Graph-Theoretic and Geometric Algorithms Associated with Moment-Based Polygon Reconstruction

by

Stephane Durocher

B.Sc. (Computer Science with Mathematics Major)

University of Toronto, 1997

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

we accept this thesis as conforming
to the required standard

The University of British Columbia

August 1999

Abstract

The problem of reconstructing an unknown polygonal shape P, viewed as a region in the complex plane, from a finite number of its complex moments is motivated by a number of mathematical, computational, and application-oriented considerations. Complex moment information can be derived from such diverse physical processes as tomographic (line integral) measurement, exterior gravitational, or magnetic field measurement, or thermal radiation measurement, associated with an otherwise unspecified object [GMV99, MVKW95, SB86]. Milanfar *et al.* [MVKW95], building on earlier work of Davis [Dav64, Dav77], show how a finite number of complex moments of P can be effectively used to reconstruct the vertices of P. If sufficient additional information (for example, convexity or rectilinearity) about P is known, P itself can be reconstructed. Very recently, Golub *et al.* [GMV99] address some of the formidable numerical difficulties associated with this reconstruction. In addition, they demonstrate that partial information concerning a polygon's edges can also be derived from a finite number of its complex moments, raising the possibility of a more complete and general reconstruction scheme. In general, note that even simply-connected polygonal regions are not uniquely specified by even infinitely many of their complex moments [SB86], thereby precluding a completely general reconstruction scheme.

Given vertex positions for the set V of vertices of P in the Cartesian plane and partial information (expressed in terms of geometric constraints) about the edges bounding P, we examine the problem of constructing polygonal regions consistent with this information. Although in practice, the vertex positions and edge constraints may contain error in their specifications (due either to error originating in the data or introduced in the numerical computations) it is of interest to study the reconstruction problem with error-free moment information. As we shall see, this translates to a constrained directed 2-factor problem in a directed graph G defined on the vertex set V.

As observed in [GMV99], the potential edges incident on a vertex are subject to very specific local restrictions. These restrictions have the following simple geometric interpretation. Each vertex has two independent axes assigned to it: a blue and a red axis. Each axis specifies two in-directions and two out-directions such that in-directions and out-directions are orthogonal. An edge joining vertex u to vertex v belongs to the edge set of G if and only if its direction agrees with one of the axes specified at each of u and v. We will colour the head and tail of each such edge with the colour of its associated axis. A solution to the polygon reconstruction problem is a spanning subgraph H of the resulting directed and edge-bicoloured graph in which every vertex has both in-degree and out-degree one and both red-degree and blue-degree one. Thus, we are looking for a directed 2-factor of G that satisfies some local colour constraints.

This thesis addresses graph-theoretic and geometric concerns arising from moment-based polygon reconstruction. We show **NP**-hardness of various restrictions to the 2-factor problem. We investigate properties of moment sets whose graphs contain non-unique solutions. We develop algorithms to solve the problem of polygon reconstruction given both perfect and imperfect input moment data. Finally, we examine the success of reconstruction on various classes of graphs.

Contents

List of Figures

Acknowledgements

I am indebted to several individuals for their extensive contributions in time, ideas, suggestions, and encouragement over the last few years.

Professor Faith Fich started me on the road to graduate research in computer science at the University of Toronto. As a professor, she sparked my interest in theory and, as a supervisor, she introduced me to the world of research. Professor Nicholas Pippenger welcomed me to the University of British Columbia and initiated our research on parallel game trees and optimal routing policies. Professor Jim Varah is responsible for introducing the problem of polygon reconstruction from moments. His continued interest, enthusiasm, and involvement, as well as his patient explanations of the numerical mechanics of reconstruction were crucial in the development of this research. He kindly agreed to be a second reader for this thesis. Martin Robillard and Michael McAllister generously acted as proofreaders, both providing very helpful reviews.

The big thanks go out to Professor David Kirkpatrick. He provided essential guidance and motivation, shared his innumerable ideas, and gave up endless hours of his time, day after day and month after month, allowing this thesis and all our work on polygon reconstruction to come into being. His role as a supervisor, a role model to graduate students, and an all-around great guy and friend is very much appreciated. I consider myself very lucky to have had the opportunity to work closely with such a great individual.

STEPHANE DUROCHER

The University of British Columbia

August 1999

à mes chers parents,

Yves et Christiane Durocher

Chapter 1

Introduction

1.1 Problem and Motivation

The problem of reconstructing an unknown polygonal shape, P, viewed as a region in the Cartesian plane, from a finite number of its complex moments is motivated by a several mathematical, computational, and application-oriented considerations. Moment information derived from such diverse physical processes as tomographic (line integral) measurements, exterior gravitational, or magnetic field measurements, or thermal radiation measurements often provides the input data in reconstructing a two-dimensional object. In each case, these measurements represent available information associated with an otherwise unspecified object. This complex moment data potentially provides enough information to allow the complete or partial reconstruction of the original polygon, P [Dav64, Dav77, SB86, MVKW95, GMV99].

