorithm (typically the best performer) in each class – using the performance measures defined in the previous section.

Informally, one major drawback of the SR and MM algorithms is their inability to perform a generalized degree of freedom analysis. For example, SR would require an infinite repertoire of patterns. In the case of spatial constraints, some elementary patterns have been identified Hoffmann and Vermeer (1994, 1995). In the case of extending the scope of planar constraint solvers, adding free-form curves or conic sections requires additional patterns as well Hoffmann and Peters (1995); Fudos and Hoffmann (1996a). Similarly, a decomposition of underconstrained constraint graphs into wellconstrained components is possible for SR algorithms but only subject to the pattern limitations. In many cases, MM algorithms will output DR-plans with larger, nonminimal subgraphs S_i that may contain smaller solvable subgraphs. This affects the approximation factors adversely.

This inability to find general minimal dense subgraphs also affects their ability to deal with overconstrained subgraphs that arise in assemblies, which is in turn needed to perform constraint reconciliation Hoffmann and Joan-Arinyo (1998).

2.1.9 Constraint Shape Recognition (SR)

Consider the algorithm of Fudos and Hoffmann (1996b, 1997) which relies on a following strong assumption *(SR1)*:

all geometric objects in 2 dimensions have 2 degrees of freedom and all constraints between them are binary and destroy exactly one degree of freedom. Thus, in the corresponding constraint graph, the weight of all the edges is 1 and of all the vertices is 2. Because of this assumption, the SR algorithm ignores the degrees of freedom and relies only on the topology of the constraint graph.

Description of the algorithm. We give a terse description of the algorithm – the reader is referred to Fudos and Hoffmann (1996b, 1997) for details. Our description is meant only to put the algorithm into an appropriate DR-planner framework that is suited for the performance analysis.

The algorithm consists of two phases. During Phase One, SR uses the bottom-up iterative technique of Itai and Rodeh (1978): in the current graph G_i (where $G_1 = G$), specific solvable graphs (clusters) are found that can be represented as a union of three previously found clusters that pairwise share a common vertex. Such configurations of three clusters are called *triangles*. The vertices of G_i represent clusters and edges of G_i represent constraints between clusters (initially due to SR1, every vertex and every edge of G is a cluster. More specifically, once a triangle formed by three clusters has been found, the three vertices in G_i corresponding to these three clusters are simplified into one new vertex in the simplified graph G_{i+1}. The edges of G_{i+1} are induced by the edges of the three old vertices. This is repeated for k steps until there are no more triangles left. If there is only one cluster left in G_k, then the algorithm terminates. Otherwise G_k serves as an input to Phase Two of the algorithm.

Before we describe Phase Two, we note that a so-called *cluster graph* C_i corresponding to G_i is used as an auxiliary structure for the purpose of finding

triangles. Hence the partial DR-plan produced during Phase One of the algorithm is of the form:

$(G_1, C_1), \ldots, (G_k, C_k)$. The vertices of the cluster graph C_i correspond to

–vertices of the original graph G_1;

–cluster-vertices in the graph G_i; and

–edges in G_1 which have not been included into one of the cluster-vertices in the graphs $G_{i-1}, \ldots, G_1$.

In particular, the first cluster graph C_1 contains one vertex for every vertex and every edge of G_1. The edges of the cluster graph C_i connect the vertices of G_i that represent clusters to the original vertices from G_1 that are contained in these clusters. Due to the structure of cluster graphs, triangles in G_i are found by looking for *specific* 6-cycles in the corresponding cluster graph C_i. These 6-cycles consist of 3 cluster-vertices and 3 original vertices. The new cluster graph C_{i+1} is constructed from C_i by adding a new vertex c_i representing the newly found cluster S_i, and connecting it by edges to the original vertices from G_1 that are in the cluster S_i. We also remove the three old cluster-vertices in C_i which together formed the new cluster S_i. We note that this is the *only* way in which cluster-vertices are removed from cluster graphs. In particular, situations may arise where two clusters (that share a single vertex) are represented by the same vertex in the graph G_i, but they are represented by distinct vertices in the cluster graph C_i.

During Phase Two, SR constructs the remainder of the DR-plan $(G_k, C_k), \ldots, (G_m, C_m)$. First SR uses a global top-down technique of Hopcroft and Tarjan (1973) to divide G_k into a collection of triconnected subgraphs. These subgraphs are found by recursively splitting the original graph using separators of size at most 2. Thus the triconnected components can be viewed as the leaves of a binary tree T each of whose internal nodes corresponds to a vertex separator of size at most 2.

For every triconnected subgraph S in G_k, a new cluster vertex is created in the cluster graph C_k: similar to Phase One, this vertex replaces all vertices that represent existing clusters contained in S.

The next pair (G_{k+1}, C_{k+1}) in the DR-plan is found as in Phase One by finding a triangle in G_k, or effectively locating a 6-cycle in C_k. However, triangles in G_k have already been located in the course of constructing the tree T - these triangles were originally split by the vertex separators at the internal vertices. Thus, if the original constraint graph G is solvable, the remainder of the DR-plan $(G_{k+1}, C_{k+1}), \ldots, (G_m, C_m)$ is formed by repeated simplifications along a bottom up traversal of the tree T.

If the original graph was underconstrained, then it is still possible to construct a DR-plan of its maximal wellconstrained subgraphs, by introducing additional constraints and making the original graph solvable. I.e, in order to complete the bottom up traversal of the tree T in Phase Two, additional constraints need to be introduced, to make the whole graph solvable. For details, see Fudos and Hoffmann (1996b, 1997).

Example. Consider Figure 2–10. During Phase One, the triangles ADE, ABE, BCE and CEF will be discovered and simplified as P. During Phase Two, the remainder of the graph will be divided into triconnected subgraphs $P, PGKILN$ and $OKHMJP$, then $PGKILN$ and $OKHMJP$ are simplified and finally the union of $P, PGKILN$ and $OKHMJP$ is simplified.

Defining the Simplifier Map. The same simplifier is used throughout Phase One and Two: replace a subgraph S_i consisting of a triangle of clusters in G_i (or triconnected subgraphs during Phase Two), by one vertex in G_{i+1} representing this triangle. The subgraph S_i is found as a 6-cycle in the cluster graph C_i. The cluster graph C_{i+1} is constructed as described in Phase One.

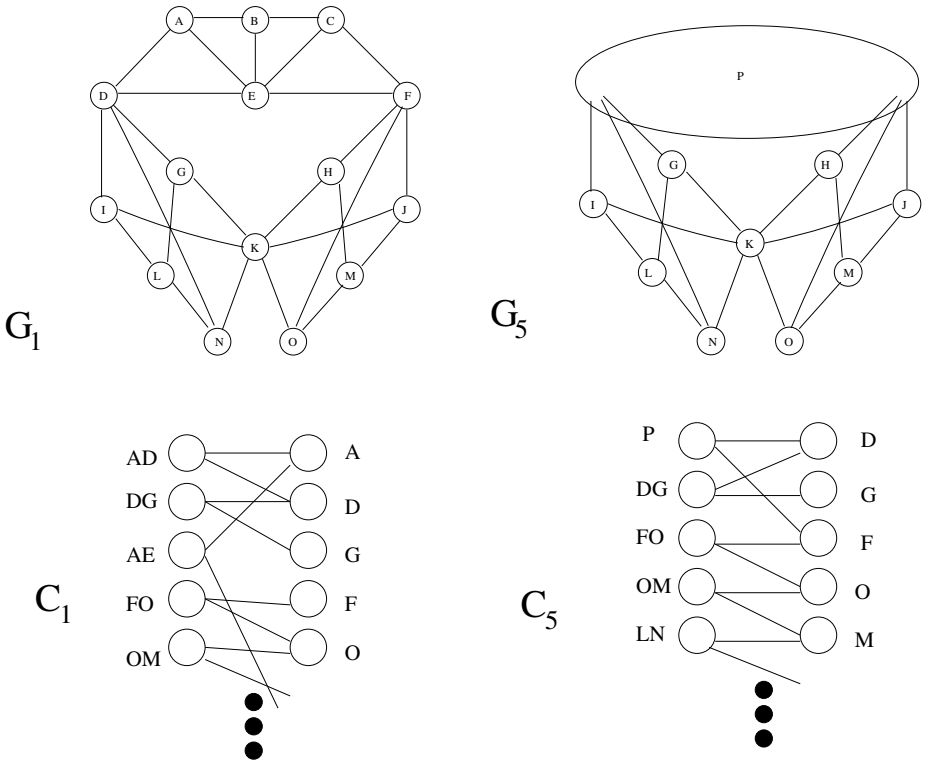


Figure 2–10: The original, cluster graph and the simplified graphs

More formally, recalling the definitions in Section 2.1.7: let G be a constraint graph; the first graph G_1 in the DR-plan is the original graph G. Let $G_i = (V, E)$ be the current graph and let S_i be a cluster found. Let $A \subseteq G_i, A = (V_A, E_A)$. Then the image of A under the subgraph simplifier T_i is $T_i(A) = A$, if the intersection of A and S_i is empty; otherwise $T_i(A) = (V_{T_i(A)}, E_{T_i(A)})$ where $V_{T_i(A)}$ is the set of all vertices of A that are not vertices of S_i plus a vertex c_i that replaces the cluster S_i. The set of edges $E_{T_i(A)}$ is a set of all edges of A that are not edges of S_i; the edges of E_A that have exactly one endpoint in S_i will have this endpoint replaced by the vertex c_i; and the edges of A that have both endpoints in S_i are removed.

Since the cluster S_i in G_i is located by finding a 6-cycle in C_i, we need to describe how C_{i+1} is constructed from C_i, i.e, the effect that T_i has on C_i (formally, T_i is a map from (G_i, C_i) to (G_{i+1}, C_{i+1})). To obtain C_{i+1}, we start with C_i and

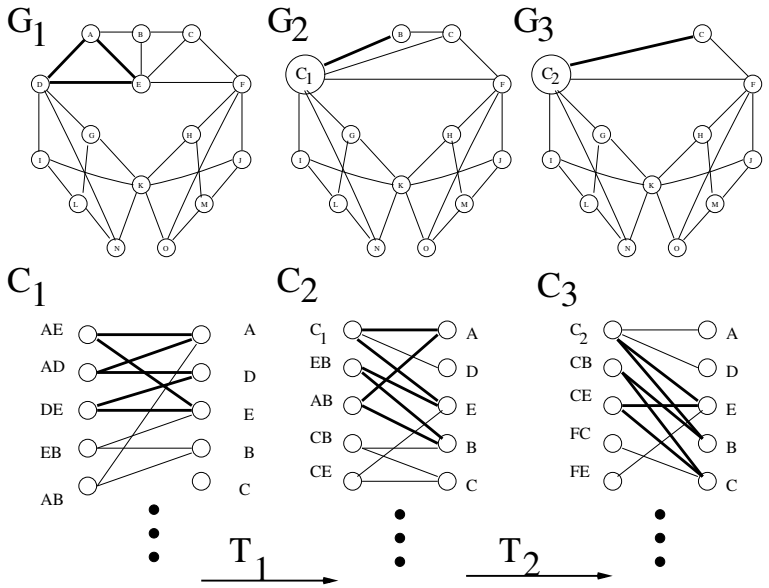


Figure 2–11: Action of the simplifier on G_i and C_i during Phase One

first add a new vertex c_i representing the cluster S_i, which is connected by edges to all the original vertices that it contains. Finally, vertices in C_i - that represent clusters entirely contained in S_i - are removed, and edges adjacent to these vertices are also removed.

Figures 2–11 and 2–12 illustrate the action of the subgraph simplifier T_i on both G_i and C_i.

A Further Concession for SR. Observe that the SR algorithm is not a general DR-planner when input geometric constraint graphs do not comply with assumption SR1. For example, for the graph shown in Figure 2–13, during Phase One, SR would not find any triangles and during Phase Two, it would conclude that the graph is underconstrained, even though the graph is wellconstrained.

SR implementations may remedy some cases similar to this one. For instance, weight 2 edges to weight 2 vertices often arise from incidence constraints between geometric elements of same type (two coincident points or two coincident lines), and such cases are easily accounted for by working with equivalence classes of such vertices. Moreover, most planners will evaluate the density of a graph before

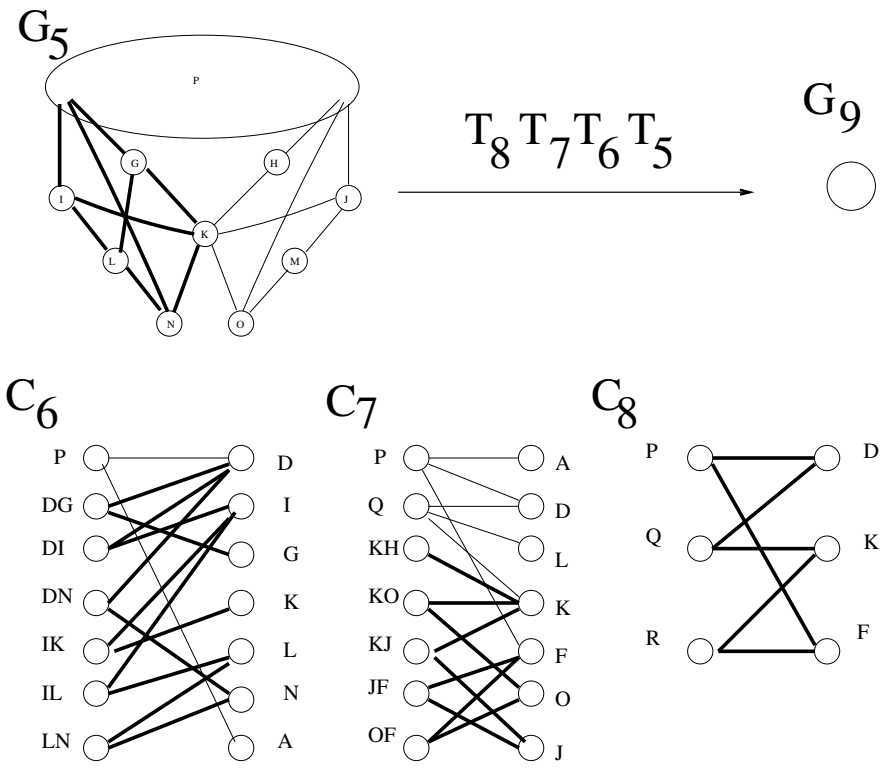


Figure 2–12: Action of the simplifier during Phase Two

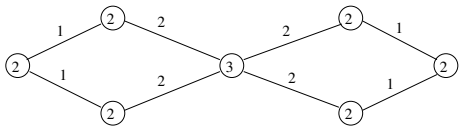


Figure 2–13: This solvable graph would not be recognized as solvable by SR

announcing under or overconstrained situations. Thus, an implementation of the SR algorithms of Fudos and Hoffmann (1996b, 1997) may or may not construct a DR-plan for the graph of Figure 2–13 depending on the original problem statement. It is clear that such heuristics enlarge the class of solvable graphs, but fall short of decomposing *all* solvable constraint graphs.

However, even when input graphs do comply with SR1, and SR checks overall density of a graph, SR could *still* mistake a graph that is not solvable for solvable. Figure 2–14 shows an example which the SR algorithm may process incorrectly: since the graph contains no solvable triangles, SR proceeds immediately to Phase

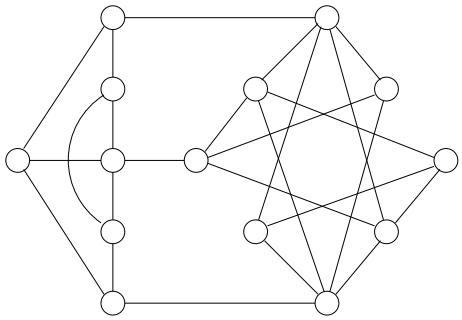


Figure 2–14: Weight of all vertices is 2, weight of all edges is 1

Two. During Phase Two the triconnectivity of the entire graph could be erroneously construed to mean that the graph is solvable. In fact, the graph does have density -3, which superficially seems to support such a conclusion. However, the graph is certainly not wellconstrained: the eight vertices on the right form a subgraph of density -2, that is, an overconstrained subgraph. Moreover, if this is overconstrained subgraph is replaced a subgraph of density -3, the resulting graph has density -4 uncovering that it is not welloverconstrained either, and therefore not solvable.

Figure 2–14 demonstrates that the density calculation heuristics is insufficient to determine the existence of minimal dense subgraphs of triconnected constraint graphs. What is needed is a general algorithm for finding minimal dense subgraphs.

In order to give a performance analysis of the SR algorithm for those classes of constraint graphs where it *does* produce a satisfactory DR-plan, we make a *strong concession (SR2)* that: only "triangular" structures are "acceptable" for the remainder of this section. I.e, we modify the definitions of validity, solvability preserving, strictly solvability preserving and completeness by replacing the words "solvable subgraph" by the following recursive definition: "either a subgraph that can be simplified into a single vertex by a sequence of consecutive simplifications of triangles of solvable subgraphs, (using the simplifier described in Section 2.1.9), or a vertex, edge or triconnected subgraph'.

Performance Analysis. In this section, we analyze the SR algorithm with respect to the various performance measures defined in Section 2.1.7.

Claim 3 *Under the concession SR2, the SR algorithm is valid.*

Proof We will show that every pre-image - of the "solvable" cluster S_i found at the i^{th} iteration of the SR algorithm - is also "solvable," using the stricter SR2 definition of "solvable". It can be easily checked that the other requirements necessary for validity from Section 2.1.7 also clearly hold.

Let S_i be a cluster that has been found in G_i by the SR algorithm (by locating a 6-cycle in C_i). Then from the description of the algorithm, and by assertion SR2, there is a sequence of simplifications, say $T_{i+1}, \ldots, T_m$ whose composition maps S_i to a single vertex. For every pre-image $A \subseteq G_j$ $A = T_j^{-1} \ldots T_{i-1}^{-1}(S_i)$, for $j \leq i-1$, the composition $T_m \circ \ldots T_i \circ T_{i-1} \circ \ldots \circ T_j$ will map A to a single vertex, hence A is "solvable". ∎

Claim 4 *The SR algorithm is strictly solvability preserving under the concession SR2, and therefore is solvability preserving as well.*

Proof Let A be a "solvable" subgraph, i.e there is a sequence of simplifications $A_1, ..., A_m$ such that $A = A_1$ and A_m consists of only one vertex. If the intersection of A and the currently found cluster S_i is empty then $T_i(A) = A$ and it remains "solvable". If this intersection is not empty, then a subgraph $B = A \cap S_i$ is simplified into a new cluster vertex $c_i = T_i(S_i)$. We need to show that in this case $T_i(A)$ remains "solvable" as well. Suppose that some subgraph of B is a vertex of a 6-cycle in one of the cluster graphs formed during the simplification $A_1, ..., A_m$. Then clearly the new cluster vertex c_i could perform the same function: i.e, it could also be a vertex in that 6-cycle of one of the cluster graphs formed during the simplification $A_1, ..., A_m$. Also if A was triconnected, then so is $T_i(A)$. This proves that there is a sequence (essentially a sequence $A_1, \ldots, A_m$ modified by replacing

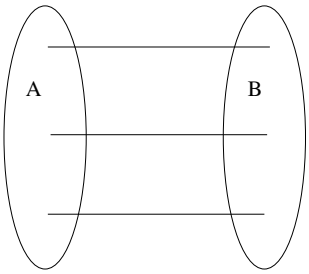


Figure 2–15: Two triconnected subgraphs not composed of triangles

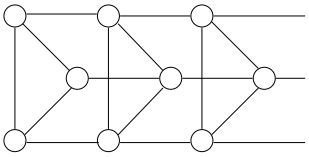


Figure 2–16: Solvable graph consisting of $n/3$ solvable triangles

B by c_i) which terminates in a single vertex, thus demonstrating the "density" of $T_i(A)$. ∎

Claim 5 *Even under concession SR2, SR is not complete.*

Proof We will show that there are cases when the SR picks large, nonminimal "solvable" subgraphs S_i to simplify, ignoring smaller "solvable" subgraphs of S_i.

Consider Figure 2–15. If subgraphs A and B are triconnected but not composed of triangles, then they are "solvable," but since the entire graph is triconnected neither A nor B will be simplified by SR. However, since the whole graph $A \cup B$ is triconnected, it will chosen by SR as S_1 during Phase Two. ∎

Claim 6 *The (worst and) best-choice approximation factor of SR under concession SR2 is at most* $O(\frac{1}{n})$.

Proof Consider Figure 2–16. The entire graph consists of $\frac{n}{3}$ triangles. During Phase One SR will successfully locate and simplify each one of them. However, during Phase Two SR will not be able to decompose the entire solvable graph into smaller solvable ones (since entire graph is triconnected) so the size of the corresponding DR-plan is $O(n)$. On the other hand, the optimal DR-plan would

simplify neighboring pairs of triangles, one pair at a time, thus the optimal size is a small constant. ∎

Next, we consider three other performance measures discussed in Section 2.1.7.

Claim 7 *Under the concession SR2, the algorithm SR adapts to underconstrained graphs.*

Proof Suppose that the graph G is underconstrained. Let A be a "solvable" subgraph that is not contained in any other "solvable" graph. Since Phase Two of the SR algorithm is top-down, either SR will find A and simplify it as one of the S_i's, or $A \cap S_i \neq \emptyset$ for one of the S_i. In either case, SR adapts to G. ∎

Observation 1 *Under the concession SR2, the SR algorithm has the Church-Rosser property, since the new graph G_{i+1} remains "solvable" if G_i is "solvable", regardless of the choice of the "solvable" S_i that is simplified at the i^{th} stage.*

Claim 8 *Under the concession SR2, the SR algorithm can incorporate design decompositions P if and only if P fulfills the following requirement: any pair of "solvable" subgraphs P_k and P_t in P satisfy $P_k \subseteq P_t$ or $P_t \subseteq P_k$ or $P_k \cap P_t$ contains no edges.*

Proof For the 'if' part we consider the most natural modification of the original SR algorithm, and find a topological ordering O of the given design decomposition P - which is a set of "solvable" subgraphs of the input graph G, partially ordered under the subgraph relation - such that O is embedded as a subplan of the final DR-plan generated by this modified SR algorithm; i.e, O forms a subsequence of the sequence of "solvable" subgraphs S_i, whose (sequential) simplification gives the DR-plan.

We take any topological ordering O of the given design decomposition P and create a DR-plan for the first "solvable" subgraph P_1 in P. I.e, while constructing the individual DR-plan for P_1, we "ignore" the rest of the graph. This individual DR-plan induces the first part of the DR-plan for the whole graph G. In particular,

the last graph in this partial DR-plan is obtained by simplifying P_1 using the simplifier described in 2.1.9 (and treating P_1 exactly as SR would treat a cluster S_j found at some stage j). Let G_i be the last graph in the DR-plan for G created thus far. Now, we consider the next subgraph P_2 in the ordering O, and find an individual DR-plan for it, treating it not as a subgraph of the original graph G, but as subgraphs of the simplified graph G_i. This individual DR-plan is added on as the next part of the DR-plan of the whole graph G.

The crucial point is that the simplification of any subgraph, say P_k, will not affect any of the unrelated subgraphs $P_t, t \geq k$, unless $P_k \subseteq P_t$. This is because, by the requirement on P, P_k and P_t share no edges. Therefore, when the cluster vertex for P_k is created, none of the clusters inside P_t is removed.

The process - of constructing individual DR-plans for subgraphs in the decomposition P and concatenating them to the current partial DR-plan - is continued until a partial DR-plan for the input graph G has been produced, which completely includes topological ordering O of the decomposition P as a subplan. Let G_p be the last graph in this partial DR-plan. The rest of the DR-plan of G is found by running the original SR algorithm on G_p and the corresponding cluster graph C_p.

For the 'only if' part, consider Figure 2–17. Let $P = \{P_0, P_1, P_2\}$, where $P_0 = ABD, P_1 = BCD, P_2 = ABCD$. Then SR cannot produce any DR-plan that would incorporate P as subplan. ∎

2.1.10 Generalized Maximum Matching (MM)

Consider the algorithms of Ait-Aoudia et al. (1993); Pabon (1993); Kramer (1992); Serrano and Gossard (1986); Serrano (1990) as well as graph rigidity and matroid based methods for distance constraints in 2 dimensions Hendrickson (1992), Gabow and Westermann (1988), Imai (1985) as well as more general constraints Sugihara (1985) all of which more or less use (generalized) maximum

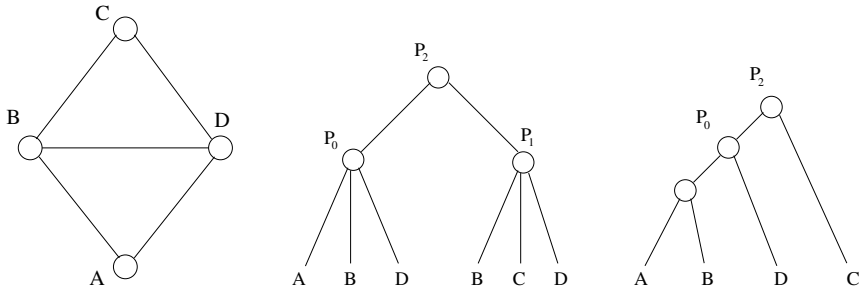


Figure 2–17: Constraint graph, intended and actual decompositions

matching (or equivalent maximum network flow) for finding solvable subgraphs in specialized geometric constraint graphs. These methods either assume that the constraint graph has zero density or they reduce the weight of an arbitrarily selected set of vertices in order to turn the graph into one of zero density.

In this thesis we will analyze what we consider to be the most general algorithm of this kind Pabon (1993) (it generalizes the algorithm of Ait-Aoudia et al. (1993), although the latter provides a more complete analysis), supplemented (by us, as suggested by a reviewer) with a method from Hendrickson (1992).

Note that while the algorithm of Pabon (1993) both locates solvable subgraphs and describes how to construct a corresponding DR-plan, Sugihara (1985), Hendrickson (1992), Gabow and Westermann (1988) only describe algorithms that allow to verify whether a given graph is solvable, but do not explicitly state how to use these algorithms for successively decomposing into small solvable subgraphs, i.e, for constructing DR-plans.

While neither of the previously known algorithms analyzed in this thesis perform well according to our previously defined set of criteria, later, we will describe our network flow based *Frontier Algorithm* that improves the performance in several key areas.

Description of the Algorithm. As in the case of SR, we give a terse description of the MM algorithm – the reader is referred to Ait-Aoudia et al. (1993);

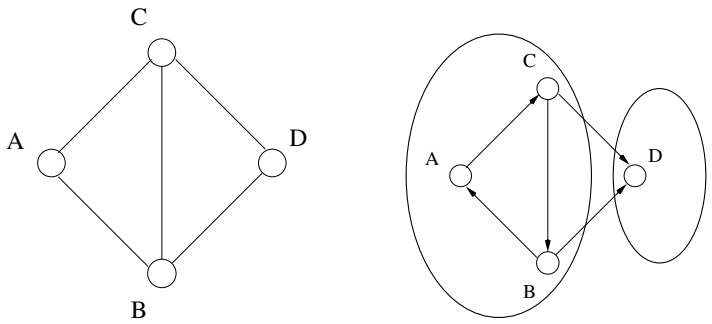


Figure 2–18: Original graph and it decomposition

Pabon (1993) for details. Our description is meant only to put the algorithm into an appropriate DR-planner framework that is suited for the performance analysis.

By using maximum flow, the input constraint graph is decomposed into a collection of subgraphs that are strongly connected. The flow information also provides a partial ordering of the strongly connected subgraphs representing the order in which these subgraphs should be simplified. This ordering in turn specifies the DR-plan. It is important to note that these strongly connected components represent a sequence of solvable subgraphs of the original constraint graph, *only if* the input constraint graph is wellconstrained.

Consider Figure 2–18. All the vertices have weight 2, all the edges have weight 1, and the geometry is assumed to be in 2 dimensions (i.e geometry dependent constant $D = 3$). The output of the MM algorithm is:

–a set of vertices whose total weight is reduced by 3 units, say weight of vertices A,B and C to be reduced by one unit each, (this corresponds to fixing 3 degrees of freedom - the number of degrees of freedom of a rigid body in 2 dimensions);

–two strongly connected components ABC and D;

–the DR-plan, i.e, the information that the subsystem represented by ABC should be simplified/solved first and then its union with D should be simplified/solved.

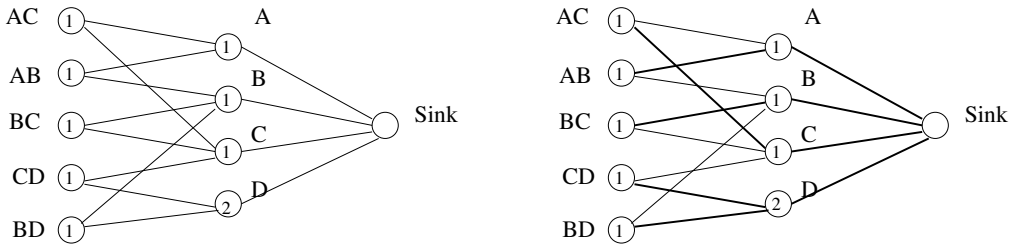


Figure 2–19: Modified bipartite graph and maximum flow in this graph

In order to produce such an output, the MM algorithm first constructs the network $X = (VX, EX)$ corresponding to the original weighted constraint graph $G = (V, E)$ (after the weights of A, B, C were reduced by one unit each). The set of vertices VX is the union of VX_1 and VX_2 where the vertices in VX_1 correspond to the vertices in V, vertices in VX_2 correspond to the edges in E. An edge $ex \in EX$ would be created between $vx_1 \in VX_1$ and $vx_2 \in VX_2$ if the vertex in V corresponding to the vx_1 is an endpoint of an edge in E corresponding to vx_2. The edge ex has infinite capacity. All the vertices in VX_1 are connected to the special vertex called *Sink*. The capacity of connecting edges is equal to the weight of the corresponding vertices in V. For example, the left half of Figure 2–19 shows the bipartite graph corresponding to the constraint graph of Figure 2–18. Next the maximum flow in the network X is found, with vertices in VX_2 being source vertices of capacity equal to the weights of the corresponding edges in E. See right half of Figure 2–19 (thick edges have nonzero flow).

The maximum flow found in X induces a partition of the original graph G into a partially ordered set of strongly connected components – giving a sequence of solvable subgraphs of G, provided G is wellconstrained – as in Figure 2–18, according to the following rules: if the flow in X from the vertex $z \in VX_2$ that is connected to the vertices $x, y \in VX_1$ is sent toward x, then in the graph G, the edge corresponding to z (between vertices x and y) becomes an oriented edge directed from y to x. If the flow from z is sent toward both x and y, then the edge

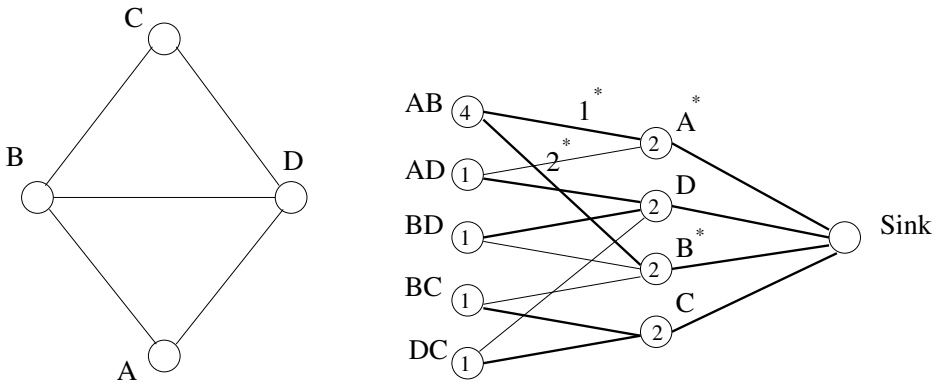


Figure 2–20: Constraint graph and network flow with 3 extra flow units at AB

is bidirected toward both x and y. (Recall that a strongly connected component S in this directed graph is a subgraph such that for any two vertices $a, b \in S$, there is an oriented path from a to b.) The partial ordering of components is induced as follows. Let K and L be two strongly connected components in the oriented version of G. If all the edges between vertices of K and L are pointing toward L, then L should be simplified after K.

Defining the Simplifier Map. We capture the transformations performed by the MM DR-planner described above, by describing its simplifier maps (recall the definitions in Section 2.1.7).

Let $G = (V, E)$ be the geometric constraint graph. Denote by $G_1 = (V_1, E_1 = E)$ the directed graph after weights of some vertices have been reduced as described above and a partial ordering of strongly connected components has been found. First $S_1 \subseteq G_1$ is located such that S_1 is strongly connected. Then S_1 is simplified into the vertex v_1 of weight zero. The other vertices of G_1 remain unchanged. Edges of G_1 that had both endpoints outside of S_1 are unchanged, edges that had both endpoints in S_1 are removed, edges that had exactly one endpoint in S_1 have this endpoint replaced by v_1. In the next step, C_2 - the next strongly connected component in a topological ordering of the components in $G_2 = T_1(G_1)$ - is located. The subgraph S_2 is set to be $\{v_1\} \cup C_2$. Then S_2 is simplified into the vertex v_2

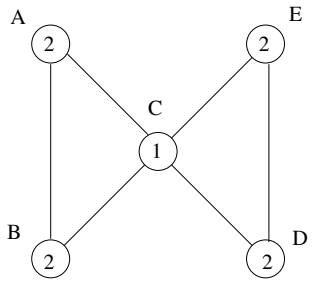


Figure 2–21: DR-plan depends on the initial choice

of weight zero. This process is continued until G_k consists of one vertex. The simplifier maps are formally defined as follows.

–$T_i(S_i) = v_i$ where v_i is the vertex in G_{i+1} of weight 0.

–If $B \subseteq G_i, B \cap S_i = \emptyset$, then $T_i(B) = B$.

–If $B \subseteq G_i, B \cap S_i \neq \emptyset$, then the image of B, $T_i(B) = \{v_i\} \cup (B \setminus S_i)$.

Redefining Solvability.

Claim 9 *The MM algorithm is a not a general DR-planner.*

Proof While the MM algorithm can correctly classify as solvable and decompose wellconstrained and welloverconstrained graphs and correctly classify underconstrained graphs that have no overconstrained subgraphs as being unsolvable, it is unable to correctly classify an underconstrained graph that has an overconstrained subgraph. Consider Figure 2–21, the weight of all the edges is 1, of the vertices as indicated. Graph $ABCDE$ has density -3, it contains overconstrained subgraphs AC, BC, CD, CE, and removal of say edge AC will result in $ABCDE$ becoming non-dense, hence $ABCDE$ is not welloverconstrained and not solvable. The MM algorithm may or may not detect this, depending on the initial choice of vertices whose weights are to be reduced. Suppose that the weight of the vertex E was reduced by 1 and the weight of the vertex D by 2. Then the corresponding maximum possible flow f and the corresponding decomposition into strongly components are shown in Figure 2–22. Note that it is impossible to simplify the strongly connected component A, and thus there can be no DR-plan

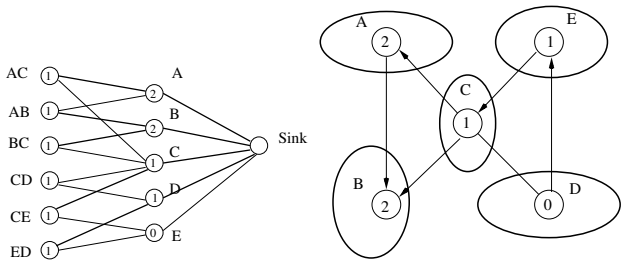


Figure 2–22: Maximum flow and decomposition given the bad initial choice

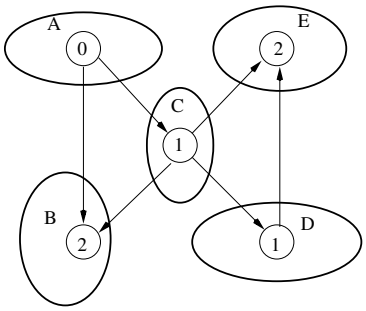


Figure 2–23: Decomposition given the good initial choice

for this initial choice of vertices for reducing weights. On the other hand, if the weight of A had been reduced by 2 and weight of D by 1 then the maximum flow is larger than f (in fact, no choice of vertices for weight reduction can give a larger flow) and it can be checked that MM would yield a DR-plan, see Figure 2–23 $(S_1 = A, S_2 = AC, S_3 = ABC, S_4 = ABCD, S_5 = ABCDE)$. ∎

Thus MM runs into problems in the presence of overconstrained subgraphs, unless G happens to be welloverconstrained, which cannot be apriori checked without relying on an algorithm for detecting overconstrained subgraphs and replacing them by wellconstrained ones. Therefore it is necessary and sufficient to locate overconstrained subgraphs and replace them by wellconstrained ones, in order to guarantee that MM will work.

The author of Pabon (1993) does *not* specify how to do this. The following modification similar to that of Hendrickson (1992) could be used, as was suggested by a reviewer, and completed here.

First a maximum flow in the unmodified network X (i.e where vertex weights are not reduced) is found. After that, for every source vertex $v \in VX_2$, except one vertex v_m, the following two steps are repeated.

–Three extra units of flow are sent from v, possibly rearranging existing flows in X.

–These 3 units of flow are removed, without restoring original flows.

–For vertex v_m, 3 units of flow are sent but not removed.

For example, consider Figure 2–20. The constraint graph is shown on the left, weights of all vertices is 2, the weights of all edges is 1. The resulting network flow is shown on the right, assuming that AB was the last vertex v_m. The extra 3 units of flow and their destination vertices A and B are marked by asterisks. This particular flow induces a weight reduction of A by 1 unit and of B by 2 units.

This modification identifies overconstrained subgraphs since it is impossible to send all 3 extra units from at least one edge of an overconstrained graph. These overconstrained graphs have to be modified by the designer to become wellconstrained, and the flow algorithm is *run again*.

This is *important* because otherwise the DR-algorithm cannot proceed, since in underconstrained graphs with overconstrained subgraphs, strongly connected components do not necessarily correspond to a sequence of solvable subgraphs.

To reflect the modification above, we make the following *concession* for MM,

(MM1): the input constraint graph G has no overconstrained subgraphs in G. A subgraph $A \subseteq G$ is "solvable" if it has zero density, after the vertices for weight reduction by D are chosen (these can be chosen arbitrarily, provided there are no overconstrained subgraphs).

Performance Analysis. In this section, we analyze the MM algorithm with respect to the various performance measures defined in Section 2.1.7.

Note that despite the concession MM1, the MM algorithm has the following drawbacks: the DR-plan is uniquely determined, once vertices whose weights are reduced are chosen. For example, in Figure 2–20 after the weight of A is reduced by 1 and the weight of B by 2 units, solvable subgraphs to be simplified are $S_1 = \{A, B\}, S_2 = \{T_1(S_1), D\}, S_3 = \{T_2(S_2), C\}$. While the subgraph BCD is also solvable (if say initially weights of vertices B and C were reduced instead of A and B), it cannot be chosen as one of the S_i after weights of A and B are reduced.

This causes MM to have bad worst-choice and best-choice approximation factors and to be unable to incorporate general designer decomposition.

The following is straightforward from the description of the simplifiers.

Claim 10 *Under the concession MM1, the MM algorithm is a valid DR-planner.*

Claim 11 *Under the concession MM1, the MM algorithm is strictly solvability preserving (and therefore solvability preserving).*

Proof Suppose that a subgraph A of the input graph G is "solvable," i.e $d(A) = 0$. Let S_i be the "solvable" subgraph to be simplified at the current stage. Let $B = A \cap S_i, C = A \setminus B$. Since we assume that G does not contain any over-constrained subgraphs, $d(B) \leq 0$ and $d(A \cup S_i) \leq 0 \Rightarrow d(A \cup S_i) = d(A) + d(S_i) - d(B) \leq 0 \Rightarrow 0 + 0 - d(B) \leq 0 \Rightarrow d(B) = 0$. Thus $d(T_i(A)) = d(T_i(C)) + d(T_i(B)) = d(C) + 0 = d(A) - d(B) = 0$, therefore $T_i(A)$ is also solvable. ∎

Claim 12 *Under the concession MM1, the MM algorithm is complete.*

Proof Let A be a proper solvable subgraph of the $S_i, i \geq 2$. Since A is solvable, there cannot be any edges outside of A pointing toward A (this is because there is no room for flows of "outside" edges toward the vertices of A). Recall that S_i is the union of $C_i \cup \{v_{i-1}\}$, where C_i is the first strongly connected component

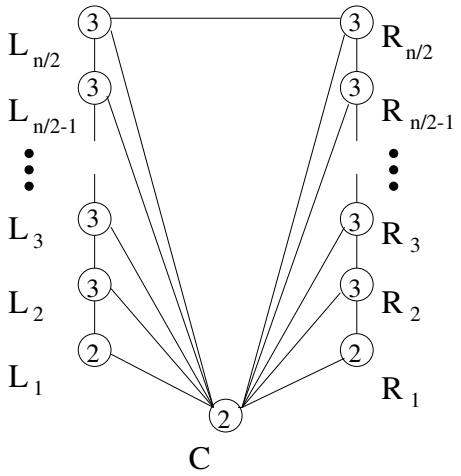


Figure 2–24: Bad best-choice approximation

in the topological ordering of the components at the stage i, and v_{i-1} is the simplification of S_{i-1}. However, unless $A = v_{i-1}$, the first strongly connected component at this stage would have been $A \setminus \{v_{i-1}\}$, not C_i, which contradicts the choice of C_i at stage i. ∎

Claim 13 *Under the concession MM1, the best-choice (and worst-choice) approximation factor of MM is at most* $O(\frac{1}{n})$.

Proof To prove the bound on the best-choice approximation factor consider Figure 2–28. The left and right columns contain $n/2$ vertices each. The weights of all the vertical edges are 2, the weights of all other edges are 1, the weights of the vertices are as indicated, and the geometry dependent constant $D = 3$.

Note that all solvable subgraphs in Figure 2–28 could be divided into 3 classes. The first class consists of the subgraphs $CL_1L_2; CL_1L_2L_3; \ldots; CL_1L_2 \ldots L_{n/2-1}L_{n/2}$. The second class consists of the subgraphs $CR_1R_2; CR_1R_2R_3; \ldots; CR_1R_2 \ldots R_{n/2-1}R_{n/2}$. The third class contains the solvable subgraphs that contain both L and R vertices. There is only one element in this class - the entire graph $CL_1L_2 \ldots L_{n/2}R_1R_2 \ldots R_{n/2}$. There is an optimal DR-plan of constant size that takes $S_1 = CL_1L_2, S_2 = S_1 \cup L_3, \ldots, S_{n/2-1} = S_{n/2-2} \cup L_{n/2}$. After that it takes $S_{n/2} = CR_1R_2, S_{n/2+1} = S_{n/2} \cup R_3, \ldots S_n = S_{n-1} \cup R_{n/2}$. Finally it takes $S_{n+1} = S_{n/2-1} \cup S_n$.

However all DR-plans found by MM will have size $O(n)$. The reason for this is that MM is unable to simplify solvable subgraphs on the left of the Figure 2–28 independently from the solvable subgraphs on the right. More formally let S_1 be the first subgraph simplified by MM under some DR-plan Q. If S_1 belongs to the third class of solvable subgraphs then size of Q is $O(n)$. Otherwise, without loss of generality we can assume that S_1 belongs to the first class. According to the definition of MM, the simplification of S_1 is a vertex v_1 of weight 0. After this simplification any strongly connected component that contains some R_i should also contain all of the $R_1 \dots R_{n/2}$. Hence there is an S_i in Q such that $R_1 R_2 \dots R_{n/2} \subset S_i$. Hence the size of Q is $O(n)$. ∎

Next, we consider three other performance measures discussed in Section 2.1.7.

Claim 14 *Under the concession MM1, a simple modification of the MM algorithm is able to adapt to underconstrained graphs. This modification, however, increases the complexity by a factor of n.*

Proof Suppose that the graph G is underconstrained. Consider the maximum flow found by MM in the network X corresponding to G, as described in subsection 2.1.10. There are two cases.

The first case is when the last vertex v_m in X corresponds to an edge of some solvable subgraph A_1. Then all vertices v_i in X corresponding to vertices of A_1 have their capacities completely filled, since A_1 is solvable. On the other hand, at least one vertex v of G will not have its capacity filled completely, since G is underconstrained. Let W be the set of such vertices v. The new modification of MM would proceed by removing all vertices of X corresponding to vertices of W and edges adjacent to W, as well as flows originating at such edges. Once this is done, a new set W is recomputed and removed, until all remaining vertices of X that represent vertices of G have their capacities filled completely. These vertices comprise a solvable subgraph A_1 such that no supergraph of A_1 is solvable. Once

the subgraph A_1 (and its DR-plan Q_1) is found, it could be removed from G and the process applied recursively to $G \backslash A_1$ to find a solvable subgraph A_2, its DR-plan Q_2 and so on.

The second case is when the last vertex v_m in X is not contained in any solvable subgraph. Then constructing and removing sets W as described above will completely exhaust G, without finding any solvable graphs. The new modification of MM would proceed by finding another maximum flow in network X corresponding to G such that the last vertex v_m^i is different from v_m. Again two cases are considered for vertex v_m^i. If all $v_m^i \in G$ are not contained in any solvable subgraph, then G does not contain any solvable subgraph and the process terminates.

Note that this modification offhand increases the complexity of MM by a factor of n, and there does not seem to be any obvious way to prevent this factor.

This modification of MM outputs a DR-plan $Q = Q_1, \ldots, Q_k$ for set of solvable subgraphs $A_1, \ldots, A_k$ such for any i no supergraph of A_i is solvable and there are no solvable subgraphs $B \subseteq G$ such that $B \cap A_j = \emptyset, \forall j$. Thus this modification of MM is able to adapt to underconstrained graphs. ∎

Observation 2 *(i) Under the concession MM1, MM has the Church-Rosser property since simplifying any S_i that is solvable at the current stage preserves the density of the whole graph G_{i+1} (i.e G_{i+1} is solvable if and only if G_i is).*

(ii) Under the concession MM1, MM is able to incorporate a design decomposition P specified by the designer if and only if for every $P_k, P_t \in P$ such that $P_k \cap P_t \neq \emptyset$ either $P_k \subseteq P_t$ or $P_t \subseteq P_k$.

Proof is similar to the corresponding proof for SR algorithm in Claim 8.

2.1.11 Comparison of Performance of SR and MM

Next, we give a table comparing the SR and MM DR-planners with respect to the performance measures of Section 3. "Underconstr" refers to the ability to

deal with underconstrained graphs, "Design decomposition" refers to the ability to incorporate design decompositions specified by the designer.

The superscript '*' refers to a narrow class of DR-plans: those that require the solvable subsystems S_i to be based on triangles or a fixed repertoire of patterns. The superscript '†' refers to results that were not true for the original MM algorithm developed by (Ait-Aoudia et al. (1993); Pabon (1993)) and proved in this thesis through a modification of MM described in Section 2.1.10. The superscript $\diamondsuit$ refers to a restricted class of graphs in which there are no overconstrained subgraphs. It also refers to a further modification of MM described in Claim 14, which however, increases the complexity by *a factor of* n. The superscript '$\circ$' refers to strong restrictions on the design decompositions that can be incorporated into DR-plans by SR and MM.

2.1.12 Analysis of Two New DR-planners

We present two new Decomposition-Recombination (DR) planning algorithms or DR-planners. The new planners follow the overall structural description of a typical DR-planner, based on the DR-solver $\mathcal{S}$ given in previous sections. Furthermore, the new DR-planners adopt and adapt features of older decomposition methods such as SR (shape recognition) and MM (generalized maximum matching) that were analyzed and compared previously. In particular, those methods as well

Performance measure	SR	MM
Generality	No	Yes†
Underconstr.	No(Yes*)	Yes$^{\dagger,\diamondsuit}$
Design decomposition	No(Yes$^{*,\circ}$)	No(Yes$^{\dagger,\circ}$)
Validity	No(Yes*)	Yes†
Solvability	No(Yes*)	Yes†
Strict solvability	No(Yes*)	Yes†
Complete	No(No*)	Yes†
Worst approximation factor	0 ($O(\frac{1}{n})^{*}$)	$O(\frac{1}{n})^{\dagger}$
Best approximation factor	0 ($O(\frac{1}{n})^{*}$)	$O(\frac{1}{n})^{\dagger}$
Church-Rosser	No(Yes*)	Yes†
Complexity	$O((m+n)^2)$	$O(n(m+n))^{\dagger}$

as the new planners are based on *degree of freedom* analysis of geometric constraint *hypergraphs*.

It should be noted that the SR and MM based algorithms Bouma et al. (1995); Owen (1991, 1993); Bouma et al. (1995); Hoffmann and Vermeer (1994), Hoffmann and Vermeer (1995); Latham and Middleditch (1996); Fudos and Hoffmann (1996b, 1997), Ait-Aoudia et al. (1993); Pabon (1993); Kramer (1992); Serrano and Gossard (1986); Serrano (1990), were being developed even as the issue – of efficient decomposition of constraint systems for capturing design intent in CAD/CAM – was still in the process of crystallization.

In contrast, our development of the new DR-planners is *systematically guided* by the new performance measures, to closely reflect several desirable characteristics $\mathcal{C}$ of DR-planners for CAD/CAM applications.

An important *building block* of both the new DR-planners is the routine used to isolate the solvable subsystems S_i at each step i. In both new DR-planners, the solvable subsystem/subgraph S_i is isolated using an algorithm developed by the authors based on a subtle modification of incremental network flow: this algorithm, called "Algorithm Dense," first *isolates* a dense subgraph, and then finds a minimal dense subgraph inside it, which ensures its solvability. The interested reader is referred to earlier papers by the authors: Hoffmann et al. (1997) for a description as well as implementation results, and Hoffmann et al. (1998) for a comparison with prior algorithms for isolating solvable/dense subgraphs. Here, we shall only note several desirable features of Algorithm Dense.

(a) A useful property of the dense subgraph G' found by the Algorithm Dense (in time $O(n(m+n))$) is that the densities of all proper subgraphs are strictly smaller than the density of G'. Therefore, when G' corresponds to a wellconstrained subsystem, then G' is in fact minimal, and hence it is unnecessary

to run the additional steps to obtain minimality. Ensuring minimality crucially affects completeness of the new DR-planners.

(b) A second advantage of Algorithm Dense is that it is much simpler to implement and faster than standard max-flow algorithms, This was borne out by our C++ implementation of Algorithm Dense both for finding dense and minimal dense subgraphs. By making *constantD* a parameter of the algorithm, our method can be applied uniformly to planar or spatial geometry constraint graphs. Furthermore, the new algorithm handles not only binary but also ternary and higher-order constraints which can be represented as *hyperedges*.