1.2 Previous Work

For years, mathematicians and physicists have studied the relationship between shape and moment [ST43]. The moments of a region are the integrals of the powers of the independent variables over that region. The kth harmonic moment of a

2-dimensional polygon P is given by [GMV99]:

$$m_k = \int\int_P z^k \; dx \; dy \tag{1.1}$$

In early work, Davis showed that given only its complex moments up to the third order, the positions of all vertices of a triangle within the Cartesian plane can be successfully reconstructed [Dav64]. Thus, any three-vertex polygon can be reconstructed from its complex moments.

Davis later proved a generalization of an earlier result by Motzkin and Schoenberg, a quadrature formula that provides the essential mathematical basis to polygon reconstruction from moments [Dav77, GMV99]:

Theorem 1.1 (Davis, 1964) *Let $z_1, \ldots, z_n$ designate the vertices of a polygon P in the Cartesian plane. Then we can find constants $a_1, \ldots, a_n$ depending upon $z_1, \ldots, z_n$ but independent of f, such that for all f analytic in the closure of P:*

$$\int\int_P f''(z) \; dx \; dy = \sum_{j=1}^{n} a_j f(z_j) \tag{1.2}$$

This formula allowed Milanfar *et al.* [MVKW95] to generalize Davis' result and show how a finite number of complex moments of P can be used effectively to reconstruct the vertices of P.

Using their method, given complex moments for a polygon P within the Cartesian plane, one may reconstruct the positions of the vertices of P. Given sufficient additional information about the interconnection of vertices within P, such as convexity, one may reconstruct P itself. For n vertices, there are $n(n-1)/2$ possible edges. Thus, without additional information about the edge set of P, vertex position alone may be insufficient for complete reconstruction (see example in figure 1.1).

Very recently, Golub *et al.* [GMV99] address some of the formidable numerical difficulties associated with this reconstruction. In addition, they demonstrate that partial information concerning a polygon's edges can also be derived from a

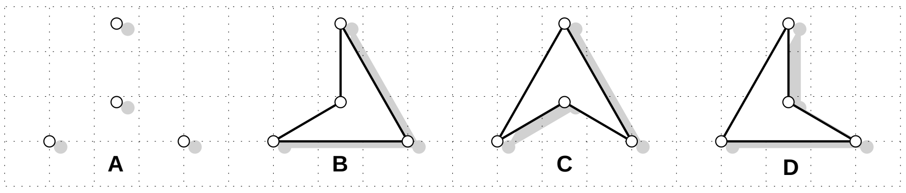

Figure 1.1: Even with as few as four non-convex points, given only vertex positions without additional information, three different polygons may be reconstructed.

finite number of its complex moments, raising the possibility of a more complete and general reconstruction scheme. This further development not only allows the reproduction of original vertex positions, but also provides edge orientation constraints between vertices of P (see example in figure 1.2).

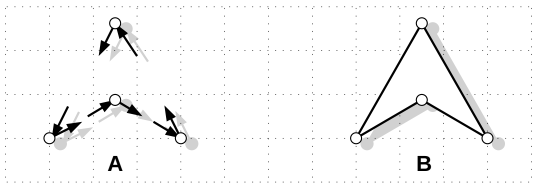

Figure 1.2: Given the same four vertex positions along with possible edge directions at every vertex, only a single solution remains possible.

Specifically, given vertex positions for the set V of vertices of P in the Cartesian plane, for every $v \in V$, we may derive two angles, ϕ_1 and ϕ_2, each allowing in-edges at angles $\phi_1 \bmod \pi$ and $\phi_2 \bmod \pi$, and out-edges at angles $\phi_1 \bmod \pi + \frac{\pi}{2}$ and $\phi_2 \bmod \pi + \frac{\pi}{2}$ [GMV99]. This smaller set of potential edges greatly restricts the choice of edges in reconstructing P. In so doing, this additional extracted information constrains the edge reconstruction problem, allowing reconstruction of non-convex shapes from moments.

In general, even simply-connected polygonal regions are not uniquely specified by even infinitely many of their complex moments [SB86], thereby precluding a completely general reconstruction scheme. Since there exist distinct two-dimensional polygons whose moment signatures are identical (see Chapter 4) reconstruction from moments is limited, independent of any reconstruction approach. In cases involving ambiguity, differentiating between possible reconstructed solutions requires additional information, aside from the input complex moments.

Having applied this initial phase of the solution, the reconstruction problem remains far from being solved. The numerical methods provide positions for vertices of P and partial information about potential bounding edges of P, expressed in terms of geometric constraints. We examine the problem of constructing polygonal regions consistent with this information. For many non-convex polygons, this latter phase of the reconstruction process becomes quite convoluted. The problem involves interconnecting reconstructed vertices with a superset of potential edges and applying constraint propagation along neighbouring vertices in search of a polygonal subset of edges. Although, in practice, the vertex positions and edge constraints may contain error in their specifications, due either to error originating in the moment data or error introduced in the numerical computations, it is of interest to study the reconstruction problem with error-free moment information. As we shall see, within a graph-theoretic setting, this formulation translates directly into a restricted directed 2-factor problem on a directed graph G defined on the vertex set V. We examine the problem of reconstructing polygonal regions, or 2-factors, consistent with the derived partial information about vertex positioning and edge orientation.

1.3 Thesis Organization

The organization of the material in this thesis is as follows. Chapter 1 introduces the polygon reconstruction problem, gives motivation for its examination, and presents a brief discussion of related research. Chapter 2 provides general background and definitions from graph theory, complexity, and polygon reconstruction. Chapter 3 discusses the 2-factor problem along with proofs that restrictions to the 2-factor problem specific to our polygon reconstruction setting are **NP**-hard. Chapter 4 deals with properties of various graphs with respect to solutions to the reconstruction problem and discusses the uniqueness of solutions within specific graphs. Chapter 5 explains the various modules of the reconstruction algorithm. Chapter 6 examines the behaviour of the algorithm on various classes of graphs and challenges faced by the algorithm. Chapter 7 presents a list of potential future research areas which remain open within the domain of polygon reconstruction from moments. Finally, Chapter 8 summarizes the formal discussion of the polygon reconstruction problem.

Chapter 2

Problem Definition

Before beginning our examination of the problem, we define terms that arise in general graph theory, related complexity theory, common problems found in graph-theoretic and classical complexity, and terms that arise in our discussion of polygon reconstruction. Finally, we formally define the problem of reconstructing the polygon P from its complex moments.

2.1 Definition of Terms

2.1.1 General Graph Theory

The following terms[1] arise in many general graph theory problems.

A **graph**, $G = (V, E)$, is a collection of vertices, V, and edges, E. A **vertex**, $v \in V$, is a single point or node in G. An **edge**, $e \in E$, represents a relationship between two vertices in G. We say edge $uv \in E$ if and only if there exist vertices $u, v \in V$ such that u and v are related under some relation $R \subseteq V \times V$. Graphically, we represent e by drawing a straight[2] line from u to v. If the edge relation is asym-

[1]See the glossary in Appendix A for a complete list of definitions.
[2]Within our domain, all edges are assumed to be straight lines.

metric, then G is a **directed graph** or **digraph**. uv and vu are distinct possible edges between vertices v and u. Given graphs $G = (V, E)$ and $H = (V', E')$, H is a **subgraph** of G if and only if H is a graph, $V' \subseteq V$, and $E' \subseteq E$. If $V' = V$ then H is a **spanning** subgraph of G. An edge coming from a vertex u into vertex v is called an **in-edge** locally at v. Conversely, an edge going from vertex v out to a vertex u is described as an **out-edge** locally at v.

We represent vertex adjacency through a boolean **adjacency matrix**, M, where $M_{u,v} = true$ if and only if there exists an edge from vertex u to vertex v. The **degree** of a vertex v is the total number of edges incident upon v. Similarly, the **in-degree** of v is the number of in-edges at v and the **out-degree** of v is the number of out-edges at v. The degree of a graph, G, is defined as the maximum degree amongst all vertices $v \in V$.

If we are given a pre-determined fixed positioning for the vertices of a graph G, then we say that G is an **embedded** graph. Any fixed positioning of the vertices of G is described as an **embedding** of G. If there exists a two-dimensional embedding of graph G in the plane in which none of the edges of G cross, then we say G is **planar**. Such an embedding is a **non-crossing** embedding. All of the edges in a **rectilinear** embedding lie along a single orthogonal north-south, east-west axis.

A graph G is **simply-connected** if for any two vertices $u, v \in V$ we can find some sequence of vertices $\{v_1, \ldots, v_k\} \subseteq V$ such that $v_1 = u$, $v_k = v$, and for all $1 \leq i \leq k - 1$, $v_i v_{i+1} \in E$ or $v_{i+1} v_i \in E$. A **connected component** $G' \subseteq G$ is a maximal simply-connected subgraph of G. There cannot exist vertices $v \in V - V'$ and $u \in V'$ such that $uv \in E$ or $vu \in E$. A **cycle** is a sequence of neighbouring vertices, $\{v_1, \ldots, v_k\} \subseteq V$, such that $v_i v_{i+1} \in E$ for all $1 \leq i \leq k - 1$ and $v_k v_1 \in E$. A **k-cycle** is a cycle of length k.