(c) A third specific advantage of our flow-based algorithm for isolating dense subgraphs (solvable subsystems) is that we can run the algorithm on-line. That is, the constraint graph and its edges can be input continuously to the algorithm, and for each new vertex or edge the flow can be updated accordingly. This promises a good fit with interactive, geometric constraint solving applications, i.e, the criterion (viii) of $\mathcal{C}$.

As will be seen below the main *difference* between the two new planners, however, lies in their simplifier maps T_i, i.e, the way in which they abstract or simplify a solvable subgraph S_i once it has been found.

Comparison. As a prelude to the analysis of the new DR-planners Condense and Frontier we give a table below which extends the comparison between the SR and MM to the new DR-planners Condense and Frontier.

The "complexity" entries for the 2 new DR-planners are directly based on the complexity of a building block Algorithm Dense (briefly discussed above) for isolating minimal dense subgraphs S_i. "Under-const" refers to the ability to deal with underconstrained systems, "Design-dec" refers to the ability to incorporate design decompositions specified by the designer, "Solv." and "Strict solv." refer

to (strict) solvability preservation, "Worst and Best approx" refer to the worst and best choice approximation factor.

Perf. meas.	SR	MM	Condense(new)	Frontier
Generality	No	Yes†	Yes	Yes
Under-const	No(Yes*)	No	Yes	Yes
Design-dec	No(Yes$^{*,\circ}$)	No	No(Yes$^{\circ}$)	Yes
Validity	No(Yes*)	Yes^{+}	Yes	Yes
Solv.	No(Yes*)	Yes^{+}	Yes	Yes
Strict solv.	No(Yes*)	Yes^{+}	No	Yes
Complete	No(No*)	No	Yes	Yes
Worst approx.	0 ($O(\frac{1}{n})^{*}$)	$O(\frac{1}{n})$	$O(\frac{1}{n})$	$O(\frac{1}{n})$
Best approx.	0 ($O(\frac{1}{n})^{*}$)	$O(\frac{1}{n})^{\dagger}$	$O(\frac{1}{n})$	$O(\frac{1}{2})$
Church-Rosser	No(Yes*)	Yes†	Yes	Yes
Complexity	$O(s^2)$	$O(n^{D+1}s)^{\dagger}$	$O(n^3 s)$	$O(n^3 s)$

Note. The variable s in the complexity expressions denotes the number of vertices, n, plus the number of edges, m, of the constraint graph. Recall that D refers to the number of degrees of freedom of a rigid object in the input geometry (in practice, this could be treated as a constant). ♢

The superscripts '*' and '+' refer to narrow classes of DR-plans: those that require the solvable subsystems S_i to be based on triangles or a fixed repertoire of patterns, or to represent rigid objects that fixed or grounded with respect to a single coordinate system. The superscript '†' refers to results that were left unproven by the developers of the MM based algorithms (Ait-Aoudia et al. (1993), Pabon (1993)) and proved in this paper through a crucial modification of MM described previously. The modification also results in the improvement of the complexity of the best MM algorithm to $O(n(s = n + m))$.

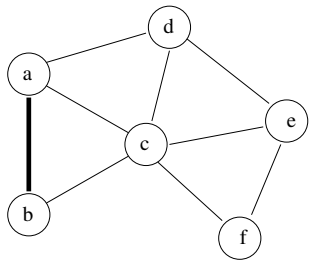


Figure 2–25: Sequential extension

The superscript '$\circ$' refers to a strong restriction on the design decompositions that can be incorporated into DR-plans by SR and the new DR-planner Condense. In fact, the other new DR-planner Frontier also places a (however, much weaker) restriction on the design decompositions that it can incorporate, as will be shown later.

2.1.13 The DR-planner Condense and its Performance

This DR-planner was sketched by the authors already in Hoffmann et al. (1998). The authors' flow-based algorithm discussed above Hoffmann et al. (1997) is applied repeatedly to constraint graphs to find minimal dense subgraphs or clusters (which we know to be generically solvable containing more than one vertex. DR-planner Condense consists of two conceptual steps. A minimal dense cluster can be *sequentially extended* under certain circumstances by adding more geometric objects one at a time, which are rigidly fixed with respect to the cluster. After a cluster has been thus extended, it is then simplified into a single geometric object, and the rest of the constraint graph is searched for another minimal dense subgraph. The following example illustrates sequential extension. Consider the constraint graph G of Figure 2–25. We assume that all vertices have weight 2 and all edges have weight 1. The geometry-dependent constant $D = 3$. The vertex set $\{a, b\}$ induces a minimal dense subgraph of G which could be chosen by DR-planner Condense as the initial minimal dense cluster, which could be extended sequentially by the vertices c, d, e, f, one vertex at a time, until it cannot be

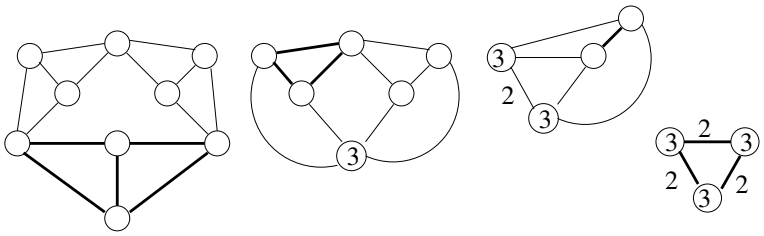


Figure 2–26: Sequence of simplifications from left to right

extended any further. The resulting subgraph is called an *extended dense subgraph or cluster*.

The simplification of an extended cluster is taken to be a single geometric object with D degrees of freedom. This is done as follows: an extended cluster A is replaced by a vertex u of weight D; all edges from vertices in A to a vertex w outside A are combined into one edge (u, w), and the weight of this induced edge is the sum of the weights of the combined edges. After the simplification, another solvable subgraph is found, and the process is continued until the entire graph is simplified into a single vertex.

This is illustrated by the sequence of simplifications of Figure 2–26. Initially all vertices have weight 2, all edges have weight 1. The vertices connected by the heavy edges constitute minimal or sequentially extended clusters. After four simplifications the original graph is replaced by one vertex.

Defining Subsystem Simplifiers. We capture the transformations performed by the DR-planner Condense by describing simplifier maps. Let G be the input constraint graph; the first graph G_1 in the DR-plan is the original graph G. Let $G_i = (V, E)$ be the current graph and let S_i be a cluster found at the current stage. Let A be any subgraph of G_i. Then $T_i(A)$ is defined as follows.

- If $A \cap S_i = \emptyset$ then $T_i(A) = A$.
- If $A \cap S_i \neq \emptyset$ then $T_i(A) = (V_{TA}, E_{TA})$ where V_{TA} is the set of all vertices of A that are not vertices of S_i plus one vertex c_i of weight D which represents

the simplification of the cluster S_i. The set of edges E_{TA} is formed by removing edges with all endpoints in S_i, and combining edges with at least one endpoint outside S_i, (as well as their weights) as described earlier in this section.

Performance Analysis. In this section, we analyze the DR-planner Condense with respect to the various performance measures that were defined in preceeding sections.

Claim 15 *Condense is a valid DR-planner with the Church-Rosser property. In addition, DR-planner Condense finds DR-plans for the maximal solvable subgraphs of underconstrained graphs.*

Proof If the graph G is not underconstrained, then it will remain so after the replacement of any solvable subgraph by a vertex of weight D, i.e, after a simplification step by DR-planner Condense. Thus, if $G = G_1$ is wellconstrained, it follows that all of the G_i are wellconstrained. Moreover, we know that if the original graph is solvable, then at each step, DR-planner Condense will in fact find a minimal dense cluster S_i that consists of more than one vertex, and therefore G_{i+1} is strictly smaller than G_i for all i. Thus the process will terminate at stage k when G_k is a single vertex. This is independent of which solvable subgraph S_i is chosen to be simplified at the i^{th} stage, showing that DR-planner Condense has the Church-Rosser property.

On the other hand, if G is underconstrained, since the subgraphs S_i chosen to be simplified are guaranteed to be dense/solvable, the process will not terminate with one vertex, but rather with a set of vertices representing the simplification of a set of maximal solvable subgraphs (such that no combination of them is solvable). This completes the proof that DR-planner Condense is a DR-planner that can adapt to underconstrained graphs.

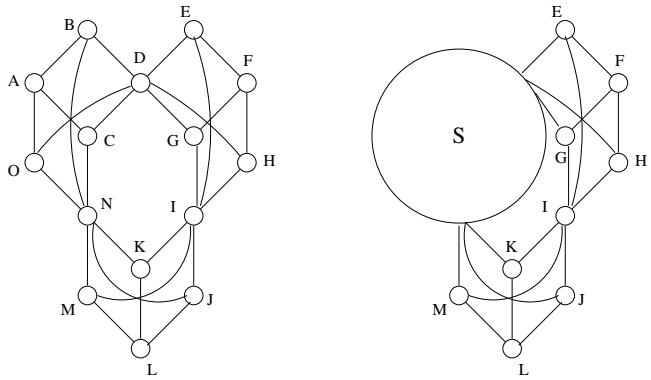


Figure 2–27: Original and simplified graphs

The proof of validity follows straightforwardly from the properties of the simplifier map. ∎

Claim 16 *DR-planner Condense is solvability preserving.*

Proof The simplifier maps T_i do not affect subgraphs outside of S_i. ∎

Claim 17 *DR-planner Condense is not strictly solvability preserving.*

Proof Consider the constraint graph of Figure 2–27. The vertex weights are 2, the edge weights are 1, and the geometry-dependent constant $D = 3$. The graphs $ABCDNO, EFHGID$ and $NMKLIJ$ are all dense/solvable. Suppose that first the cluster $S_1 = ABCDNO$ was found and simplified into one vertex S of weight 3. Now the graph $SEFHGI = T_1(EFHGID)$ is not dense/solvable anymore. ∎

Intuitively, the reason why DR-planner Condense is not strictly solvability preserving is that the removal of the vertices D and N loses valuable information about the structure of the solvable graph.

Claim 18 *DR-planner Condense is complete.*

Proof This is because DR-planner Condense finds minimal dense subgraphs at each stage. ∎

Claim 19 *Best-choice (and worst-choice) approximation factor of DR-planner Condense is at most* $O(1/n)$.

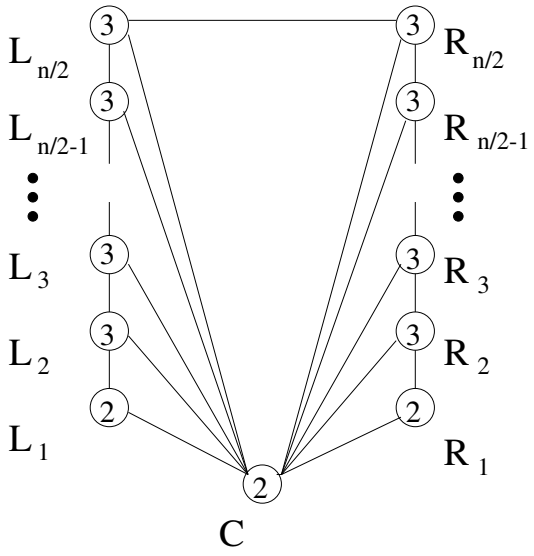


Figure 2–28: Bad best-choice approximation

Note. This proof mimics the MM approximation factor proof, except that now the subgraph S_i is not a strongly connected component. ♢

Proof To prove the bound on the best-choice approximation factor consider Figure 2–28. The left and right columns contain $n/2$ vertices each. The weights of all the vertical edges are 2, the weights of all other edges are 1, the weights of the vertices are as indicated, and the geometry dependent constant $D = 3$.

Note that all solvable subgraphs in Figure 2–28 could be divided into 3 classes. The first class consists of the subgraphs $CL_1L_2; CL_1L_2L_3; \ldots ; CL_1L_2 \ldots L_{n/2-1}L_{n/2}$. The second class consists of the subgraphs $CR_1R_2; CR_1R_2R_3; \ldots ; CR_1R_2 \ldots R_{n/2-1}R_{n/2}$. The third class contains the solvable subgraphs that contain both L and R vertices. There is only one element in this class - the entire graph $CL_1L_2 \ldots L_{n/2}R_1R_2 \ldots R_{n/2}$. There is an optimal DR-plan of constant size that takes $S_1 = CL_1L_2, S_2 = S_1 \cup L_3, \ldots , S_{n/2-1} = S_{n/2-2} \cup L_{n/2}$. After that it takes $S_{n/2} = CR_1R_2, S_{n/2+1} = S_{n/2} \cup R_3, \ldots S_n = S_{n-1} \cup R_{n/2}$. Finally it takes $S_{n+1} = S_{n/2-1} \cup S_n$.

However all DR-plans found by DR-planner Condense will have size $O(n)$. The reason for this is that DR-planner Condense is unable to simplify solvable subgraphs on the left of the Figure 2–28 independently from the solvable subgraphs

on the right. More formally let S_1 be the first subgraph simplified by DR-planner Condense under some DR-plan Q. If S_1 belongs to the third class of solvable subgraphs then the size of Q is $O(n)$. Otherwise, without loss of generality we can assume that S_1 belongs to the first class. According to the definition of DR-planner Condense, $T_1(S_1)$ is a single vertex of weight 3 that replaces several vertices including vertex C. Now for any graph A that belongs to the second class, $T_1(A)$ is not solvable anymore (it has density -4). Hence there is an S_i in Q such that $R_1R_2 \ldots R_{n/2} \subset S_i$. Hence the size of Q is $O(n)$. ∎

Next, we consider the last performance measure - incorporating input decomposition.

Observation 3 *In general, DR-planner Condense cannot incorporate a design decomposition of the input graph for reasons similar to those given for the SR algorithm. It can incorporate a design decomposition, only if every pair A and B of subgraphs in the decomposition are either completely disjoint or $A \subseteq B$ or $B \subseteq A$.*

2.1.14 The DR-planner Frontier and its Performance

Intuitively, the reason why DR-planner Condense is not strictly solvability preserving is that the simplification of a minimal or extended dense cluster into a single vertex loses valuable information about the structure of the cluster. The algorithms described in this section preserve this information at least partially by designing a simplifier that keeps the structure of the *frontier vertices* of the cluster, i.e, those vertices that are connected to vertices outside of the cluster. However, the way in which the minimal dense clusters and their sequential extensions are found is identical to that of DR-planner Condense – i.e, by using the authors' Algorithm Dense from Hoffmann et al. (1997).

Informally, under DR-planner Frontier, all internal (i.e not frontier) vertices of the solvable subgraph S_i found at the i^{th} stage are simplified into one vertex called the *core* c_i. The core vertex is connected to each frontier vertex v by an edge whose

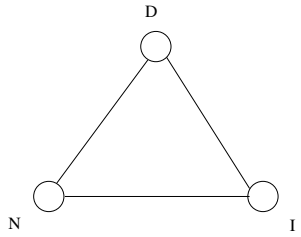


Figure 2–29: The simplified graph after three clusters has been replaced by edges

weight is equal to that of v. All other edges, except ones between frontier vertices, of S_i are removed. This is repeated until the solvable subgraph S_m found is the entire remaining graph G_m.

If the solvable subgraph S_i has only two frontier vertices, then all internal vertices of S_i should be removed and no new core vertex created. Instead the two frontier vertices of S_i should be connected by an edge whose weight w is chosen so that the sum of the weights of the two frontier vertices less w is equal to the constant D. This ensures that the graphs G_i become steadily smaller as i increases. For instance, Figure 2–27 is such a special case, where every cluster has only 2 frontier vertices. Hence after three iterations it would be simplified by the DR-planner Frontier into Figure 2–29.

Defining the Subsystem Simplifier. We capture the transformations performed by the Frontier DR-planner by describing simplifier maps.

Let S_i be the solvable subgraph of G_i found at stage i, SI be the set of inner vertices of S_i, FI be the set of frontier vertices of S_i, and A be any subgraph of G_i. Then the simplifier map $T_i(A)$ is defined as follows

–$T_i(S_i) = c_i$, where the weight of c_i is equal to the geometry-dependent constant D.

–If $A \cap S_i = \emptyset$, then $T_i(A) = A$.

–If $A \cap S_i \neq \emptyset$ and $A \cap SI = \emptyset$, then the image of A under the map T_i is A minus those edges that A shares with S_i.

–If $A \cap SI \neq \emptyset$, then $T_i(A) = (V_{TA}, E_{TA})$, where V_{TA} is the set of all vertices of A that are not vertices of SI, plus the vertex c_i representing the simplification of S_i. The set of edges E_{TA} is formed as described earlier in this section.

Performance Analysis. In this section, we analyze the DR-planner Frontier with respect to the various performance measures definied in previous sections.

Claim 20 *DR-planner Frontier is a valid DR-planner with the Church-Rosser property. In addition, DR-planner Frontier finds DR-plans for the maximal solvable subgraphs of underconstrained graphs.*

Proof The proof is identical to the one used for DR-planner Condense. ∎

Before we discuss the solvability preserving property of DR-planner Frontier, we would like to consider the following example shown in Figure 2–30. All edges have weight 1, vertices as indicated. Initially, the graph $BCDEIJK$ is solvable. The graph $ABCDEFGH$ is also solvable, vertices A and B are its inner vertices, vertices C, D, E, F, G and H are its frontier vertices. After $ABCDEFGH$ has been simplified into the graph $MCDEFGH$, the new graph $MCDEIJK$ is no longer dense (edges MC, MD, ME, MH and MF have weight 2 now). However, note that according to the definition of the DR-planner Frontier simplifier map, the image of $BCDEIJK$ is not the graph $MCDEIJK$, but the graph $MCDEFGHIJK$ which is dense. This graph $MCDEFGHIJK$ is welloverconstrained, since it has density -1 and it could be made wellconstrained by say removing constraints FG and FH. Thus the image of $ABCDEFGH$ is also solvable.

In general, the following claim holds.

Claim 21 *Let A and B be solvable subgraphs such that $A \cap B \neq \emptyset$ and $A \cap B \neq v$ where v is a single vertex of weight less than geometry dependent constant D, then $A \cup B$ is solvable.*

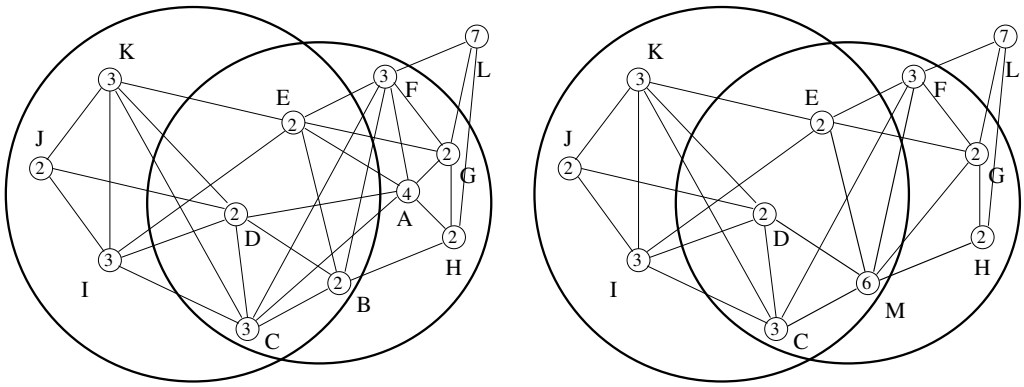


Figure 2–30: Original graph BCDEIJK is dense, new graph MCDEIJK is not

Proof Since A is solvable, the density of $A \cap B$, $d(A \cap B) \leq -D$ (a solvable graph cannot contain an overconstrained subgraph, unless it is a welloverconstrained graph in which case it can be replaced by an equivalent wellconstrained graph). Hence the density of $A \cup B$, $d(A \cup B) = d(B) + d(A \setminus B) = -D + d(A) - d(A \cap B) \geq -D$. Equality occurs when $d(A \cap B) = -D$, otherwise $A \cup B$ is overconstrained. If $A \cup B$ is overconstrained it is welloverconstrained, since it could be converted into wellconstrained by reducing weights of some edges of $B \setminus A$ or $A \setminus B$. ∎

This property can be used to show that

Claim 22 *DR-planner Frontier is solvability preserving as well as strictly solvability preserving.*

Proof Let B be a solvable graph, and suppose that the solvable graph S_i was simplified by DR-planner Frontier. Then B would only be affected by this simplification if B contains at least one internal vertex of S_i (recall that frontier vertices of S_i remain unchanged). But then, by the definition of the DR-planner Frontier simplifier, $T_i(B) = T_i(B \cup S_i)$. Since $T_i(B \cup S_i)$ is obtained by replacing S_i by solvable $T_i(S_i)$, and according to the previous claim, the union of two solvable graphs is solvable, thus $T_i(B) = T_i(B \cup S_i)$ is also solvable. ∎

Claim 23 *DR-planner Frontier is complete*

Proof This is because DR-planner Frontier (just as Condense) finds minimal dense subgraphs at each stage. ∎

Claim 24 *DR-planner Frontier has worst-choice approximation factor* $O(1/n)$.

Proof Consider Figure 2–31 - the solvable constraint graph G. Initially DR-planner Frontier will locate the minimal dense subgraph ABC (since this is the only minimal dense subgraph of G). It will not be able to locate any dense subgraphs disjoint from ABC, or including only frontier vertices of ABC. If it attempts to locate a dense subgraph that includes the entire (simplified) cluster ABC, and does so by inspecting the other vertices in the sequence $A, B, C, H, I \ldots, F, E$, (i.e going counterclockwise), then DR-planner Frontier would not encounter any dense subgraphs after ABC, until the last vertex of the sequence E is reached. The minimal dense subgraph found at this stage is the entire graph G. Thus the size of the DR-plan corresponding to this choice of vertices is proportional to n. On other hand, there is a DR-plan of constant size. This DR-plan would first locate the minimal dense subgraph $S_1 = ABC$ and simplify it. After that it would simplify $S_2 = ABCE = S_1 \cup \{E\}$, after that $S_3 = S_2 \cup \{F\}$ etc going clockwise until the vertex H is reached. At every stage i, the size of S_i is constant, hence the size of this DR-plan is constant. ∎

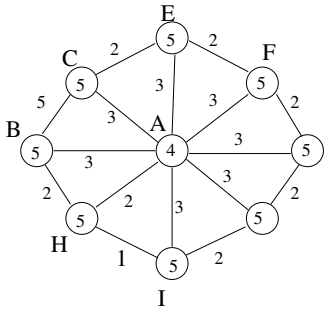


Figure 2–31: $1/n$ worst-choice approximation factor of DR-planner Frontier

Claim 25 *The best-choice approximation factor of DR-planner Frontier is at least* $\frac{1}{2}$.

Proof Let G be the weighted constraint graph. Let P be an optimal DR-plan of G, let p be the size of P (i.e the size of every cluster S_i simplified under the optimal DR-plan is less than $p+1$). We will show that there is a DR-plan P' that is "close to" P. Complete resemblance ($P = P'$) may not be possible, since the internal vertices of the cluster S_i', found by DR-planner Frontier at the i^{th} stage, are simplified into one core vertex, thereby losing some information about the structure of the graph. However we will show that there is a way of keeping the size of P' within the constant D of the size of P.

Suppose that DR-planner Frontier is able to follow the optimal DR-plan up to the stage i, i.e $S_i = S_i'$. Suppose that there is a cluster S_j in the DR-plan P such that $i < j$ and S_j contains some internal vertices of S_i. Therefore the simplification of S_j by DR-planner Frontier may be different from simplification of S_j by P. However, since the union of S_i and S_j is solvable, DR-planner Frontier could use $S_j' = T_i'(S_i) \cup S_j$ instead of S_j. The size of S_j' differs from the size of S_j by at most D units, where D is the constant depending on the geometry of the problem. Hence the size of P' is at most $p + D$, and since p is at least D, the result follows.

∎

Next, we consider the last performance measure - ability to incorporate design decomposition.

Observation 4 *DR-planner Frontier can incorporate a design decomposition of the input graph if and only if all pairs of subgraphs A and B in the given design decomposition satisfy: the vertices in $A \cap B$ are not among the internal vertices of either A or B.*

Note. This condition on the design decomposition puts no restriction on the sizes of the intersections of the subgraphs in the decomposition, and is far less restrictive than the corresponding conditions for SR and Condense. ♢

Proof The proof is similar to the case of the DR-planner SR. For the 'if' part, we find a topological ordering O of the given design decomposition P - which is a set of solvable subgraphs of the input graph G, partially ordered under the subgraph relation - such that O is embedded as a subplan of the final DR-plan generated by DR-planner Frontier; i.e, O forms a subsequence of the sequence of solvable subgraphs S_i, whose (sequential) simplification gives the DR-plan.

We take any topological ordering of the given design decomposition P and create a DR-plan for the first solvable subgraph A in P. I.e, while constructing the individual DR-plan for A, we "ignore" the rest of the graph. This individual DR-plan induces the first part of the DR-plan for the whole graph G. In particular, the last graph in this partial DR-plan is obtained by simplifying A using the simplifier described in Section 2.1.14 (and treating each A exactly as DR-planner Frontier would treat a cluster S_j found at some stage j). Let G_i be the last graph in the DR-plan for G created thus far. Next, we consider the next subgraph in the ordering O, and find an individual DR-plan for it, treating it not as a subgraph of the original graph G, but as a subgraph of the simplified graph G_i. This individual DR-plan is added on as the next part of the DR-plan of the whole graph G.

The crucial point is that the simplification of any subgraph, say A, will not affect any of the unrelated subgraphs B in P. This is because by the requirement on the decomposition P, A and B share at most frontier vertices. As a result, by the functioning of the DR-planner Frontier, when the core vertex for A is created, none of the solvable subgraphs inside B are affected.

The process - of constructing individual DR-plans for subgraphs in the decomposition P and concatenating them to the current partial DR-plan - is continued until a partial DR-plan for the input graph G has been produced, which completely includes some topological ordering of the decomposition P as a subplan.

Again, let G_k be the last graph in this partial DR-plan. The rest of the DR-plan of G is found by running the original DR-planner Frontier on G_k.

For the 'only if' part, we consider a DR-plan Q produced by DR-planner Frontier.

We first observe that the sequence of (original) solvable subgraphs whose sequential simplification gives Q can never contain two subgraphs A and B such that $A \cap B$ contains both internal vertices of A and internal vertices of B. This is because if, for instance, A is simplified before B, then B (on its own) cannot be simplified at a later stage (although $A \cup B$ could), since an internal vertex of B that is also an internal vertex of A will disappear from the graph (will be replaced by a core vertex representing everything internal to A), the moment A has been simplified.

Next, we consider the remaining case where $A \cap B$ contains some internal vertices of A but only frontier vertices of B. In this case, potentially B could be simplified before A, and A will not be affected, since the frontier vertices of B are unchanged by the simplification. However, since a given design decomposition P could contain an arbitrary number of overlapping subgraphs, we can choose decompositions P such that all topological orderings of P are infeasible i.e, no DR-plan can incorporate them as a subsequence. For instance, if there are 3 subgraphs A, B and C in P such that $A \cap B$ contains only frontier vertices of B but some internal vertices of A; $B \cap C$ contains only frontier vertices of C, but some internal vertices of B; and $C \cap A$ contains only frontier vertices of A, but some internal vertices of C. This forces the DR-plan to simplify B before A, C before B and A before C, which is impossible. ∎

2.2 Relating Problems of Chapter 1 to some Measures of Chapter 2

DR-planning problem of geometric constraint solving community directly corresponds to our RD-dag problem. Underlying weighted graph G is the same for both problems, number of geometric degrees of freedom $D = -K$, density function

$d()$ is the same etc. Stably dense graphs correspond to solvable graphs/systems. Solving optimal RD-dag problem is equivalent to finding optimal DR solution sequence.

Maximal stably dense subgraphs correspond to the roots of tree in DR-plans. Clearly size of the optimal DR solution sequence is greater or equal to the size of the minimum (stably) dense subgraph of G, and to the size of the minimum (stably) dense subgraph of $T^1(G)$ and and to the size of the minimum (stably) dense subgraph of $T^2(G)$ etc. Hence an efficient solution of minimum dense subgraph problem would be helpful for finding optimal DR solution sequence. Since it desirable that every S_i were cluster minimal (see Section 1.1) a minimal dense subgraph problem is also important for obtaining optimal DR solution sequence.

CHAPTER 3
MAXIMAL, MAXIMUM AND MINIMAL STABLY DENSE PROBLEMS

3.1 Finding Maximum Dense Subgraph

Unbounded case.

Claim 26 *Problem of finding maximum dense subgraphs when weights are allowed to be unbounded is NP-Complete*

Proof By straightforward reduction from CLIQUE. ∎

Bounded case. Suppose that the weight of all vertices is 3, of all edges 1 (it important that weights be bounded, because unbounded maximum dense subgraph problem is NP-complete). Constant $K = 0$.

Consider a following LP:

$$\max \Sigma x_i$$

s.t

$$\Sigma y_{ij} - 3\Sigma x_i \geq 0$$

$$x_i \geq y_{ij}, \forall i, j$$

$$0 \leq x_i, y_{ij} \leq 1, \forall i, j$$

Lemma 4 *Let $S(G)$ be a solution of the LP above, for a given graph G. Then vertices v_i s.t $x_i = 1$ form the largest dense subgraph A of G (dimension dependent constant $K = 0$). (If there are more than 1 dense subgraphs of size $|A|$ then the one that has highest density is formed).*

Proof Let A be the (densest) maximum dense subgraph. Let B be a subgraph constructed as described above. Suppose that $A \neq B$. Consider an x_p that has the smallest positive value in B. If $x_p = 1$ then $B = A$, contradiction. Suppose that $0 < x_p < 1$. Consider edges y_{pj} such that $0 < y_{pj} = x_p < 1$. If there are 3 or more such edges than both x_p and y_{pj} could be increased by some positive δ, resulting in a better LP solution than B, contradiction. Therefore there are at most 2 such edges. Remove x_p and its adjacent edges from B and set some vertex $v_l \in A \setminus B$ equal to x_p. This creates another optimal LP solution B'. Consider an x'_p that has the smallest positive value in B' and repeat. Eventually the value of x'_p will reach 1 and by construction $|A \setminus B| = |B \setminus A|$ and density of A is equal to the density of B. ∎

Note that since maximum dense subgraph is also maximal, technique above locates maximal dense subgraph as well.

3.2 Finding a Stably Dense Subgraph

3.2.1 Distributing an Edge

In Hoffmann et al. (1997) we have described an algorithm *Distribute*(e_i, A, K) that "distributes" weight of an edge e_i to its endpoints (v^i_R, v^i_L) (or into graph A), assuming that all other edges in A has already been "distributed". Formally edges of A, $\{e_j\} = A$ are distributed if there is a mapping (distribution) of edge-weights $w(e^j)$ into two parts f^j_R, f^j_L (corresponding to the endpoints of the edge e_j) such that **(**)**

$\forall e_j \in A$ both f^j_R and f^j_L are non-negative integers and

$$f_R^j + f_L^j = w(e^j)$$

$$\sum_{j|v_{R,L}^j == t} f_{R,L}^j \leq w(v_t), 1 \leq t \leq n$$

i.e for any vertex $v_t \in A$ distribution from all of it adjacent edges does not exceed weight of v_t.

Distributing new edge e_i into a graph A edges $\{e_j\}$ has already been distributed requires finding a distribution of edges $\{e_j\} \cup e_i$ such that all conditions **(**)** hold for $A \cup e_i$, except that the second condition for edge e_i is more strict **(***)**

$$f_R^i + f_L^i = w(e^i) - K + 1$$

We will give examples of distribution for $K = 0$ and below, Hoffmann et al. (1997) contains relevant implementation details that involve finding network flows in bipartite graphs.

In the examples below all edges have weight 1, all vertices have weight 2.

In Figure 3–1, current graph $A = \{AE, AB, AC, BE, BC, EC, CD\}$, these edges has already been distributed, distribution is indicated by the arrows (so $f_{AE}^A = 1, f_{AE}^E = 0, f_{BC}^C = 1, f_{BC}^B = 0, \dots$). Currently we want to distribute edge ED, but for ED we should use condition **(***)** , i.e we have to distribute $1 - 0 + 1 = 2$ units. We can distribute one unit toward D without violating condition 3 of **(**)** , but in order to distribute second unit we need to change current distribution (since currently there is no room in either E or D). One way to rearrange the distribution

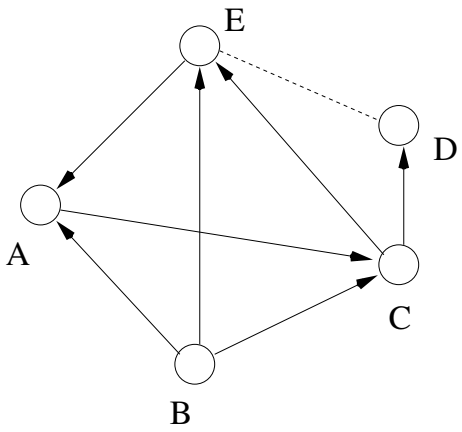


Figure 3–1: Before the distribution of edge ED

is by reversing flow in BE from E to B, this will free up one slot in E, so we can send remaining unit of ED into E. Resulting distribution is shown in Figure 3–2.

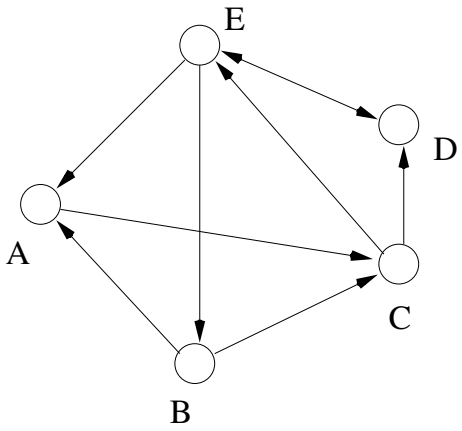


Figure 3–2: After the distribution of edge ED

Note that before we will distribute next edge, say AD, distribution of ED should comply with second condition of (**) and not (***), so instead of distributing 2 units on ED we distribute only one, hence we should remove either 1 unit going toward E or 1 unit going toward D. Resulting distribution is shown in Figure 3–3.

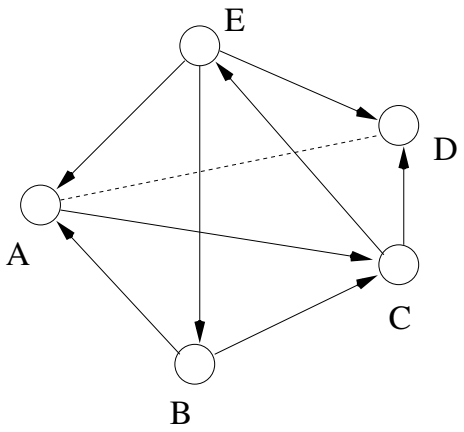


Figure 3–3: Before the distribution of edge AD

Why do we add and than remove 1 extra unit, and what is the entire purpose of algorithm *Distribute*(e_i, A, K)? Consider Figure 3–4, where edge BD is about to be distributed.

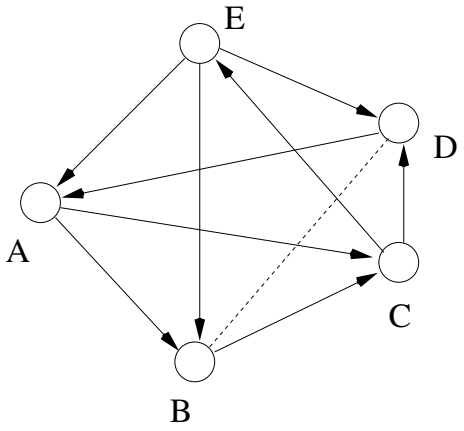


Figure 3–4: Before the distribution of edge BD

We can send 1 unit of BD to D (rerouting ED to E) but there is no room anywhere for 1 extra unit. Note also that the graph A that would result from adding edge BD is dense for $K = 0$ (it has 5 vertices of weight 2 and 10 edges of weight 1). In general inability to distribute an extra unit indicates a presence of a dense subgraph. Not also that non of the proper subgraphs of A are dense for $K = 0$, and that we were able to distribute all the edges up until now.

In general it is true if for some sequence $\{e_i\}, i \leq p$, no possible distribution of edge e_p exists (and all previous edges $e_i, i < p$ were successfully distributed) this means that there is a dense subgraph B induced by a subset of edges $\{e_i\}$ and $e_p \in B$. Conversely, if there is a dense subgraph B, and all edges of B form a subsequence of $\{e_i\}$ and the last edge of B in $\{e_i\}$ is e_p then this edge e_p cannot be distributed.

Properties of Distribute().

Property 1 *If* $Distribute(e, A, K) \neq \emptyset$*, then* $d(Distribute(e, A, K)) \geq K$.

Property 2 *Suppose that edge* $e \in B$ *and subgraph* $B \subseteq A$ *is dense. Then* $Distribute(e, A, K) \neq \emptyset$

Property 3 *Suppose that edge* $e \in B$ *and subgraph* $B \subseteq A$ *is unique dense subgraph of* A *containing* e*. Then* $Distribute(e, A, K) = B$

Property 4 *If* $Distribute(e, A, K) \neq \emptyset$ *then at least one endpoint ofe* $\in$ $Distribute(e, A, K)$.

Property 5 *Let* e *be an edge of a graph* A*. Suppose that graph* $A \setminus e$ *does not contain any overconstrained subgraphs. Let* $B = Distribute(e, A, K)$ *and* $B \neq \emptyset$ *then* B *is stably dense.*

Proof By Property 1 $d(B) \geq K$. Suppose that B is not stably dense, i.e $d(B) \geq K$ and there is $C \subset B$ such that $d(C) > K$ and when C is replaced in B by subgraph $C', d(C') = K$ then resulting graph B' is not dense, $d(B') < K$. Therefore $d(B \setminus C) < 0$. Also $e \in C$ (otherwise $C \subseteq A \setminus e$, contradicting condition that $A \setminus e$ does not contain any overconstrained subgraphs). Since $d(B \setminus C) < 0$, there is a vertex $v \in B \setminus C$ such that sum of flows to v is less than weight of v (for example in Figure 3–6, vertex $v = G$ and edge $e = BD$). However this vertex v would not be labeled by $Distribute(e, A, K)$, because if v were labeled then there would be a sequence of flows from v to e and this sequence could be reversed therefore successfully distributing edge e. This contradicts the

assumption that e could not be distributed. Hence $v \notin Distribute(e, A, K)$ and therefore $Distribute(e, A, K) \subset B$, contradicting definition of B. Hence our original assumption that B is not stably dense was wrong. ∎

Note that the condition that A does not contain any overconstrained subgraphs is necessary, consider for example Figure 3–5 depicting points and distances in 2D. Let $e = HB, A = BCDEF$ then $HBCDEF = Distribute(e, A, K)$ but $HBCDEF$ is not stably dense, only dense.

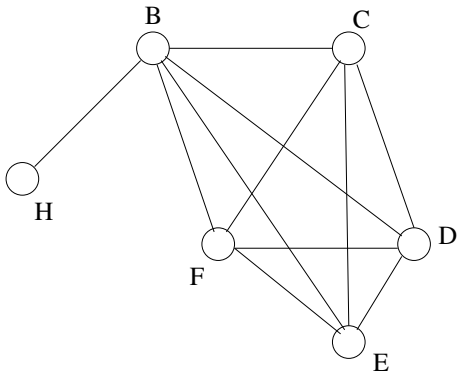


Figure 3–5: Locating dense graph instead of stably dense

3.2.2 Finding Dense and Stably Dense Subgraph

Algorithm *Distribute(e,A)* also provides a mechanism for determining a dense subgraph B that contains undistributable edge $e \in B$. We will illustrate this algorithm on an example Figure 3–6.

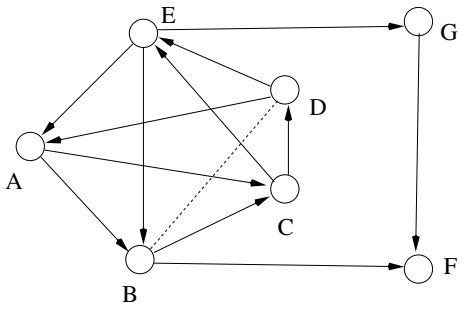


Figure 3–6: Before the distribution of edge BD

Suppose that all edges except BD has already been distributed, arrows indicating actual distributions. While attempting to distribute 2 units from edge BD we will first label endpoints of BD. I.e labeled set $L = \{B, D\}$. We will place one unit of BD into D. Then we will try to find a place for the one remaining unit of ED. Using standard augmenting path techniques we will label and examine all such vertices v, such that there is a flow distributed from v to at least one currently labeled vertex. Vertices A, E and C fit the description of v. Since none of them has room for one unit of ED we update $L = \{A, B, C, D, E\}$. Since there are no more vertices v such that there is flow from v to some vertex in L, algorithm outputs a found dense subgraph L and terminates.

A following is pseudocode description of the algorithm just demonstrated.

Algorithm Dense(G)

1. $A = \emptyset, B = \emptyset$
2. **for every** vertex v **do**
3. **for every** edge e incident to v and to C **do**
4. B=Distribute(e,A,K)
5. **if** $B \neq \emptyset$
6. **goto** Step 11
7. **endif**
8. **endfor**
9. add vertex v to A
10. **endfor**
11. **return** B

Properties of Algorithm Dense().

Property 6 *Algorithm Dense() finds a stable dense subgraph.*

Proof Follows from Property 5. ∎

Property 7 *Let G be a dense graph. Then Dense$(G) \neq \emptyset$.*

3.2.3 PushOutside()

Note that Property 5 only guarantees that some stably dense subgraph containing currently distributed edge is found. In general we are interested in finding maximal stably dense subgraphs, which may not be found by Distribute. Consider for example Figure 3–7 (constant $K = 0$). Suppose that algorithm *Dense*() distributed vertices A,B,C and D in that order. Then *Dense*() will return BCD instead of ABCD, because flows on edges AC and AB will be directed toward vertex A, hence A will never be labeled.

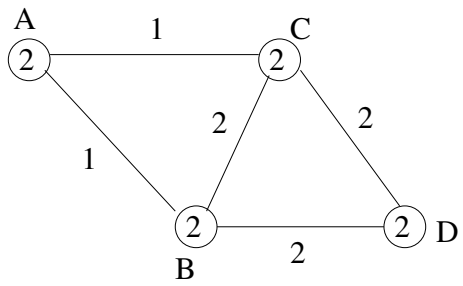


Figure 3–7: Dense graph BCD found instead of maximal dense ABCD

Assume that graph $A \setminus e$ does not contain any overconstrained subgraphs and suppose that algorithm $Distribute(e, A, K)$ has returned stably dense subgraph B. We would like to locate stably dense $C \subseteq A, e \in C$ such that there is no stably dense $D \subseteq A$ such that $C \subset D$. In other words C is stably dense dense subgraph of A that contains edge e. To construct C we will start with $B = C$ and we will check whether C can be enlarged by using technique of *Pushoutside(B,A)*.

Algorithm Pushoutside(B,e,A)

1. $C = B$
2. **for every** vertex $v \in C$ **do** label(v)=1 **end for**
3. **for every** edge e adjacent to C **do** $C = C \cup Distribute(e, A, K)$ **end for**
4. **return** C

Property 8 *Assume that graph $A \setminus e$ does not contain any overconstrained subgraphs. Let B be a stably dense subgraph returned by $Distribute(e, A, K)$. In 2D case if $|B| \geq 2$ then $C = Pushoutside(B, e, A)$ is a maximal stably dense subgraph of G that contains e. Similar result holds in 3D if $|B| \geq 3$.*

Proof For all maximal stably dense $C \subseteq G$ such that $B \subset C$ there is an edge $e \in C \setminus B$. Execution of $Distribute(e, A, K)$ will locate such C that will be enlarged accordingly. ∎

For a 3D case when found stably dense subgraph B is an edge there might be several maximal stably dense subgraphs of A containing B, as in Figure 3–8. We can use following algorithm in order to find all of them.

Algorithm Pushoutsidedge(B,e,A)

1. $i = 1$
2. $C_i = B, C_0 = \emptyset$
3. **while** $C_{i-1} \neq B$
4. **for every** vertex $v \in C$ **do** label(v)=1 **end for**
5. **for every** edge e adjacent to C_i **do** Distribute(e, A, K)
6. **if** C_i is not dense **then** i
7. **end for**
6. **return** C_i
7. **end while**

Properties of Pushoutsideedge().

Property 9 *Assume that graph $A \setminus e$ does not contain any overconstrained subgraphs. Let B be a stably dense subgraph returned by $Distribute(e, A, K)$ in 3D case. If B is an edge $= e$ then $Pushoutsideedge(B, e, A)$ returns all maximal stably dense subgraphs of A that contain e.*

Finding Minimal Dense Subgraphs. Note that Property 2 does not guarantee that subgraph found is minimal dense. Consider for example Figure 3–9 (constant $K = 0$). If algorithm $Dense()$ distributed edges in order AC,AB and

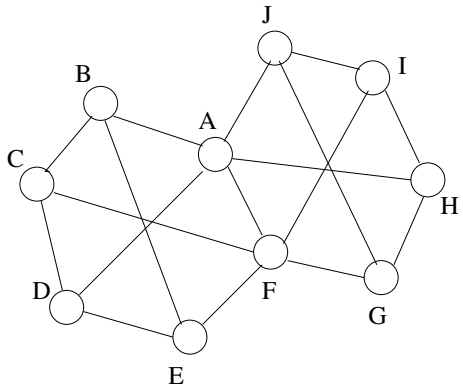


Figure 3–8: Graphs ABCDEF and FGHIJA are maximal stably dense in 3D

BC then nonminimal dense graph ABC would be found instead of minimal dense subgraph BC.

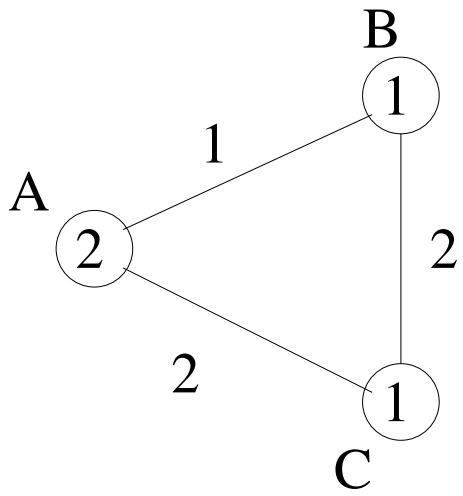


Figure 3–9: Dense graph ABC found instead of minimal dense BC

We will fix this problem using technique below. Once a dense subgraph B has been found the following algorithm *Minimal(B)* can be used to locate minimal dense subgraph C of B.

Algorithm Minimal(B)

1. $A = B$
2. **for every** vertex $v \in A$ **do** label(v)=0
3. **for every** unlabeled vertex $v \in A$ **do**

4. $C = A \setminus v$

5. D= Dense(C)

6. **if** $D \neq \emptyset$ **then** $A = D$

7. $label(v) = 1$

8. **endfor**

9.**return** A

Eventually out graphs will have clusters as components so an appropriate modification is needed.

begin findMinimal(S,C,F)

$A = S$

for all clusters S_j of S

$A = A \setminus S_j$

$X = distributeCluster(C, A)$

if $X \neq \emptyset$ **then** $A = X$

else $A = A \cup S_j$

end if

end for

return A

end findMinimal

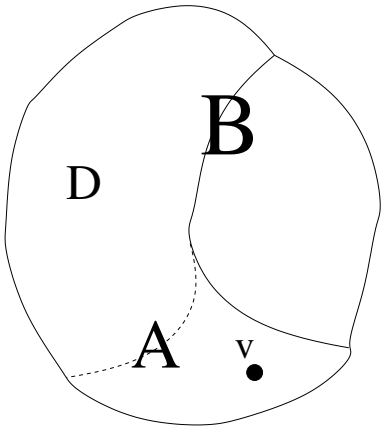


Figure 3–10: Actions of Minimal() algorithm

Properties of Minimal().

Property 10 *Let $B = Dense(A)$ and $C = Minimal(B)$, then C is minimal stably dense.*

More efficient Algorithm for Minimal(). It is possible to use minimum cut techniques of Karger (1996) in order to find a minimal subgraph.

Input: a dense graph G, last vertex found v, distributed flows $f(e, u)$ and undistributed flows $g(e)$.

Output: a weighted graph G' and a vertex $v' \in G'$. The weight of the edge $e' \in G$ is equal to the flow in corresponding edge $e \in G$ toward v + $g(e)$.

Goal: find a cut in G' separating v' from at least one vertex, such that the weight of the cut is at most K and if L' is the part of G' containing v' than there is no $M' \subset L', v' \in M'$ such that the cut separating M' from G has weight less than K.

Claim 27 *The graph $L \subseteq G$ corresponding to the L' is minimal dense.*

Proof Let A be the the total amount of undistributed flows on the edges from v' to L', B be the total amount of undistributed flows on the edges from v' to G'/L' and C be the weight of the cut edges such that none of the endpoints is v'. Then $A + B = K$ (by definition of K) and $B + C < K$ (since weight of the cut was less than K). Thus $A > C$ so the $d(L') > 0$. Thus L' is dense and minimal because of a minimality condition on the cut. ∎

Algorithm for finding a minimal cut of weight less than K. 1. Run the contraction algorithm $\alpha = 1 + \epsilon, \epsilon = 1/\log n$

2. Take the smallest one containing v' of . Any other cut on the side of v' will have size at least $(1 + 1/\log n)m$ where m is the size of the minimum cut. Contract all the vertices on the side not containing v' into one vertex, run contraction algorithm again until the size of the second best cut is $>$ K. The number of iterations is $O((\log n)(\log K))$, and the cuts are found with high probability.

3.3 Structure and Properties of Frontier algorithm

Note 1 *We have described DR-planner Frontier in Chapter 1, but we did not specify an actual algorithm. Now we will describe an actual implementation. Also DR-planner Frontier was designing DR-plan, while Frontier Algorithm will construct complete recursive decomposition, which is a stronger than just a DR-plan because it involves finding all maximal stably dense subgraphs, not just some set as in DR-plans.*

In previous sections we have described subroutines of Frontier Algorithm for finding maximal stably dense subgraph that contain a given edge, as well as transformation replacing this subgraph once it has been found. Now we will describe the high level structure of Frontier Algorithm and later will demonstrate its properties.

3.3.1 Informal Description of Frontier Algorithm

Frontier Algorithm will proceed by adding edges of graph G one by one to the initially empty subgraph A and *distributing* these edges as was described in Section 3.2.1. If we were able to distribute all the edges of G then G does not contain any dense subgraph, and algorithm terminates. If we have encountered an edge that we are unable to distribute then this implies that A contains a dense subgraph B (see algorithm *Dense* of Section 3.2.1). Once we have located such a dense subgraph B we will replace graph B by another graph B'. In order for Frontier Algorithm to correctly locate all maximal dense subgraphs this replacement B' should satisfy two criteria. One criteria is that this replacement will allow us to continue the distribution process. This criteria calls for removal of edges that currently cannot be distributed, i.e the edges that cause overconstrainedness of B. Another criteria is that this replacement B' should not destroy well-constrainedness of any maximal stably dense subgraph, otherwise we might never identify it later as a stably dense subgraph.

3.3.2 Assumptions

While Frontier Algorithm can be applied to arbitrary edge and vertex weighted graph, we will for sake of simplicity restrict analysis of its properties to cases of points and distances in 2D or 3D. I.e dimension dependent constant $K = -3$ or -6, weights of all edges are 1, and weights of all vertices are either 2 or 3 respectively.

3.3.3 Joining Pairs of Clusters

The previous disclaimer allows to abotain following results that limit the number of maximal stably dense graphs.

Claim 28 *For any two distinct stably dense graphs A, B, in 2D, if $|A \cap B| > 1$ then $A \cap B$ is stably dense graph. Similarly in 3D, if $|A \cap B| > 2$ then $A \cap B$ is stably dense graph. Also in 3D if $A \cap B = \{\{v\}, \{u\}\}$ then $A \cup B$ is stably dense.*

Proof Let $C = A \cap B$. In 2D, if $|C| > 1$ then either $d(C) \leq K = -3$ or C can be replaced by $C', d(C') = K$ without violating solvability of A or B. Therefore $d(A \cup B) = d(A) + d(B) - d(C) \geq K + K - K \geq K$. Similar proof holds for 3D case, when $K = -6$. ∎

3.3.4 Relevant Transformation Notation

Frontier Algorithm will begin with a graph $G_1 = G$ and will change it into $G_2, G_3, \ldots, G_k$. Stably Dense graph or *cluster* found during i^{th} stage of Frontier Algorithm will be denoted $S_i, S_i \subseteq G_i$. This cluster's transformation by Frontier Algorithm that was described above will be denoted $T_i(S_i)$. In general $T_i : A \subseteq G_i \to B \subseteq G_{i+1}$. For a graph $A, A \subseteq G_i, |A \cap S_i| \leq 1$ the mapping is $T_i(A) = A$. For a graph $A, A \subseteq G_i, |A \cap S_i| \geq 2$ the mapping is $T_i(A) = (A \setminus S_i) \cup T(S_i)$. For $A \subseteq G$ we will denote $T^i(A) = T_{i-1}(T_{i-2}(\ldots(T_1(A))))$. Let $X \subset G_i$, an *underlying subgraph* $T^{-1}(X) \subset G$ is a subgraph in G induced by vertices that ended up in X at stage i.

3.3.5 Recombination by Frontier Algorithm

Figure 3–11 illustrate transformation by Frontier Algorithm. Left half represent a graph G before transformation. All vertices have weight 2, all edges have

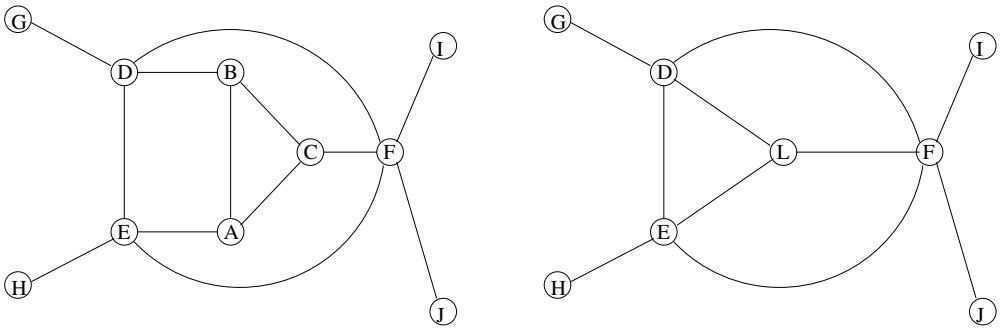


Figure 3–11: Transformation by Frontier Algorithm

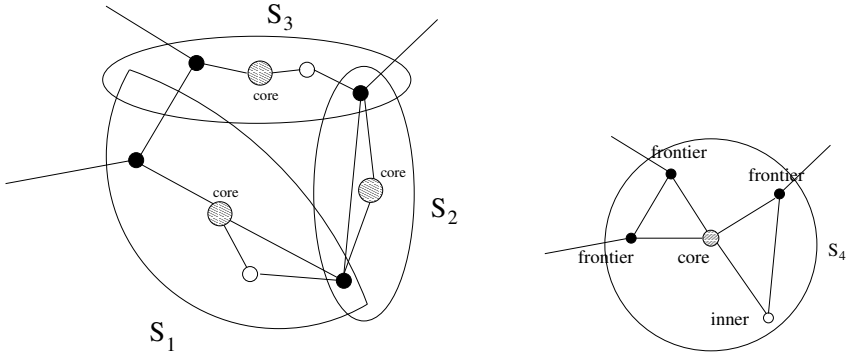


Figure 3–12: General transformation by Frontier Algorithm

weight 1, geometry-dependent constant $K = -3$. Suppose that Frontier Algorithm located a stably dense graph $S = ABCDEF$. It will recombine this graph into new graph S' as follows. All *frontier* vertices of S that are connected to vertices outside of S remain unchanged. I.e vertices D,E and F remain unchanged. All edges between frontier vertices remain unchanged. All non-frontier or *inner* vertices of S are combined into one new *core* vertex (i.e vertex L). The weight of edges connecting core vertex L to a frontier vertex, say vertex D, is equal to the sum of all edges between D and inner vertices, i.e 1 in our example. The weight of core vertex is chosen so that overall density of $S' = K = -3$, i.e weight of L is equal to 3.

More generally, in Figure 3–12 dark nodes represent frontier vertices, unfilled nodes represent inner vertices and dashed vertices represent core vertices of previously found stably dense subgraphs $S_1, \ldots, S_3$. Frontier Algorithm will

transform a stably dense union of $S_1, \ldots, S_3$ into one new subgraph on the right, using rules above.

This replacement satisfies first criteria above, since B' has density exactly K and second criteria as demonstrated in Property of Frontier Algorithm 15 below.

3.3.6 Removing Undistributed Edges

Sometimes while executing line 6 of Frontier Algorithm we may find a new stably dense graph while there are still some edges of v remain undistributed. For example, consider Figure 3–13. Suppose that C is the last vertex to be distributed, and 3 edges remain to be distributed - CE, CF and CA. Once a CE is distributed a cluster CDEF is discovered. Now all undistributed edges from C to other vertices of this cluster (i.e CF) will never be distributed (they just make this cluster overconstrained, and under definition of stably dense graphs they can be removed. Their removal will not affect finding maximal stably dense subgraphs, as stated in Property 12). All other undistributed edges (i.e CA) will now be distributed, so as to enable us to discover cluster ABC for example.

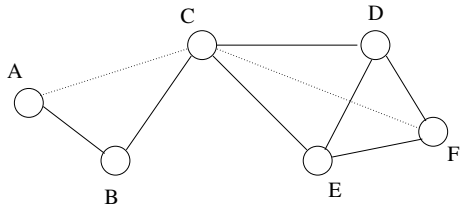


Figure 3–13: Edge CF will not be distributed, edge AC will

3.3.7 Pseudocode

Frontier Algorithm

1. $F = \emptyset$
2. $ClustersInF = \emptyset$
3. $VList = \{\{1\}, \{2\}, \ldots, \{n\}\}$
4. **while** $VList \neq \emptyset$
5. $\quad v = VList.remove(), flag = 0$

6. **for** edges e adjacent to v and F (see Section 3.3.6)

8. $S = distributeEdge(e, F, K)$ (see Section 3.2)

9. $S = checkForIntersections(S, F)$ (see Claim 28)

10. $S = pushOutside(S, F)$ (see Section 3.2.3)

11. **if** $S \neq \emptyset$

12. **then** $flag = 1, S' = simplify(S, F)$ (see Section 3.3.5,Section 3.3.6)

13. $ClustersinF.add(S')$

15. **end for** $F = F \cup v$

16. **end while**

end Frontier Algorithm

1. **begin checkForIntersections(S,F)**
2. **for** all clusters $S_i \in VList()$
3. $A_i = S \cap S_i$
4. **if** $|A_i| \geq 2$ (in 2D, or ≥ 3 in 3D, or A_i is a pair of vertices in 3D)
5. **then** $S = S \cup S_i$
6. $VList.remove(S_i)$
7. **end if**
8. **end for**
9. **end checkForIntersections(S,F)**

3.3.8 Example of actions by Frontier algorithm

We will demonstrate how Frontier algorithm generates a complete recursive decompostion for an example constraint graph G by describing various stages of simplification of the Figure 3–14, Figures 3–15 to 3–17 resulting in a complete recursive decomposition of Figure 3–18.

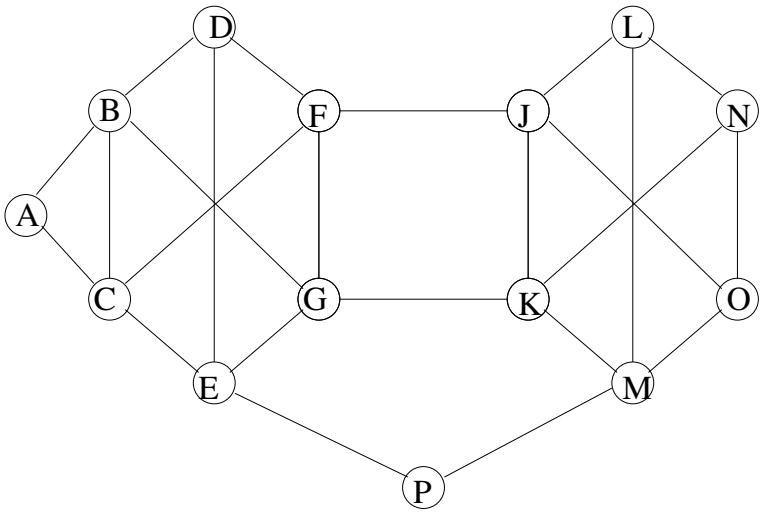


Figure 3–14: Initial graph

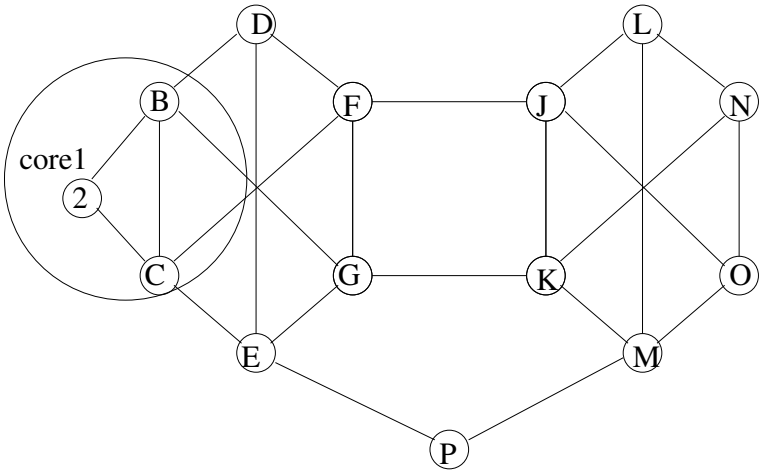


Figure 3–15: After A-C have been distributed

The weight of all vertices in $G = G_1$ is equal to 2, the weight of all the edges is equal to 1, the constant K is equal to -3. Vertices are distributed in alphabetic order. Circles indicate found nontrivial clusters.

3.3.9 Modifying Frontier Algorithm for Minimality

Here we will show how to generalize minimality-forcing technique of Section 3.2.3, so it would guarantee cluster minimality property. Let S be the stably dense subgraph found by Frontier Algorithm. Recall that a sequence $\emptyset = S_0 \subset S_1 \subset \ldots S_k = S$ has a cluster minimality if there is no proper dense

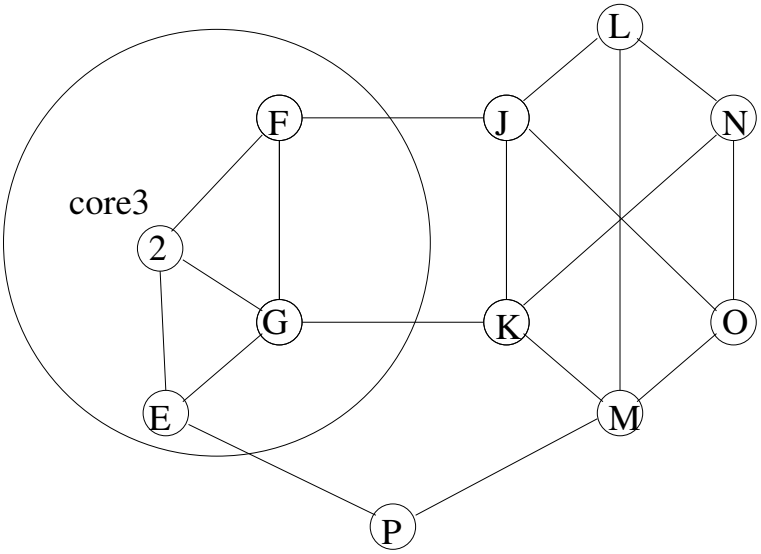


Figure 3–16: After A-G have been distributed

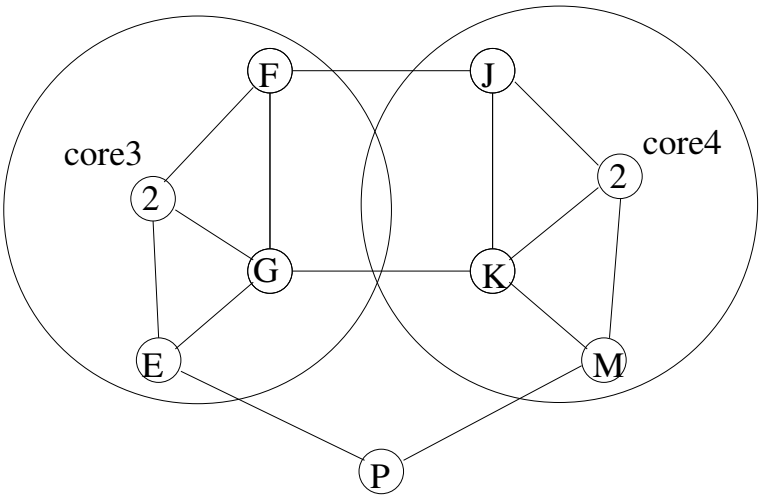


Figure 3–17: After A-P have been distributed

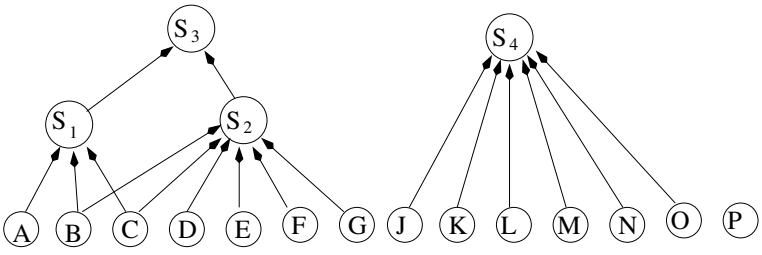


Figure 3–18: Corresponding complete recursive decomposition

subgraph $X \subset S_j$ such that $S_{j-1} \subset X$. Consider for example Figure 3–20. The following sequence of dense subgraphs $S_1 = \{A, B, C, D, E, F\}, S_2 = \{A, G, H, I, J, K\}, S_3 = \{K, L, M, N, O, F\}, S_4 = S_1 \cup S_2 \cup S_3 \cup \{P\}$ is not cluster minimal, since $X = S_1 \cup S_2 \cup S_3$ is dense and $S_1 \subset X \subset S_4$.

In order to construct cluster-minimal sequences we will modify Frontier Algorithm, forcing every cluster S_j to contain cluster S_{j-1} that was found during the preceeding stage $j-1$. In other words S_j is an "extension" of S_{j-1}, and we apply *PushOutside()* technique to assure that this extension is minimal.

FindSequence(SC,SI,M)

Idea of an algorithm. We will be using algorithm $FindSequence(SC, SI, X)$. Graph SC is the current dense subgraph that we want to decompose. Graph $SI \subseteq SC$ is assumed to have been decomposed already and a current step S_j in decomposition sequence of SC is required to take SI as one of component clusters. Graph $X \subset SC \setminus SI$ is a subset of vertices any possible S_j has to contain. Initially $SC = S, SI = \emptyset, X = \emptyset$ (X can also be taken to be the last vertex v distributed when S was found, since by Property 4 vertex v is contained in every dense subgraph of S). Figure 3–19 shows recursive applications of $FindSequence$.

Pseudocode

1. **While** $M \cup SI$ doesn't contain all the vertices in SC
2. **do**
3. Let w be a vertex not in M
4. **While** $S \setminus \{w\}$ contains no dense subgraph that includes SI
5. (successful pushoutside SI and distribution of all other edges in $S \setminus SI$).
6. **do**
7. add w to M
8. **end do**

9. Let SN be the dense subgraph found (that includes $McupSI$ and excludes w)

10. Let $FSN = Frontier(SN)$ = Frontier simplification of SN

11. Let $FSC = SC$ with SN replaced by FSN

12. **Return** concatenate($Findsequence(SN, SI, M), Findsequence(FSC, FSN, \emptyset)$);

13. **end do**

14. **Return** the list [SI, SC]

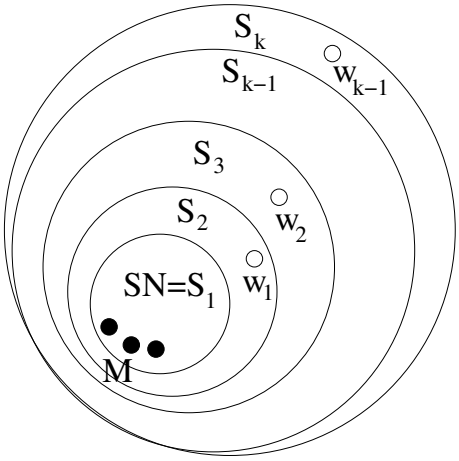


Figure 3–19: Enforcing cluster minimality

Example of Actions by Modified Frontier Algorithm. Consider Figure 3–20, where $S = \{A, \ldots, Q\}$ is the stably dense subgraph and $SI = M = \emptyset$. Suppose vertices $\{A, \ldots, F\}$ were distributed and minimal dense subgraph $S_1 = \{A, \ldots, F\}$ was discovered and simplified as usual by Frontier Algorithm (if graph is minimal dense then it decomposition sequence contains only one step and we dont have to worry cluster minimality property). Next all remaining vertices except K and P were distributed - i.e. no clusters other than S_1 could be found so far. Finally vertex P was distributed and after that vertex K was distributed. Now dense graph $SC = \{A, \ldots, Q\}$ has been found, $SI = S_1 = \{A, \ldots, F\}$ and $X = \{K\}$. Application of $PushOutside(SI, SC)$ will return $S_2 = \{A, \ldots, O\}$. Subsequent steps will locate $S_3 = \{A, \ldots, P\}$ and $S_4 = \{A, \ldots, Q\}$.

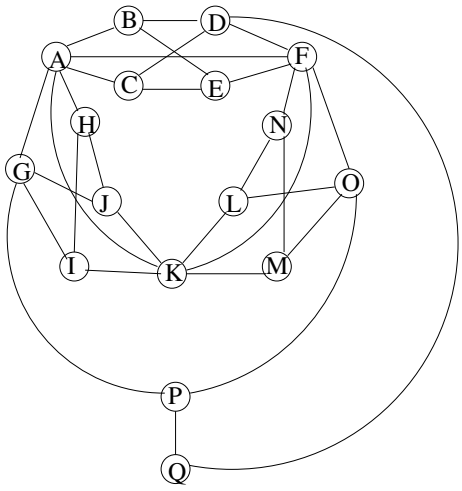


Figure 3–20: Example of finding minimal

Property 11 *Frontier Algorithm modified as described above has cluster minimality property.*

3.3.10 Complete Decomposition Property

Theorem 1 *Given a graph G, Frontier Algorithm solves recursive decomposition problem for G.*

Proof We already know From Section 2.1.14 that DR-planner Frontier constructs a DR-plan (=recursive decompostion). Using Properties 15, 14 and 11 we can see that it in fact constructs complete recursive decompostion since it finds all maximal stably dense subgraphs. Hence Frontier algorithm solves recursive decomposition problem. ∎

3.4 Maximum Stably Dense Property

Claim 29 *Since Frontier Algorithm locates all maximal stably dense subgraphs it can be used to find maximum stably dense subgraph as well. However the running time is only guranteed to be polynomial if weights of vertices and edges are bounded.*

3.5 Other Properties of Frontier Algorithm

Property 12 *For every i and every maximal stably dense subgraph $A \subseteq G_i$ the image $T_i(A)$ is also stably dense subgraph (of G_{i+1}).*

Proof In 2D, if $|S_i \cap A| \leq 1$ then $T_i(A) = A$, otherwise $T_i(A) = (A \setminus (S_i \cap A)) \cup T_i(S_i)$ which is stably dense by Claim 28. In 3D we might have a slight complication when we remove some of the undistributed edges (Section 3.3.6). Consider for example Figure 3–21. Suppose that vertex A is the last one to be distributed. Suppose that there only edges AE and AF remaining and AE is distributed before AF. Then cluster ABCDEF will be discovered and edge AF will never be distributed. It seems that graph AFGHIJ that used to be stably dense now become unstably dense. However note that the image $T^i(AFGHIJ) = T^i(ABCDEFGHIJ)$ that still remains stably dense, so maximal graph still remains stably dense. This also true in general case since $d(T^i(S_i)) = K = -6$ is equal to the density of a pair of vertices 3 units each. ∎

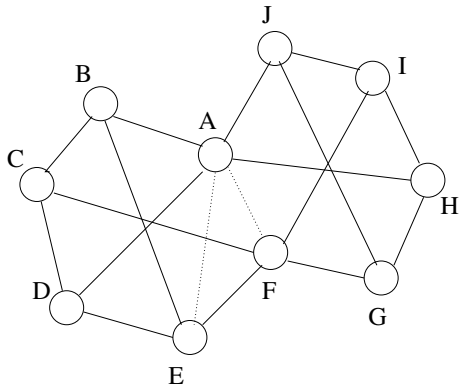


Figure 3–21: Removing AF does not affect solvability of entire graph

Property 13 *For every stably dense subgraph $A \subseteq G_i$, the graph $T^{-1}(A) \subseteq G_{i-1}$ is also stably dense.*

Proof If $|T^{-1}(A) \cap S_{i-1}| \leq 1$ then $T^{-1}(A) = A$ and result follows. Suppose that $|T^{-1}(A) \cap S_{i-1}| > 1$ and suppose that $B = T^{-1}(A)$ is not stably dense, i.e $d(B) \geq K$ and there is $C \subset B$ such that $d(C) > K$ and when C is replaced in B by subgraph $C', d(C') = K$ then resulting graph B' is not dense, $d(B') < K$. Consider $C \cap T^{-1}(c_i)$, where c_i is a core vertex of cluster S_i simplified at stage i. If $C \cap T^{-1}(c_i) = \emptyset$ then $T^i(C) = C$, contradicting the fact that A is stably dense. If

$C \cap T^{-1}(c_i) \neq \emptyset$ then we can replace $C \cap T^{-1}(c_i)$ by c_i and resulting graph (A) will be stably dense, hence (using definition of stably dense graph) graph $T^{-1}(A)$ is also stably dense, contradiction. ∎

Property 14 *For every cluster S_i chosen by Frontier Algorithm, subgraph $T_0^{-1}(S_i) \subseteq G$ is stably dense.*

Proof Property 5 guarantees that S_i is stably dense and then applying Property 13 i times insures that $T_0^{-1}(S_i) \subseteq G$ is stably dense as well. ∎

Property 15 *For every maximal stably dense subgraph $A \subseteq G$, Frontier Algorithm will find cluster S_i such that $A = T^{-1}(S_i)$.*

Proof Due to the Property 12 $T^i(A)$ is stably dense for every i. At some stage j the last undistributed edge $e \in T^i(A)$ will be distributed. Because of the Property 8 and Property 9 cluster S_j such that $A = T^{-1}(S_j)$ will be discovered. ∎

CHAPTER 4
MINIMUM STABLY DENSE SUBGRAPH PROBLEM

4.1 Relation of Minimum Stably Dense to other Problems

4.1.1 Maximum Number of Edges Problem

A *maximum number of edges (optimization) problem* $P1(G,k)$ involves finding a subgraph $A \subseteq G$ such that number of vertices of A, $|V(A)| = k$ and number of edges of $A, |E(A)|$ is maximized. In other words the problem is to find subgraph of a given size that has largest number of edges (in some papers Kortsarz and Peleg (1993) this problem is also referred to as *k-dense problem*. However throughout this paper we are using different meaning of term *dense*, as defined below).

A corresponding *decision version of maximum number of edges problem* $AD1$ is this: given (G,k,p), is there $A \subseteq G, |A| = k, \frac{|E(A)|}{|A|} \geq p$?

Claim 30 *$AD1$ is NP-complete*

Proof By reduction from k-CLIQUE. Let $Y = (G,k)$ be an instance of k-CLIQUE, take $p = (k-1)/2, G' = G, k' = k$ then $Z = (G', k', p)$ is in $AD1$ iff Y is in k-CLIQUE. ∎

A *bounded decision version of maximum number of edges problem* $ADB1$ is this: given (G,k,p), where $p \leq const$, is there $A \subseteq G, |A| = k, \frac{|E(A)|}{|A|} \geq p$?

Claim 31 *$ADB1$ is NP-complete.*

Proof By reduction from k-CLIQUE problem, using the same construction of new graph G' as in Theorem 2 of Section 4.2, finding the subgraph $A' \subseteq G', |A'| = ((k-1)/4 - 1)k(k-1) + k(k-1)/4$ that has largest number of edges among all subgraphs of this size and finally checking whether the corresponding subgraph A in the original graph G is a clique of size k. Note that $p \leq 3$ for A'. ∎

4.1.2 Decision Version of Minimum Dense Subgraph

A *minimum dense subgraph optimization problem* $P2(G, X)$ for a given constant X and edge/vertex-weighted graph G, involves finding a smallest subgraph $A \subseteq G$ that is *dense*, i.e

$$\sum_{e \in E(A)} w(e) - \sum_{v \in V(A)} w(v) \geq X$$

A corresponding *decision version of minimum dense subgraph problem* $AD2$ is this: given (G, X, p) is there a dense $A \subseteq G, |A| \leq p$?

Claim 32 *$AD2$ is NP-complete.*

Proof See Lemma 5 in Section 4.2. ∎

A *bounded decision version of minimum dense subgraph problem* $ADB2$ is this: given (G, X, p) where $X \leq const$ is there a dense $A \subseteq G, |A| \leq p$?

Claim 33 *$ADB2$ is NP-complete.*

Proof See Theorem 2 in Section 4.2. ∎

4.1.3 Relationships between two Decision Problems

Claim 34 *Problem $P2 \leq_T P1$.*

Proof Let (G, X) be an instance of problem $P2$. Wlog assume that all weights $w(e) = 1, \forall e \in E; w(v) = c, \forall v \in V$. We will find smallest dense subgraph by examining subgraphs of size 1,2,3,... that contain largest number of edges until dense one is found. I.e take $k = 1$. Let A_k be the graph found by $P1(G, k)$. If A_k is dense then return A_k as an output of $P2(G, X)$. Otherwise set $k = k + 1$, repeat.

∎

Claim 35 *Problem $P1 \leq_T P2$.*

Proof Let (G, k) be an instance of problem $P1$. Construct another graph $G' = G \cup \{N\}$ (see Figure 4–1). Set weights of all vertices $w(v) = 0$, weights of all edges adjacent to N to k^2, weights of all other edges to 1. Let $X = k^3 + \frac{k(k-1)}{2}$.

Note that if there is a subgraph $B \subseteq G$, such that $|V(B)| = k, |E(B)| = \frac{k(k-1)}{2}$ then corresponding subgraph $B \cup \{N\} \subseteq G'$ is dense. Also for any smaller subgraph $C \subseteq G'$ such that $|V(C)| \leq k < k+1 = |V(B) \cup \{N\}|$ it holds that C is not dense, since $|E(C)| \leq (k-1)k^2 + \frac{(k-1)(k-2)}{2} < k^3 + \frac{k(k-1)}{2} = X$. Therefore $B \cup \{N\}$ is minimum dense subgraph of G'.

In general if subgraph $B \subseteq G$ of size k has largest number of edges T among all subgraphs of size k, then when $w(N) = w(v_i) = 0, w(v_i, N) = k^2, w(v_i, v_j) = 1, \forall v_i, v_j \in G$ and $X = k^3 + T$ the corresponding subgraph $B \cup \{N\} \subseteq G'$ is the smallest dense subgraph of G'.

This suggest the following algorithm for finding subgraph of G of size k that has largest number of edges. We will set $X = k^3 + \frac{k(k-1)}{2}$ and check whether the smallest dense subgraph $B \cup \{N\}$ found by $P2$ has size $k+1$. If yes then solution of $P1$ is B, terminate. If not then set $X = X - 1$, repeat. ∎

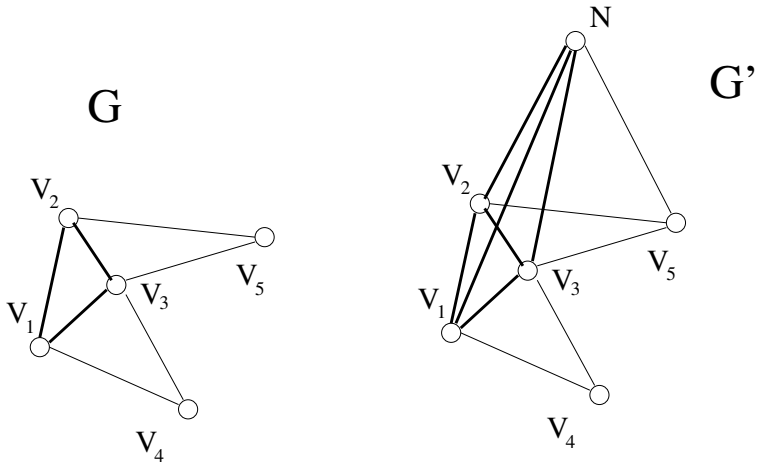


Figure 4–1: Reduction

Claim 36 *Problem* $ADB2 \leq_m ADB1$.

Proof Since problem $ADB2$ is NP-Complete, $ADB2 \leq_m k - CLIQUE$. And it was shown in Claim 31 that $k - CLIQUE \leq_m ADB1$. Hence $ADB2 \leq_m ADB1$.

∎

Claim 37 *Problem* $ADB1 \leq_m ADB2$.

Proof Similar to proof of Claim 36, using composition of many-one reduction from $ADB1$ to $k-CLIQUE$ and from $k-CLIQUE$ to $ADB2$ in Theorem 2 in Section 4.2. ∎

4.1.4 Relationships between Approximation Algorithms

Let $A1(G,k,c)$ be a polynomial time algorithm that returns a graph $B, |V(B)| = k$ such that $|E(B)/E(L)| > c$ where L is a subgraph of G of size k that has largest number of edges among all the subgraphs of this size.

Let $A2(G,X,c)$ be a polynomial time algorithm that returns a dense subgraph B such that $|B|/|L| < c$ where L is the minimum dense subgraph of G.

Various approximation algorithms for problem $AD1$ has been proposed Asahiro and Iwama (1995), Feige and Seltzer (1997), Ye and Zhang (1997),Kortsarz and Peleg (1993),Srivastav and Wolf (1999). Consider following natural algorithm $AP1$ for finding approximate solution of problem $P1$ assuming the existence of algorithm $A2()$.

(Approximating $A1$ by $A2$). Consider the following algorithm $AP1$. Let (G,k,c) be an input for $A1()$. Construct a graph G' as in Claim 35. Let $X = k^3 + k(k-1)/2$. Let $B \subseteq G'$ be the subgraph found by $A2$. If $B = \emptyset$ then set $X = X - 1$, repeat. Otherwise if $|B| = k+1$ then return B. If $|B| \neq k+1$ then remove $(|B|/k-1)|B|$ vertices of B in greedy fashion using technique of Asahiro and Iwama (1995) (the one that has smallest degree should be removed first, then second smallest etc)

Conjecture 1 *A polynomial time algorithm $AP1$ provides a guaranteed approximation factor solution of problem $P1$.*

On other hand it seems difficult to find approximate solution of problem $P2$ (assuming the existence of algorithm $A1()$), as example below demonstrate.

(Approximating $A2$ by $A1$). Consider the following approximation algorithm $AP2$ for finding minimum dense subgraph of G. Wlog suppose that $w(e) = 1, w(v) =$

$c, \forall e \in E, v \in V$. We start with $k = 1$, find $B = A1(G, k, c)$ and check whether B is dense. If yes then output B, otherwise increase k, repeat.

Claim 38 *For any $d > 0, d < |G|/|L|$ there is an instance (G, X) such that $AP2$ returns dense subgraph B such that $|B|/|L| > d$, where L is the minimum dense subgraph.*

Proof Let $\{O_k\}$ be the set of minimal dense graphs of density exactly X, size of each $|O_k| = k, k \leq n$. Let $\{S_k\}$ be the set of underconstrained graphs of density exactly $X - 1$, size of each $|S_k| = k, k < n$ and $S_n = O_n$. Let $G_k = O_k \cup S_k$ and $G = \cup G_k$. Suppose that for every $k < n$ a call to $A1$ by $AP2$ returns S_k and not O_k, hence $AP2$ will output O_n instead of O_1. ∎

4.2 NP-Completeness of Minimum Stably Dense Subgraph Problem

Notation: (a, b, c, G) problem denotes a minimum dense graph problem where all vertices of graph G have weight a, all edges have weight b and constant $K = c$

Lemma 5 *The general (a, b, c, G) problem is NP-Complete.*

Proof This is shown by reduction from *CLIQUE OF SIZE P*, by converting an instance of *CLIQUE OF SIZE P* $(G, P), G = (V, E)$ into an instance of $((P - 1)/2, 1, 0, G = G')$ problem (we assume that $P - 1$ is even). Then there is a minimum dense graph A' of size P in G' if and only if there is a clique A of size P in G, since density of any graph A' of size x $d(A') \leq x(x-1)/2 - x(P-1)/2$, hence all dense graphs have size $x \geq P$ and clique of size P would be the smallest dense graph ∎

Theorem 2 *The $(2, 1, 0, G)$ problem is NP-Complete.*

Proof This is shown by converting an instance of $((P - 1)/2, 1, 0, G)$ problem into an instance of $(2, 1, 0, G')$ problem and using previous lemma. ∎

Description of the conversion.

Every vertex v of G is replaced by a subgraph $H(v) = (V', E')$ in G'. This subgraph has $(deg(v) + 1)(\frac{P-1}{4} - 1) + 1$ vertices, where $deg(v)$ is degree of the

vertex v in graph G, see Figure 4–3 (we assume that 4 divides $(P-1)$). All vertices have weight 2, all edges have weight 1. All vertices of $H(v)$ are arranged either in a leftmost column of $(P-1)/4$ **reed** vertices r_i or a rectangle of $deg(v)$ columns and $(P-1)/4-1$ rows of **key** vertices $k_{i,j}$. Set of all reed vertices is denoted by R, set of all key vertices by K. For all i, j s.t $0 \leq i \leq (P-1)/4-2, 1 \leq j \leq deg(v)$ there are edges between $k_{i,j}$ and $k_{i+1,j}$ as well as between $k_{i,j}$ and r_i. Also there is an edge between $k_{(P-1)/4-1,j}$ and $r_{(P-1)/4-2}$ as well as between $k_{(P-1)/4-1,j}$ and $r_{(P-1)/4-1}$ for all $j, 1 \leq j \leq deg(v)$. There are $deg(v)$ key vertices $k_{0,j}$ that are connected to the vertices outside of $H(v)$. These vertices correspond to the edges in G that are incident to v, see Figure 4–2. These vertices are called **portals**, their set is denoted by S.

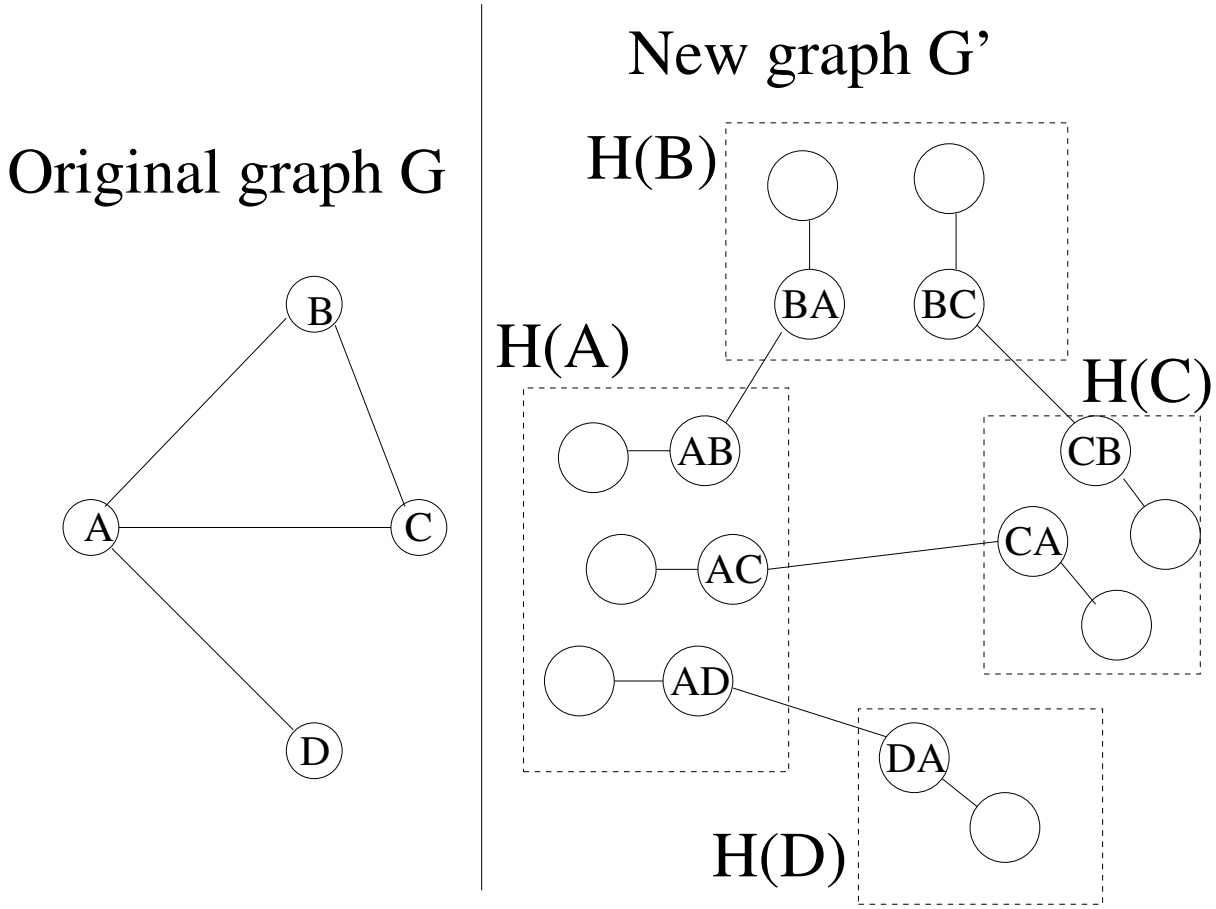


Figure 4–2: Gadgets for reduction to CLIQUE

Lemma 6 *Density of $H(v)$ is equal to $-(P-1)/2$.*

Proof Consider a following mapping $F : E' \to V', F(k_{i,j}, r_x) = k_{i,j}, F(k_{i,j}, k_{i+1,j}) = k_{i,j}$ (essentialy every edge is mapped to its upper endpoint). Then for all $k_{i,j}$ there exist exactly 2 edges e_1 and e_2 such that $F(e_1) = F(e_2) = k_{i,j}$, hence density of $H(v), d(H(v)) = |E'| - 2|V'| = 2|K| - 2|K| - 2|R| = -2|R| = -(P-1)/2$.

(Similarly density of a union of any number of rows of keys and of all reeds is equal to $-(P-1)/2$). ∎

Lemma 7 *Density of any $A \subseteq H(v), d(A) < 0$.*

Proof Consider a function $L_A : V_A \to \{0, 1, 2\}$, for $w \in A, L_A(w) = 2$ if there exist 2 edges e_1 and e_2 in A such that $F(e_1) = F(e_2) = w$

$L_A(w) = 1$ if there exists 1 such edge e_1 in A such that $F(e_1) = w$

$L_A(w) = 0$ otherwise. Let $L_A^t = \{w | L_A(w) = t\}$.

Then density of $A, d(A) = -2|L_A^0| - |L_A^1|$. Consider $B = A \cap R \subseteq L_A^0$. If $B \neq \emptyset$ then $L_A^0 \neq \emptyset$ therefore $d(A) < 0$. If $B = \emptyset$ then consider a $k_{i,j} \in A, i \geq i^*, j \leq j^*, \forall k_{i^*,j^*} \in A$ (lower-left key in A). This $k_{i,j} \in L_A^0$ therefore $d(A) < 0$. ∎

Lemma 8 *Let $B' \subseteq G'$ be a minimal dense subgraph. Then $A = B' \cap H(v)$ is either empty or has density $-(P-1)/2$ for all vertices $v \in G$ (in other words B has the same density as original vertex v).*

Proof Let $S_A = A \cap S$ (set of all portal vertices of A). If $A \neq \emptyset$ then by Lemma 7, $S_A \neq \emptyset$. Since B' is minimal dense $d(A) > -|S_A|$, otherwise $d(B' \setminus A) \geq 0$ but B' is minimal. If $k_{0,j} \in S_A$ then $k_{i,j} \in A, \forall i$, (i.e entire column of keys is in A) otherwise $d(B' \setminus (\bigcup_i k_{i,j})) \geq 0$ but B' is minimal. For the same reason all such $k_{i,j} \in L_A^2$, hence all reeds $r_i \in A$. Therefore $d(A) = |E_A| - 2 * |V_A| = |E_A| - 2 * |L_A^2 = A \cap K| - 2 * |R| = -2 * |R| = -(P-1)/2$. ∎

Lemma 9 *Let $B \subseteq G$ be a clique of size P. Then in corresponding $H(B) = B' \subseteq G'$ there exists a dense subgraph $C' \subseteq B'$ such that $|C'| = ((P-1)/4 - 1)P(P-1) + P(P-1)/4$.*

Proof Graph C' contains all reed vertices of $H(B)$ as well as all columns of key vertices $k_{i,j}$ of $H(v_1)$ iff $k_{0,j} \in H(v_1)$ is connected to $k_{0,j'} \in H(v_2)$ and both $v_1, v_2 \in B$. I.e columns of key vertices $k_{i,j}$ of $H(v)$ such that portal $k_{0,j}$ is connected to $H(w), w \notin B$ are not included in C'.

This graph $C' = C_1 \cup \ldots \cup C_P$, where $C_i = C \cap H(v_i)$ for P vertices $v_i \in B$ Density of each $C_i, d(C_i) = -(P-1)/2$. There are $P(P-1)/2$ edges between C_i's. Therefore C' is dense. ∎

Lemma 10 *Let $B \subseteq G$ be a clique of size P. Let C' be a minimum dense graph in G'. Then $|C'| = ((P-1)/4 - 1)P(P-1) + P(P-1)/4$ and $U(C') = \{v \in G | H(v) \cap C' \neq \emptyset\}$ is a clique of size P.*

Proof By Lemma 9 $C' \cap H(v)$ is either empty or has density $-(P-1)/2$ for all vertices $v \in G$. Let $C = C_1 \cup \ldots \cup C_t, C_i = C' \cap H(v_i)$. Let $IE_{C'}$ be a number of edges between C_i's. Since C' is dense $IE_{C'} \geq t(P-1)/2$. Since $IE_{C'} \leq t(t-1)/2$ it follows that $t \geq P$. Therefore $IE_{C'} \geq P(P-1)/2$. Hence the total number of key vertices in C' is greater than or equal to $2(P(P-1)/2)((P-1)/4-1)$ and the total number of reed vertices is greater than or equal to $P(P-1)/4$. Since equality are reached when C' derived from $H(B)$ is considered, claim follows. ∎

This completes proof of Theorem 2.

4.3 Special Cases of Minimum Stably Dense

4.3.1 Flow-based Solution for No-overconstrained Case

Assume that all dense subgraphs A of G have the same density $d(A) = K$ (i.e there are no overconstrained subgraphs). In this case the following two claims hold.

Claim 39 *If $d(X) \leq K, \forall X \subseteq G$ then for any two minimal dense graphs $A, B \subseteq G$ it follows that $A \cap B = \emptyset$.*

Proof Proof by contradiction, suppose that $A \cap B = C \neq \emptyset$. Since A and B were minimal, $d(C) < K$, hence $d(A \cup B) = d(A) + d(B) - d(C) = 2K - d(C) > K$,

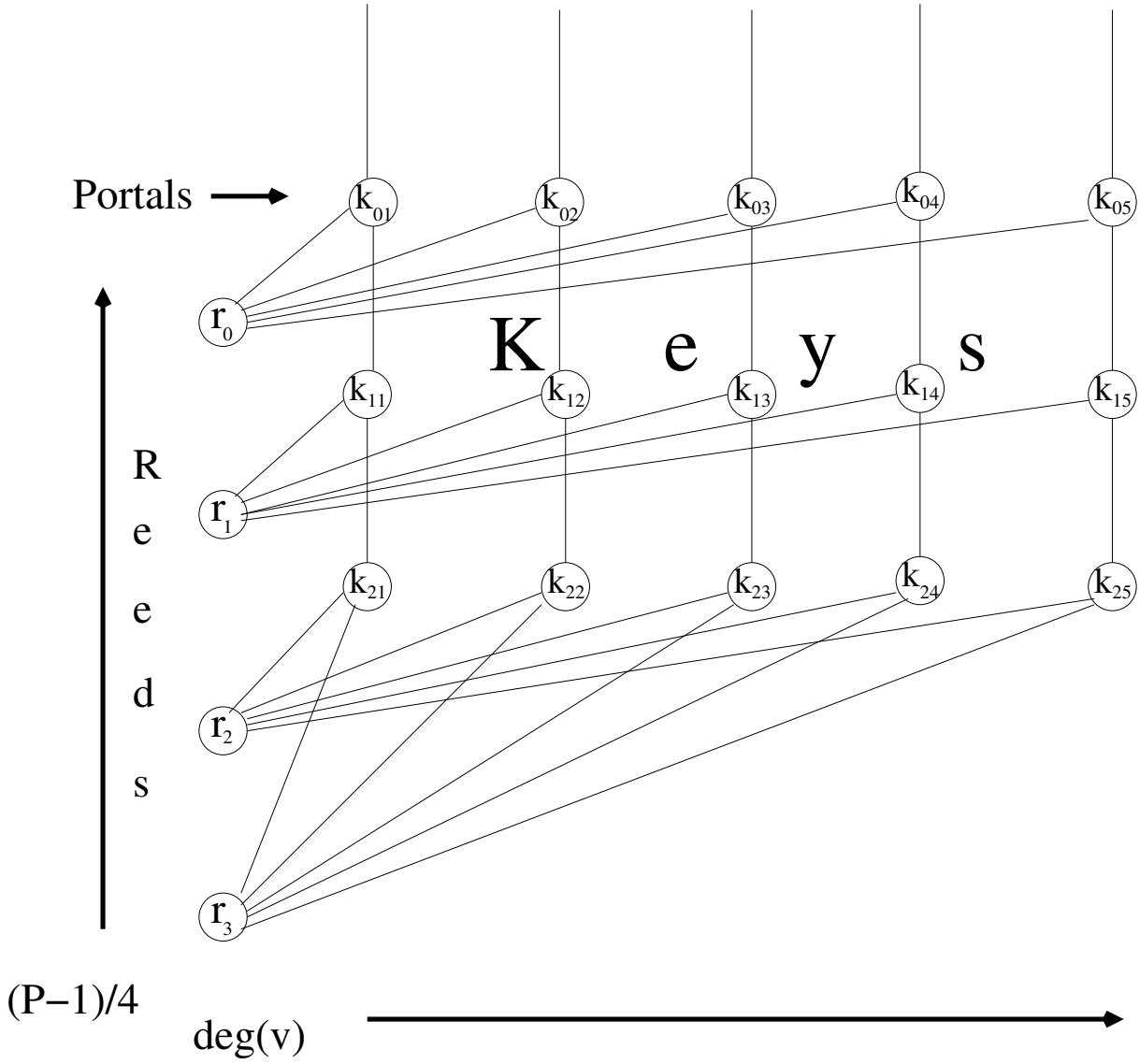


Figure 4–3: $H(v)$ representing vertex v

contradiction since we assumed that no subgraph of G has density greater than K.

∎

Thus when there are no overconstrained subgraphs graph G can be decomposed into a union of pairwise disjoint minimal dense graphs $\cup_i A_i$ and remainder subgraph C that does not contain any dense subgraphs. Formally $G = \cup_i A_i \cup C, d(A_i) = K, A_i \cap A_j = \emptyset, \forall B \subseteq C, d(B) < K$.

From this structure it clearly follows that

Claim 40 *If $d(X) \leq K, \forall X \subseteq G$ then then ND = number of minimal dense subgraphs of G is less than or equal to $|G| = n$.*

Proof Let $A_1, \ldots, A_{ND}$ be minimal dense subgraphs of G, then $\cup A_i \subseteq G$ and since $A_i \cap A_j = \emptyset, \forall i, j$ it follows that $\Sigma|A_i| = \cup|A_i| \leq |G| = n$. Since every $|A_i| \geq 1$ it follows that $ND \leq n$. ∎

Also since A_i are edge-disjoint (as well as vertex-disjoint) it follows that

Claim 41 *First dense subgraph A of G found by Dense(G) algorithm is minimal dense.*

Proof Proof by contradiction, suppose that $A = A_1 \cup A_p, p > 1$. Let e be the last edge *distributed* in by routine *Distribute(e)*. Since $A_1, \ldots, A_p$ are edge-disjoint it follows that e belongs to only one subgraphs, wlog $e \in A_1$. This means that all edges of say A_2 were distributed before e, but then dense graph A_2 would have bee found before A, contradiction. ∎

Properties above yield following algorithm for finding minimum dense subgraph

Algorithm 1 *Suppose that $d(X) \leq K, \forall X \subseteq G$. Then following routine finds all minimal dense subgraphs of G: apply Dense(G) to find first minimal dense subgraph A_1. Take $G = G \setminus A_1$ and repeat until no more dense subgraphs left. By examining the sizes of A_is we locate a minimum dense subgraph A_p.*

4.3.2 Preflow-push based Solution for No-overconstrained Case

Again, suppose that $d(X) \leq K, \forall X \subseteq G$. Assume that there is at least one dense subgraph. Assume that weights of all vertices are the same ($= c$). Then an algorithm of Kortsarz and Peleg (1993) can be used to find a subgraph $A \subseteq G$ that maximizes

$$\alpha(A) = \frac{\sum w(e(A)) - K + 1}{\sum w(v(A))}$$

Claim 42 *Subgraph A found above is a minimal dense subgraph of smallest cardinality.*

Proof Since

$$\alpha(A) - 1 = \frac{d(A) - K + 1}{c|A|},$$

the maximum of $\alpha()$ is reached by such an A that numerator is equal to 1 (thus A is dense) and denominator is as small as possible (thus A is minimal and of smallest cardinality). ∎

Note 2 *The algorithm of [GGT] could be adapted to finding maximum of $\alpha()$ since subgraph A that maximizes modified dual difference $\sum w(e(A)) - K + 1 - \lambda(\sum w(v(A)))$ is the same A that maximizes original dual difference $\sum w(e(A)) - \lambda(\sum w(v(A)))$. The only difference arises in the computing values of λ as $\frac{\sum w(e(A)) - K + 1}{\sum w(v(A))}$ instead of the original values $\frac{\sum w(e(A))}{\sum w(v(A))}$.*

If weights of vertices are not constant but bounded by some constant say C then algorithm above yields C approximation of a minimal dense subgraph. However there is a better way of dealing with unequal vertex weights described below.

Remark 1 *It is possible to remove the condition above that demands equality of all vertex weights. Let M be the maximum vertex weight of G. Then every vertex v of weight $L \leq M$ could be replaced by vertex v' of weight M and a looping edge from v' to v' of weight $M - L$. Modified graph G' has the same dense subgraphs and all vertices of G' have the same weight M.*

4.3.3 Finding Smallest Subgraph of Largest Density

If we don't have any restrictions on density of subgraphs of G, then we still can use modification of [GGT] algorithm described above to find subgraph $A \subseteq G$

such that $d(A) \geq d(B), \forall B \subseteq G$ and $|A| \leq |C|, \forall C \subseteq G, d(C) = d(A)$. This is done by trying $K = n^2, n^2 - 1, \ldots$ until A has been found.

4.3.4 Case of Bounded Number of Overconstraints

In this subsection we will wlog suppose that all edges of G have weight 1 and all vertices have weight 3, but results could be generalized for arbitrary integer weights. Here we will relax original condition that graph G has no overconstrained subgraphs and instead will assume that constant number of edges of G can be removed without violating *solvability* of any subgraph of G. Recall that dense graph $A = (V(A), E(A))$ is *solvable* if either A does not contain any overconstrained subgraphs i.e $\forall B \subseteq A, d(B) \leq K$ or there is a set of edges $R(A) \subseteq E(A)$ such that $(V(A), E(A) \setminus R(A))$ is dense but does not contain any overconstrained subgraphs.

Claim 43 *For any graph $G = (V, E)$ we assume there is an edge set $R(G)$ such that for all solvable subgraphs $B \subseteq G$ a reduced subgraph $B' = (V(B), E(B) \setminus R(G))$ remains solvable and for all subgraphs $C \subseteq G' = (V, E \setminus R(G)), d(C) \leq K$, i.e G' has no overconstrained subgraphs.*

Proof Let $A_1, \ldots, A_k$ be the set of maximal solvable subgraphs of G. Due to maximality of A_is, all $|A_i \cap A_j| \leq 1$, i.e no two subgraphs have an edge in common. Therefore edges of E can be decomposed into mutually disjoint sets $E(G) = E(A_1) \cup E(A_2) \ldots \cup E(A_k) \cup S$. For every A_i there is an edge set $R(A_i)$ that can be removed without affecting solvability of A_i and of other subgraphs as well (due to disjointedness). Then $R(G) = \cup R(A_i) \cup S$ satisfies the criteria above.

∎

Definition 1 *Such a set of edges $R(G)$ described above is called maximal solvability preserving removable edge set or MSPRES. Graph G that has MSPRES $R, |R| = c$, where is called c-overconstrained.*

Claim 44 *It is possible to find a minimum dense subgraph of c-overconstrained graph $G = (V, E)$ in $|E|^{c+3}$ time. Also the number of wellconstrained subgraphs of G is $O(|E|^{c+3})$.*

Proof This can be done by brute force examination of all possible combinations of c edges can to be removed, considering resulting graph G and applying technique from first subsection to G' in order to find a minimum dense subgraph. Note that at least one of the combinations of c edges will correspond to no-overconstrained case examined in the first subsection, hence algorithm will yield correct result. Note also that we can map any solvable subgraph to edge combinations upon whose removal the resulting subgraph is still solvable and has density exactly K. ∎

A related result below shows that a minimal dense subgraph cannot have very large density. Hence if we are given a dense graph G of small density we can use brute force algorithm of Claim 44 to find minim dense subgraph, and if the density of G is large we know that it likely to have smaller minimal dense graph of small density.

Claim 45 *Wlog that all edges have weight 1, all vertices have weight 3. If dimension-dependent constant $K = -3$ then no minimal dense subgraph can have density more than 4.*

Proof Proof by contradiction, suppose that graph A has density 5 or more, take any vertex v of A, then the number of edges adjacent to v is greater than 8 (otherwise $A \setminus v$ is dense). If we add up this inequailty over all vertices then $2|E| > 8N, E > 4N$. Take any vertex v. It degree is less than or equal to $N - 1$ hence density of $A \setminus v$ is greater or equal to $E-(N-1)-3(N-1) \geq 4N-4N+4 \geq 4$, hence $A \setminus v$ is dense and A was not minimal dense, contradiction. ∎

4.3.5 Size Overconstrained Graphs

Now we will consider a problem of finding minimum dense subgraphs of at least a certain size $c+1$, under restriction that all overconstrained graphs have size exactly c (we will not be making any restrictions on density of such subgraphs). Note that without the restriction on size of overconstrained subgraph the problem of finding minimum dense subgraph is NP-Complete by reduction from CLIQUE, so we cannot generalize it too much in that direction.

Observation 5 *Let $A_1, \ldots, A_k$ be the set of overconstrained subgraphs of $G, |A_i| \leq c, \forall i$. Let X be a minimum dense subgraph of $G, |X| > c$. Then for all $B \subseteq G \setminus X, d(B) < 0$ (otherwise $B \cup X$ would be overconstrained subgraph of size greater than c).*

Claim 46 *Let e be an edge in G such that $e \notin A_i, \forall i$. Then there is at most one minimal dense subgraph $C \subseteq G$, such that $e \in C$ and $|C| > c$.*

Proof Suppose that this edge $e \in C, D$ minimal dense subgraphs of size at least $c+1$. Consider $F = C \cup D$. The $d(F) = d(C) + d(D) - d(C \cap D) = K + K - d(C \cap D) \leq K$. Hence $d(C \cap D) \geq K$ but $|C \cap D| > c+1$ (since $e \notin A_i, \forall i$), contradicting minimality of C and D. ∎

The following algorithm locates all minimal dense subgraphs of size $c+1$ or more. Wlog we assume that all dense graphs are connected graphs and all overconstrained subgraphs have size exactly c.

Algorithm 2 *1. Apply Algorithm 1 to graph $F = G \setminus (A_1 \cup \ldots A_k)$, Record the size of the smallest found subgraph.*

2. Distribute all the edges in F (they were removed during stage 1).

3. For all A_i distribute flows within A_i.

4. For $i = 1 : k$, take $F = F \cup A_i$ and perform Pushoutside(A_i) on all edges e adjacent to A_i, in order to find a dense graph M containing e and A_i. Once M has been found perform Minimal(M,e) in order to find a minimal dense subgraph that

contains e and A_i. Record the size of the smallest minimal subgraph found so far, repeat.

Claim 47 *Every minimal dense subgraph M of G of size $c+1$ or more, will be found by Algorithm 2.*

Proof Every edge of M will be *distributed* during execution of Algorithm 2. Consider edge e of M that was distributed last. Suppose that $e \in G \setminus (A_1 \cup \ldots A_k)$ Since $e \in M$ during its distribution some minimal dense graph containing e was found during stage 1. According to Claim 46 this graph has to be M.

Now suppose that e has exactly one endpoint in A_i (if it has both endpoints in A_i then it is not going to be last edge of M to be distributed, due to connectivity of M). While performing *Pushoutside(A_i)* on edge e, some dense graph containing e and A_i will be found and then a minimal dense graph containing e and A_i was found. Then by reasoning similar to Claim 46 this graph has to be M. ∎

Here is an estimate on running time of this algorithm.

Claim 48 *The number of minimal dense subgraphs of size $c+1$ or more is less than or equal to $\Sigma_{i=0}^{c} \binom{n}{i}$.*

Proof This is direct consequence of a result in (Frankl, 1982). ∎

4.4 Approximation Algorithms for Minimum Stably Dense

4.4.1 Randomized Approximation Algorithms

In Section 3.2 we describe a polynomial time network-flow based algorithm *Dense* that locates dense subgraphs in polynomial time. Given a graph G, *Dense* will either locate a dense subgraph $A \subseteq G$, or determine that G does not contain any dense subgraphs.

Using algorithm *Dense* as subroutine a following randomized algorithm *Randomized Dense* is proposed for finding minimum dense subgraphs (of the entire graph G that is assumed to be dense throughout this section). This algorithm locates minimal dense subgraph, which is not guaranteed to be minimum.

0. $A = G$, unmark all vertices $v \in G$

1. Choose an unmarked vertex $v \in A$ at random. If there are no unmarked vertices left (in A) then return A

2. Mark and remove v and it adjacent edges from A, apply *Dense* to $A \setminus v$.

3. If algorithm *Dense* found dense subgraph $B \subseteq A \setminus v$, then set $A = B$ goto 1.

4. Else restore v (and edges) into A, goto 1 (v remains marked).

We will show that this algorithm provides bad approximation to the minimum dense problem both in general case and in the case when weights of graph edges/vertices are bounded (second counterexample does make first (general) counterexample unnecessary, but we do include general counterexample below since it is easy to analyze).

Theorem 3 *Consider general (a, b, c, G) problem (recall that this means that weights of edges/vertices and dimension dependent constant are unbounded). There is a graph $G, |G| = n + 2y$ such that algorithm Randomized Dense will locate, with probability $1 - n^{-y}$, a minimal dense subgraph of G of size $\sqrt{n}$, where y is any positive constant. On other hand this graph G has a minimum dense subgraph of size y that algorithm Randomized Dense will find with (low) probability n^{-y}. Therefore Randomized Dense is not FPAS.*

Proof Consider a graph G that consists of two nonoverlapping dense subgraphs A and B. Graph A is minimum dense subgraph, $|A| = x$, where x is a constant ($= 2y$). Weights of edges and vertices of A are not important, as long as A is minimum dense. Graph B is a clique of size n. Every vertex of B has weight $\frac{\sqrt{n}-1}{2}$, every edge of B has weight 1, dimension dependent constant $c = 0$. Therefore every subgraph of G of size $\sqrt{n}$ is minimal dense.

Suppose that at stage t of running *Randomized Dense* algorithm we have reduced $G = A \cup B$ to $G^t = A^t \cup B^t$. If $|B^t| \geq \sqrt{n}$ then B^t still contains dense subgraphs. Hence if $|A^t| < x$ and $|B^t| \geq \sqrt{n}$ then *Randomized Dense* will

output subgraph of size $\sqrt{n}$. Therefore the only way subgraph A could be found by *Randomized Dense* is when (all) first $n - \sqrt{n} + 1$ vertices v removed by *Randomized Dense* are located in B (note that new dense subgraph found by *Randomized Dense* in $G^t \setminus v$ will be $G^t \setminus v$ itself). What is the probability of this? Probability that first removed vertex was in B is $\frac{B}{A+B} = \frac{n}{n+x}$. Probability that second removed vertex was also in B is $\frac{B-1}{A+B-1} = \frac{n-1}{n+x-1}$ etc. Hence the probability that all $n - \sqrt{n} + 1$ vertices v removed by *Randomized Dense* are located in B is

$$\frac{n}{n+x}\frac{n-1}{n+x-1}\cdots\frac{\sqrt{n}+1}{\sqrt{n}+x+1}\frac{\sqrt{n}}{\sqrt{n}+x}$$

After cancelling like terms this probability evaluates to $O(\frac{(\sqrt{n})^x}{n^x}) = O(n^{-1/2x})$. Taking $x = 2y$ claim of Theorem 3 follows.

(Note that there is nothing magic about $\sqrt{n}$ in the proof above, many other functions of n would have been equally good). ∎

The following technical lemma will allow us to estimate probability of *Randomized Dense* outputting A and will help to analyze bounded case.

Lemma 11 *If there is a dense graph $G = A \cup B, |A| << |B|$, subgraph A is minimum dense, all minimal dense subgraphs C of B have size $|C|, |C| >> |A|$ and every time vertex is removed from B^t in Step 2 of Randomized Dense the size difference is $|B^t| - |B^{t+1}| = d$ then probability of Randomized Dense outputting A is $O((\frac{|C|}{|A|+|B|})^{|A|/d})$.*

Proof Probability that first removed vertex was in B is $\frac{B}{A+B}$. Probability that second removed vertex was also in B is $\frac{B-d-1}{A+B-d-1}$ etc. Hence the probability that all $(|B| - |C|)/d + 1$ vertices v removed by *Randomized Dense* are located in B is

$$\frac{B}{B+A}\frac{B-d-1}{B+A-d-1}\cdots\frac{C+d}{C+A+d}\frac{C}{C+A}=O((\frac{|C|}{|A|+|B|})^{|A|/d})$$

∎

The following shows that Theorem 3 could be extended to the bounded case.

Theorem 4 *Consider* $(2,1,0,G)$ *problem (recall that this means that weights of edges/vertices and dimension dependent constant are bounded). There is a graph* $G, |G| = \Theta(n\log n + \log^2 n)$ *such that algorithm Randomized Dense with probability* $1-(\frac{\log^3 n}{n})^{\log n}$ *will locate a minimal dense subgraph of* G *of size* $\log^3 n$*, On other hand this graph* G *has a minimum dense subgraph of size* $\log^2 n$ *that algorithm Randomized Dense will find with (low) probability* $(\frac{\log^3 n}{n})^{\log n}$*. Therefore Randomized Dense is not FPAS.*

Proof Consider a graph G that consists of two nonoverlapping dense subgraphs A and B. Graph A is minimum dense subgraph, $|A| = \log^2 n$. Weights of edges and vertices of A are not important, as long as A is minimum dense. Graph B is an image under conversion $H()$ described in proof of Theorem 2, of a clique of size $\sqrt{n}$. Value $P = 2\log n$. Then size of an image of any vertex $H(v)$ is $\Theta(\log(n)\sqrt{n})$ and since there are $\sqrt{n}$ vertices in the original clique, size of B is $\Theta(n\log(n))$. Also size of any minimal dense subgraph of B is $\Theta(\log^3 n)$.

Suppose that at stage t of running *Randomized Dense* algorithm we have reduced $G = A \cup B$ to $G^t = A^t \cup B^t$. If B^t still contains dense subgraphs and $|A^t| < \log^2 n$ then algorithm *Randomized Dense* will return graph of size $\Theta(\log^3 n)$. Therefore in order for *Randomized Dense* to return graph A, all the vertices to be removed (in Step 2) must come from B until B^t is empty. How many vertices should be removed? Note that when *Randomized Dense* removes a vertex v from B^t it may find that $B^t \setminus v$ is not dense, but instead some proper subgraph $B^{t+1} \subset B^t \setminus v$ is dense. Suppose that for all t, until $B^{=}\emptyset$ the size

difference $|B^t| - |B^{t+1}| = \log n$. (Then number of vertices removed in Step 2, when *Randomized Dense* outputs A, is n).

If we were able to show that for our graph $G, |B^t| - |B^{t+1}| = \log n$ then Theorem 4 would follow from Lemma 11.

Consider Step 2 of *Randomized Dense*. If vertex v is a Key (see definitions in conversion used in Theorem 2) then $|B^t| - |B^{t+1}|$ is at most $O(\log n)$ If the vertex v is a Reed then $|B^t| - |B^{t+1}|$ is at most $O(\sqrt{n}\log n)$ (generally much less both for Key and Reed). Probability that randomly chosen vertex v is a Reed is $\frac{1}{\sqrt{n}}$. Hence expected value of $|B^t| - |B^{t+1}|$ is

$$\frac{1}{\sqrt{n}}\sqrt{n}\log n + (1 - \frac{1}{\sqrt{n}})\log n = O(\log n)$$

Using Lemma 11, Theorem 4 follows. ∎

Combination of Randomized Dense and Modified Minimal. In this section we combine Randomized Dense algorithm and a modification of Minimal Dense algorithm to find an approximate solution of Minimum dense subgraph problem. We will show that there are graph counterexamples where this combination does not yield constant factor approximation.

This combination will be called *Double Random Flow Minimum Algorithm (DRFMA)*. It consists of two independent parts.

1. Run Randomized Dense Algorithm, resulting in subgraph A_1 - an approximation of minimum dense subgraph.

2. Run *Global Minimal Algorithm*, described below, resulting in subgraph A_2 - an approximation of minimum dense subgraph.

Output the smallest of A_1 and A_2.

Description of *Global Minimal Algorithm*.

Run algorithm Dense(G). Recall that this algorithm terminates once an edge e cannot be distributed, indicating a presence of a dense subgraph B. Run an algorithm Minimal(B), finding subgraph A_2 - current approximation of minimum dense subgraph. If it was the weight of e that cannot be distributed then remove e from G. Continue distributing edges of G via algorithm Dense(G). If another dense subgraph B has been found then run Minimal(B), update A_2 if necessary, continue. After all edges of G have either been distributed or removed, output A_2.

Designing counterexample for DRFMA.

We would like to construct a counterexample, where DRFMA does not yield constant factor approximation. Thus both Randomized Dense and Global Minimal Algorithms should fail to yield constant factor approximation to minimum dense subgraph. Suppose that while constructing counterexample we limit ourself to graphs G that has X large minimal dense subgraphs $C_1, \ldots, C_X$ of size $L = |C_i|$ and one small minimum dense subgraph A of size $R = |A|$, where $L/R = f(n)$ is a non-constant function. Intuitively, Randomized Dense would fail if X is sufficiently large. When will DRFMA fail to locate minimum dense subgraph? Consider what happens when we distribute the edges of the last vertex of A to be distributed. Since A is dense, at this stage Dense() should return non-empty dense subgraph $B, A \subseteq B$. Hence in order for DRFMA to fail, Minimal(B) should output one of C_i instead of A. Either one edge of A was impossible to distribute previously (and therefore removed from G) (and hence Minimal(B) failed to locate A) or one of the vertices v of A was removed at random while running Minimal(B) and some other minimal dense subgraph $C_i, v \notin C_i$ was located. Probability of latter occurrence is similar to probability of Randomized Dense eliminating A, so (as will be shown below) it is alone sufficient to guarantee a failure of Global Minimal with high probability. However, the probability of former, that is a failure due to the way the algorithm deals with overconstrained graphs (by simply removing edges that it

cannot distribute and continuing) also occurs with high probability as we will show below.

Consider a Figure 4–4. Graph ABDEF is minimum dense for dimension dependent constant $K = 3$. Weights of all edges is 1, except that of AF, BF, DF and EF which is equal to 4, weights of vertices A,B,D and E is 4, that of F is 5. There are $X/2$ cliques $C_1, \ldots, C_{X/2}$ on the left and $X/2$ cliques $C_{X/2+1}, \ldots, C_X$ on the right. Every clique contains Z vertices. All of the Z vertices in any clique are connected to each other and also either to A and B or D and E. Weight of every vertex in the clique is equal to W, such that $ZW + 8 = Z(Z-1)/2 + 2Z + 3$, i.e the clique plus a pair A,B or D,E form a minimal dense subgraph. However an edge AB (DE) will make this subgraph overconstrained. First we will show that with high probability algorithm Global minimal will distribute all edges in one of such large dense graphs before AB or DE, thus AB or DE would have to be removed since they could not be distributed.

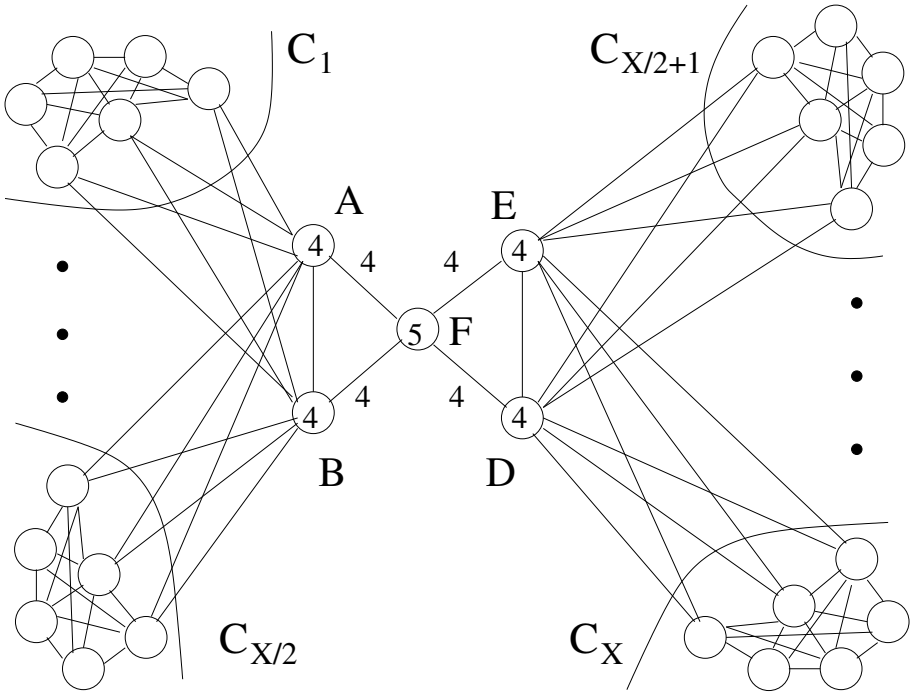


Figure 4–4: Counterexample for DRFMA

Claim 49 *Let* $W = log(n), Z = 2 * log(n) - 3$ *then with probability* $1 - e^{\frac{-n}{log^2 n}}$ *all edges of some large minimal dense subgraph (containing vertices A and B, but not edge AB) will be distributed before edge AB.*

Proof Consider the probability that edge AB will be distributed before some edge edge of all large minimal dense subgraphs containing A and B. This probability for any given large subgraph of size Z is equal to $\frac{Z}{Z+1}$. Hence for $n/2Z$ graphs this probability is $(\frac{Z}{Z+1})^{n/2Z}$ that goes to $e^{\frac{-n}{2Z^2}}$, hence claim follows. ∎

So with high probability the first edge (either AB or DE) will be removed from G. Therefore with high probability when the second edge will be distributed (either DE or AB) the minimum dense subgraph ABDEF will not be found by Minimal() algorithm. This proves following lemma.

Lemma 12 *There is a graph G, such that with probability* $1 - e^{\frac{-n}{log^2 n}}$ *algorithm DRFMA will locate a minimal dense subgraph that is* $\log(n)/5$ *greater than minimum one.*

4.4.2 Minimum Dense as Minimum Cost Flows

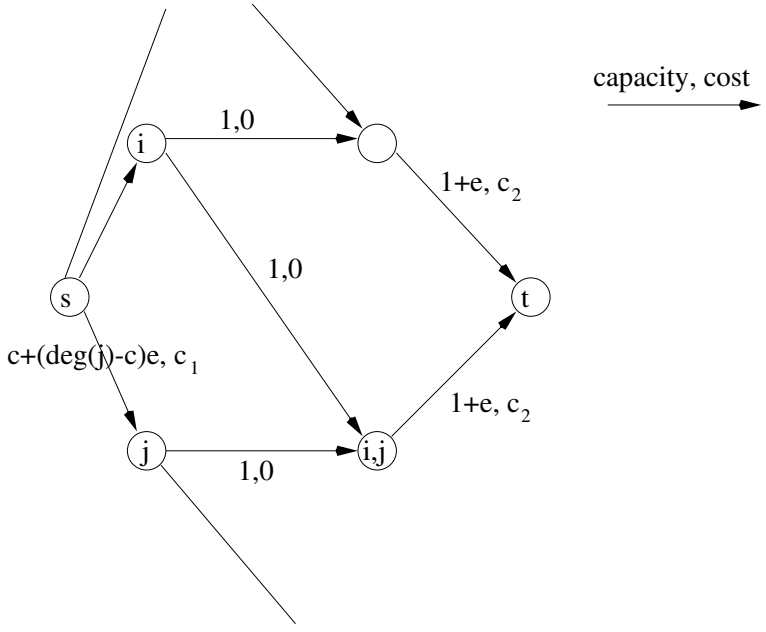


Figure 4–5: Network

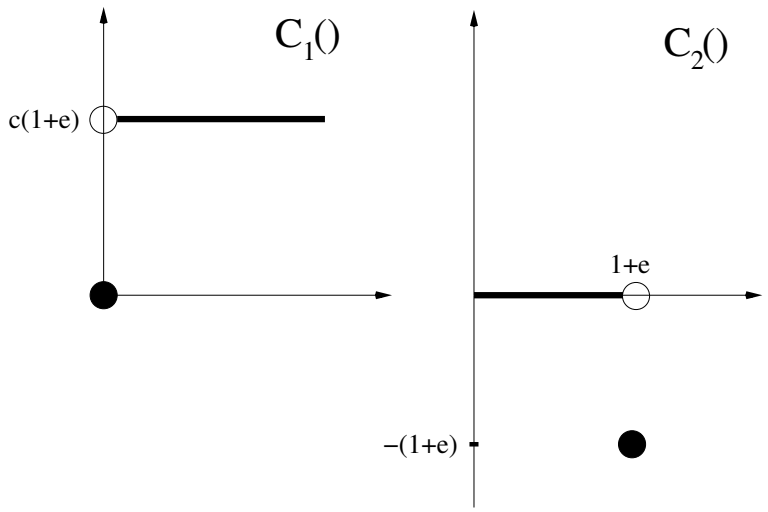


Figure 4–6: Cost functions

Let $G = (V, E)$ be the graph where the minimum dense subgraph is to be found. Without loss of generality assume that all vertices of G have the same weight $w(v) = c$ and constant $K = 0$.

The network N, that corresponds to G, consists of the bipartite graph (V_N, E_N) and the pair of source and sink vertices s, t. Vertices of V_N correspond to the vertices of V, vertices of E_N correspond to the edges of E. The network N has following (oriented) edges. A vertex $v_N \in V_N$ is connected by $w(e)$ edges (v_N, e_N) to a vertex $e_N \in E_N$ iff corresponding vertex $v \in V$ is an endpoint of $e \in E$. Capacity of every one such edge (v_N, e_N) is 1, cost function is 0. The source vertex s is connected to every vertex $v_N \in V_N$ by the edge (s, v_N). The capacity of the edge (s, v_N) is $c + (degree(v) - c)\epsilon$ where ϵ is a small positive constant. The cost function of the edge (s, v_N) is $c_1(x)$, where $c_1(x) = 0$ if $x = 0$, $c(1 + \epsilon)$ otherwise. Every vertex $e_N \in E_N$ is connected to the sink vertex t by $w(e)$ edges. The capacity of any one such edge (e_N, t) is $1 + \epsilon$. The cost function of the edge (e_N, t) is $c_2(x)$, where $c_2(x) = 0$ if $x < 1 + \epsilon$, $-1 - \epsilon$ otherwise. See Figure 4–5 and Figure 4–6.

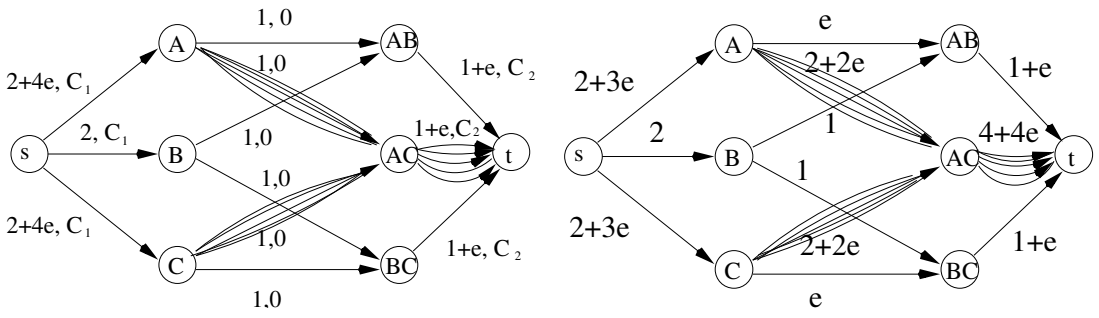


Figure 4–7: Network and a flow for p=3

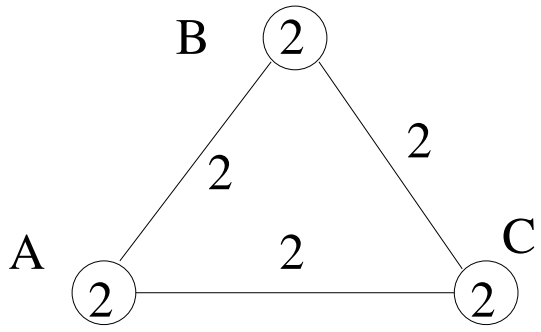


Figure 4–8: Graph corresponding to Figure 4–7

Lemma 13 *Let N be the network that corresponds to the graph G, constructed as described above. Let $b()$ be the following demand vector defined on the vertices of N, $b(s) = -b(t) \leq L, b(v) = 0, v \neq s, t$, where $L = 2p(1+\epsilon)$. Wlog dimension-dependent constant $K = 0$. Then there is a non-empty dense graph $A \subseteq G, |A| \leq p$ iff there exists a non-zero feasible flow in N (i.e it satisfies the demand vector $b()$ and the capacities of all the edges of N) of cost 0 or less, for some demand vector $b()$.*

Example. Consider Figure 4–7. On the left is a network N corresponding to the graph G on the right of of Figure 4–8 ($c = 2$). On the right of Figure 4–7 is a flow of cost zero corresponding to the $p = 3$ i.e the dense subgraph ABC. This flow is of cost zero because costs on the right (AB,BC,AC-t) cancel the costs on the left (s-A,B,C). (Note that costs of edges (AB,t), (AC,t), (BC,t) are negative, but they could be converted into positive one by changing directions of the arcs and their capacities). The way the network N is designed is such that any flow of cost zero will correspond to a feasible dense subgraph and vice versa. A Figure 4–9 shows a

flow of cost zero corresponding to the $p = 2$ i.e the dense subgraph AC. For $p = 1$ there are no flows of cost zero since there are no dense subgraphs of size 1.

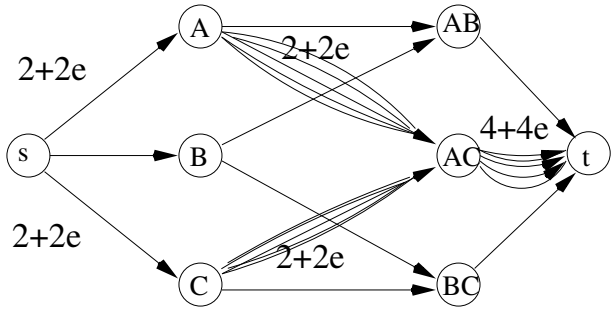


Figure 4–9: Flow for p=2

The network N given above is just one possible reduction of minimum dense graph to minimum cost flow. For example other cost functions are possible: for an edge (s, v_N) cost function could be $c_1(x) = 0$ if $x = 0$ otherwise $c_1(x) = 2 + (degree(v) - 2)\epsilon - x$, and for an edge (e_N, t) cost function could be $c_2(x) = 0$ if $x = 0$ otherwise $1 + \epsilon - x$. Then there is a non-empty dense graph $A \subseteq G, |A| \leq p$ iff there exists a non-zero feasible flow in N (i.e it satisfies the demand vector $b()$ and the capacities of all the edges of N) of cost less than 1, for some demand vector $b()$.

Converting min cost flow problem into mixed-integer problem. The problem of finding a minimum cost flow in a network described in the previous section can be stated as a following optimization problem OPT:

$$\min \sum_{i=1}^{n+3m} f_i(x_i)$$

s.t

$$Ax = b, 0 \leq x \leq u$$

where i such that $1 \leq i \leq n$ denotes "left" edges in the bipartite network, i such that $m+1 \leq i \leq n+2m$ denotes "middle edges in the bipartite network and i such that $n+2m+1 \leq i \leq n+3m$ denotes "right" edges in the bipartite network. Here n and m is number of vertices and edges of the original weighted graph (and accordingly number of vertices on the left and on the right of the corresponding network). For notational simplicity we assume that all edges have weight 1 and all vertices have weights c.

Variable x_i represents flow across an edge i, vector u represents capacity constraints described in the previous section. Matrix A and vector b describe the usual conservation/demand constraints.

Function

$$f_i(x_i) = c_1(x_i), m+1 \leq i \leq n+2m$$

$$f_i(x_i) = 0, m+1 \leq i \leq n+2m$$

$$f_i(x_i) = c_2(x_i), n+2m+1 \leq i \leq n+3m$$

where $c_1(), c_2()$ are the cost functions described in the previous section.

At this stage function that is to be optimized in OPT involves step functions. These step functions can be replaced by additional variables with integrality constraints.

Let $\forall i = 1, \ldots, n$ variable $y_i = 1$ if $x_i > 0$ and $y_i = 0$ otherwise.

Let $\forall i = n+2m+1, \ldots, n+3m$ variable $y_i = 1$ if $x_i = 1+\epsilon$ and $y_i = 0$ otherwise.

The OPT can be restated as

$$\min \sum_{i=1}^{n} c(1+\epsilon)y_1 + \sum_{i=n+2m+1}^{n+3m} -(1+\epsilon)y_i$$

s.t

$$Ax = b$$

$$0 \leq x_i \leq u_i y_i, \forall i = 1, \ldots, n$$

$$0 \leq x_1 \leq 1, \forall i = n+1, \ldots, n+2m$$

$$(1+\epsilon)y_i \leq x_i \leq (1+\epsilon), \forall i = n+2m+1, \ldots, n+3m$$

$$y_i \in \{0, 1\}$$

Open question: how to use this formulation in order to construct FPAS for the original minimum dense problem?

4.4.3 Stating Minimum Dense Problem as IP

In order to simplify the notation we assume for this section that all the vertices have weight 3, all edges have weight 1.

K - density constant, P -size of the graph

Version 1 of minimum dense problem stated as IP:

$$min \sum x_i$$

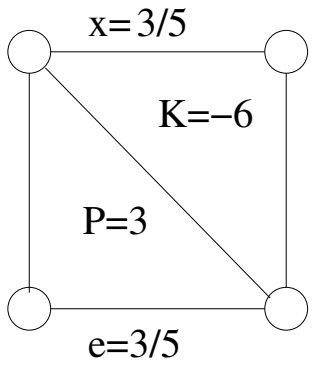
$$\sum e_{ij} - \sum 3x_i \geq K$$

$$x_i \geq e_{ij}, \forall i, j$$

$$\sum e_{ij} = 3P + K$$

$$x_i, e_{ij} \in 0, 1$$

LP relaxation is not always integral, consider Figure 4.4.3.



Version 2 :

$$max \sum e_{ij} - \sum x_i$$

$$\sum x_i = P$$

$$x_i \geq e_{ij}, \forall i, j$$

$$x_i, e_{ij} \in 0, 1$$

LP relaxation is not always integral, again consider Figure 4.4.3.

Several other IP statements of the minimum dense problem are given below, essentially using density condition, size of graph condition and the fact that if the edge is chosen to be in a subgraph, then both of it endpoints should also be in this subgraph. None of their LP relaxations are guaranteed to be integral.

Open question - how to state minimality condition in IP form? (perhaps via cuts?). This might lead to much better IP formulations.

Version 3:

$$max \sum e_{ij}$$

$$x_i \geq e_{ij}, \forall i, j$$

$$\sum x_i \leq P$$

$$\sum e_{ij} - \sum 3x_i \geq K$$

Version 4:

$$min \sum e_{ij}$$

$$\sum x_i \geq P$$

$$x_i \geq e_{ij}, \forall i, j$$

$$\sum e_{ij} - \sum 3x_i \geq K$$

Version 5:

$$max \sum e_{ij}$$

$$\sum e_{ij} + \sum x_i \leq E - 2P$$

$$\sum x_i \leq P$$

Version 6:

$$min \sum e_{ij}$$

$$\sum e_{ij} + 3 \sum x_i \geq 3E$$

$$x_i + e_{ij} \geq 1$$

$$\sum x_i \geq N - P$$

Version 7:

$$min \sum e_{ij}$$

$$\sum e_{ij} - 3 \sum x_i \geq 0$$

$$x_i \geq e_{ij}$$

$$\sum x_i \geq P$$

Semidefinite approach. An IP formulation, say

$$min \sum_{i=1}^{n} x_i$$

$$\sum x_i = P$$

$$\sum e_{ij} - 3 \sum x_i = K$$

$$x_i - e_{ij} = 0$$

can be easily converted into a semidefinite program

$$max 0.25(1 + x_{i0} + x_{j0} + x_{ij})$$

$$\sum x_{i0} = P$$

$$\sum x_{ij} = K - 2P$$

$$x_{i0} - x_{ij} = 0$$

$$x_{ii} = 1$$

matrix X is symmetric and positive definite

Standard approach Ye and Zhang (1997) is to solve semidefinite programming relaxation above , choose a random unit vector in a n-dimensional unit sphere and select P vertices closest to this vector.

Unfortunately it possible that this solution corresponds to a subgraph that is not dense (in Ye and Zhang (1997) we need to find a subgraph of size P that has largest number of vertices, so feasibility of solution is guaranteed. For our problem feasibility, i.e subgraph being dense, is not guaranteed).

REFERENCES

Ait-Aoudia, S., Jegou, R., Michelucci, D. (1993). Reduction of constraint systems. In *Compugraphics*, **25**, pages 83–92.

Asahiro, Y.,Iwama, K. (1995). Finding Dense Subgraphs. In *LNCS* **1004**, pages 102-112.

Blum, L., Shub, M., Smale, S. (1989). On a theory of computation and complexity over the real numbers: Np-completeness, recursive functions and universal machines. In *Bulletin of the Amer. Math. Soc*, **21**, pages 1-46.

Bouma, W., Fudos, I., Hoffmann, C., Cai, J., Paige, R. (1995). A geometric constraint solver. In *Computer Aided Design*, **27**, pages 487-501.

Bronsvoort, W., Jansen, F. (1994). Multiview feature modeling for design and assembly. Shah, J., Mantyla, M., editors, *Advances in Feature Based Modeling*, London.

Canny, J. (1993). Improved algorithms for sign determination and existential quantifier elimination. In *Computer Journal*, **36**, pages 1-5.

Canny, J., Emiris, I. (1993). An efficient algorithm for the sparse mixed resultant. G. Cohen, T. Mora editors, *Proc. 10th Intern. Symp. on Applied Algebra, Algebraic Algorithms, and Error Correcting Codes*, **263**, pages 89-104.

Chou, S. C., Gao, X. S., Zhang, J. Z. (1996). A method of solving geometric constraints. Technical report, Wichita State University, Dept. of Computer Science.

Collins, G. (1975). Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *LNCS* , **33**, pages 134-183.

Cox, D., Little, J., O'Shea, D. (1998). *Using Algebraic Geometry.* Springer-Verlag, Berlin.

Crippen, G., Havel, T. (1988). *Distance Geometry and Molecular Conformation.* John Wiley & Sons, London.

Durand, C. (1998). *Symbolic and Numerical Techniques for Constraint Solving.* PhD dissertation, Purdue University.

Fang, S. (1992). *Robustness in Geometric Modeling.* PhD dissertation, University of Utah.

Feige, U., Seltzer, M. (1997). *On the densest k-subgraph problem.* Technical report CS97-16, Tel-Aviv University, Dept. of Computer Science.

Frankl, P. (1982) An extremal problem for two families of sets. In *European J. Combinatorics*, **3**, pages 125-127.

Fudos, I. (1995). *Geometric Constraint Solving.* PhD thesis, Purdue University.

Fudos, I., Hoffmann, C. M. (1996a). Constraint-based parametric conics for CAD. In *Computer Aided Design*, **28**, pages 91-100.

Fudos, I., Hoffmann, C. M. (1996b). Correctness proof of a geometric constraint solver. In *Intl. J. of Computational Geometry and Applications*, **6**, pages 405-420.

Fudos, I., Hoffmann, C. M. (1997). A graph-constructive approach to solving systems of geometric constraints. In *ACM Trans on Graphics*, **13**, pages 179-216.

Gabow, H., Westermann, H. (1988). Forests, frames and games: Algorithms for matroid sums and applications. In *Proceedings of the twentieth annual ACM symposium on theory of computing*, **20**, pages 112-122.

Gallo, G., Grigoriadis, M., Tarjan R. (1989). A fast parametric maximum flow algorithm and applications. In *SIAM J. Computing*, **18**, pages 30-55.

Gao, X., Chou, S. (1998a). Solving geometric constraint systems. Part I. A global propagation approach. In *Computer Aided Design*, **30**, pages 47-54.

Gao, X., Chou, S. (1998b). Solving geometric constraint systems. Part II. A symbolic approach and decision of rc-constructibility. In *Computer Aided Design*, **30**, pages 115-122.

Goemans (1996). Mathematical programming and approximation algorithms. *Lecture at Udine School, Udine, Italy, September 1996.*

Graver, J., Servatius, B., Servatius, H. (1993). *Combinatorial Rigidity.* Graduate Studies in Mathematics, London.

Grigor'ev, D. Y., Vorobjov, N. N. (1988). Solving systems of polynomial inequalities in subexponential time. In *Journal of Symbolic Computation*, **5**, pages 37-64.

Hastad, J. (1996). Clique is hard to approximate within $n^{1-\epsilon}$. *Proceedings of Foundations of Computer Science*, IEEE press, Baltimore.

Havel, T. (1991). Some examples of the use of distances as coordinates for Euclidean geometry. In *J. of Symbolic Computation*, **11**, pages 579-594.

Hendrickson, B. (1992). Conditions for unique graph realizations. In *SIAM J. Comput.*, **21**, pages 65-84.

Hoffmann, C. M. (1997). Solid modeling. Goodman, J. E., O'Rourke, J., editors, *CRC Handbook on Discrete and Computational Geometry*, CRC Press, Boca Raton, FL.

Hoffmann, C. M., Joan-Arinyo, R. (1998). CAD and the product master model. In *Computer Aided Design*, **22**, pages 905-918.

Hoffmann, C. M., Lomonosov, A., Sitharam, M. (1997). Finding solvable subsets of constraint graphs. *Constraint Programming '97*, Linz, Austria.

Hoffmann, C. M., Lomonosov, A., Sitharam, M. (1998). Geometric constraint decomposition. Bruderlin, editor, *Geometric Constraint Solving*. Springer-Verlag, Berlin.

Hoffmann, C. M., Peters, J. (1995). Geometric constraints for CAGD. Daehlen, M., Lyche, T., Schumaker, L., editors, *Mathematical Methods for Curves and Surfaces*, Vanderbilt University Press, Baltimore.

Hoffmann, C. M., Rossignac, J. (1996). A road map to solid modeling. In *IEEE Trans. Visualization and Comp. Graphics*, **2**, pages 3-10.

Hoffmann, C. M., Vermeer, P. J. (1994). Geometric constraint solving in R^2 and R^3. Du, D. Z., Hwang, F., editors, *Computing in Euclidean Geometry*, World Scientific Publishing, Singapore.

Hoffmann, C. M., Vermeer, P. J. (1995). A spatial constraint problem. *Workshop on Computational Kinematics*, INRIA Sophia-Antipolis, Paris.

Hopcroft, J. E., Tarjan, R. E. (1973). Dividing a graph into triconnected components. In *SIAM J. Comput.*, **2**, pages 135-158.

Hsu, C. (1996). *Graph-based approach for solving geometric constraint problems.* PhD dissertation, University of Utah.

Imai, H. (1985). On combinatorial structures of line drawings of polyhedra. In *Discrete Applied Mathematics*, **10**, pages 79-92.

Itai, A., Rodeh, M. (1978). Finding a minimum circuit in a graph. In *SIAM J. Comput*, **7**, pages 413-423.

Karger, D. (1996). Minimum cuts in near-linear time. In *STOC 1996*, **12**, pages 32-47.

Khovanskii, A. (1978). Newton polyhedra and the genus of complete intersections. In *Funktsional'nyi Analiz i Ego Prilozheniya*, **12**, pages 51-61.

Klein, R. (1996). Geometry and feature representation for an integration with knowledge based systems. *Geometric modeling and CAD.* Chapman-Hall, New York.

Klein, R. (1998). The role of constraints in geometric modeling. Bruderlin, editor, *Geometric constraint solving and applications*, Springer-Verlag, Berlin.

Kortsarz, G., Peleg, D. (1993). On choosing a dense subgraph. In *IEEE 34th FOCS*, **12**, pages 692-701.

Kraker, K., Dohmen, M., Bronsvoort, W. (1997). Maintaining multiple views in feature modeling. In *ACM/SIGGRAPH Symposium on Solid Modeling Foundations and CAD/CAM Applications*, **11**, pages 123–130.

Kramer, G. (1992). *Solving Geometric Constraint Systems.* MIT Press, Boston.

Laman, G. (1970). On graphs and rigidity of plane skeletal structures. In *J. Engrg. Math.*, **4**, pages 331-340.

Latham, R., Middleditch, A. (1996). Connectivity analysis: a tool for processing geometric constraints. In *Computer Aided Design*, **28**, pages 917-928.

Lazard, D. (1981). Résolution des systèmes d'équations algébriques. In *Theoretical Computer Science*, **15**, pages 77-110.

Lazard, D. (1991). A new method for solving algebraic systems of positive dimension. In *Discrete Applied Mathematics*, **33**, pages 147-160.

Mantyla, M., Opas, J., Puhakka, J. (1989). Generative process planning of prismatic parts by feature relaxation. In *Advances in Design Automation, Computer Aided and Computational Design*, **7**, pages 49–60.

Middleditch, A., Reade, C. (1997). A kernel for geometric features. In *ACM/SIGGRAPH Symposium on Solid Modeling Foundations and CAD/CAM Applications*, **17** , pages 91–97.

Newell, M., Evans, D. (1976). Modeling by computer. In *IFIP Working Conference on CAD systems*, North-Holland, Austin.

Owen, J. (1991). Algebraic solution for geometry from dimensional constraints. In *ACM Symp. Found. of Solid Modeling*, North Holland, Austin.

Owen, J. (1996). Constraints on simple geometry in two and three dimensions. In *International Journal of Computational Geometry and Applications*, **6**, pages 421-434.

Pabon, J. (1993). Modeling method for sorting dependencies among geometric entities. In *US States Patent 5,251,290.*

Renegar, J. (1992). On the computational complexity and the first order theory of the reals, part i. In *Journal of Symbolic Computation*, **13**, pages 255-299.

Ruiz, O. E., Ferreira, P. M. (1996). Algebraic geometry and group theory in geometric constraint satisfaction for computer-aided design and assembly planning. In *IIE Transactions on Design and Manufacturing*, **28**, pages 281-294.

Saliola, F., Whiteley, W. (1999). Constraint configurations in CAD: circles, lines and angles in the plane. *Preprint York University.*

Semenkov, O. (1976). An experimental CAD/CAM system. In *3rd international IFIP/IFAC conference on Programming Languages for Machine Tools*, North-Holland, Stirling, Scotland.

Serrano, D. (1990). Managing constraints in concurrent design: first steps. *Proc. Computers in Engineering*, MIT press, Boston.

Serrano, D., Gossard, D. (1986). Combining mathematical models with geometric models in cae systems. *Proc. Computers in Engineering*, University of Chicago Press, Chicago.

Sridhar, N., Aggarwal, R., Kinzel, G. (1993). Active occurence matrix based approach to design decomposition. In *Computer Aided Design*, **25**, pages 500-512.

Sridhar, N., Aggarwal, R., Kinzel, G. (1996). Algorithms for the structural diagnosis and decomposition of sparse, underconstrained, systems. In *Computer Aided Design*, **28**, pages 237-249.

Srivastav, A., Wolf, K. (1998). Finding dense subgraphs with semidefinite programming. In *Lecture Notes in Computer Science*, **1444**, pages 181–191.

Sturmfels, B. (1993). Sparse elimination theory. *Proc. Computational Algebraic Geometry and Commutative Algebra*, Cambridge University Press, Cambridge, UK.

Sugihara, K. (1985). Detection of structural inconsistency in systems of equations with degrees of freedom and its applications. In *Discrete Applied Mathematics*, **10**, pages 297-312.

Tay, T. (1999). On the generic rigidity of bar frameworks. In *Advances in Applied Math.*, **23**, pages 14-28.

Tay, T., Whiteley, W. (1985). Generating isostatic frameworks. In *Topologie Structurale*, **11**, pages 21-69.

Wang, D. (1993). An elimination method for polynomial systems. In *J. Symbolic Computation*, **16**, pages 83-114.

Whiteley, W. (1992). Matroids and rigid structures. *Matroid Applications*, Cambridge University Press, Cambridge, UK.

Whiteley, W. (1997). Rigidity and scene analysis. *Handbook of Discrete and Computational Geometry*, CRC Press, Boca Raton.

Y. Ye and J. Zhang (2001). Approximation of dense-$n/2$-subgraph and the complement of min-bisection. *Journal of Global Optimization*, **25**, pages 55–73.

BIOGRAPHICAL SKETCH

I was born in Moscow, Russia. I attended the Mathematics Department of Moscow State University in 1989-1991. I attended Kent State University, Ohio, as an undergraduate in 1991-1995 and graduated in Spring of 1995 with a bachelor's degree in mathematics and a bachelor's degree in computer science. I continued my graduate studies in Kent State University until 1997 and worked on a joint project in Purdue University in 1997-1998. In 1998 I transferred to the University of Florida where I have been working on my PhD in computer science.