A vertex v is an **interior** vertex within an embedded graph G, if there exists a cycle $C \subset G$ such that $v \notin C$ and C encloses v within it. Conversely, v is an **exterior** vertex, if there does not exist such a cycle C.

A graph $G = (V, E)$ is **bipartite** if there exists a partition of its vertices, $V = V_1 \cup V_2$, such that, $V_1 \neq \emptyset$, $V_2 \neq \emptyset$, $V_1 \cap V_2 = \emptyset$, and every edge $v_1 v_2 \in E$ has one endpoint in each partition, $v_1 \in V_1$ and $v_2 \in V_2$ or $v_1 \in V_2$ and $v_2 \in V_1$.

2.1.2 Classical Complexity

As is often the case with the presentation of algorithms in theoretical computer science, the classes of problems **P** and **NP** find themselves present within our domain. Garey and Johnson [GJ79, page 27] formally define the class **P** as follows:

$$\begin{aligned} \mathbf{P} \; = \; \{ \; & L : \textit{there is a polynomial time deterministic} \\ & \textit{Turing machine program } M \textit{ for which } L = L_M \} \end{aligned}$$

Informally, the class **P** unites all problems that have deterministic polynomial-time algorithmic solutions. That is, their solutions can be found deterministically in time proportional to some polynomial function in terms of the input size of the problem.

Similarly, Garey and Johnson [GJ79, page 31] formally define the class **NP** as follows:

$$\begin{aligned} \mathbf{NP} \; = \; \{ \; & L : \textit{there is a polynomial time nondeterministic} \\ & \textit{Turing machine program } M \textit{ for which } L = L_M \} \end{aligned}$$

Informally, the class **NP** unites all problems that have nondeterministic polynomial-time algorithmic solutions. By these definitions, $\mathbf{P} \subseteq \mathbf{NP}$. Thus, we require a distinction between those problems that have known polynomial-time deterministic solutions and those that do not.

To compare the difficulty of two decision problems, we use a **reduction** function, R, which allows us to transform one problem to another. Papadimitriou writes:

> That is, we shall be prepared to say that problem A is at least as hard as problem B if B reduces to A. Recall what "reduces" means. We say that B reduces to A if there is a transformation R which, for every input x of B, produces an equivalent input $R(x)$ of A. Here by "equivalent" we mean that the answer to $R(x)$ considered as an input for A, "yes" or "no," is a correct answer to x, considered as an input of B. In other words, to solve B on input x we just have to compute $R(x)$ and solve A on it. [Pap94, page 159]

If we can show a polynomial-time reduction f from some problem q to a problem p, such that, $\forall x$, $x \in q \Leftrightarrow f(x) \in p$, then we say p is at least as hard as q. We define a special class of problems, the class of **NP-hard** problems. If q is **NP**-hard and q is reducible to p in polynomial time, then p is also **NP**-hard. This special class of **NP** problems have the property that, for any **NP**-hard problem q other than SAT (Section 2.1.3) there is another **NP**-hard problem p, such that, p is reducible to q in polynomial time. Thus, problem q is "at least as complex" as problem p. Therefore, **NP**-hardness is a lower bound of complexity in a hierarchy of problems for which no deterministic polynomial-time algorithmic solution is known.

2.1.3 Graph Theoretic and Complexity Problems

The following problems have been studied extensively in classical graph theory and complexity. These problems will be relevant in our examination of complexity issues with respect to the polygon reconstruction from moments problem.

As we will see in Chapter 3, our reconstruction problem is very closely related to the problem of finding an n-**factor** within a graph. Lovász and Plummer write, "a spanning subgraph regular of degree n is called an n-factor."[LP86, page xxx] N-FACTOR[3] is solvable in polynomial time [LP86, GJ79]. A **2-factor** is an n-factor of degree 2. A **directed 2-factor** of G is a spanning subgraph $H \subseteq G$ for which every vertex $v \in V'$ has exactly one in-edge and one out-edge. The problem of finding an unconstrained 2-factor in a general graph G is polynomial-time solvable [LP86, GJ79]. Closely related to the 2-factor is the **Hamiltonian cycle**. Lovász and Plummer define, "a cycle which includes every point of a graph G is called a Hamilton cycle of G."[LP86, page xxx] The problem HAMILTONIAN CIRCUIT is **NP**-hard [GJ79]. A **matching** is an n-factor of degree 1. Every vertex is met by exactly one edge. The problem of finding a matching in a general graph G is polynomial-time solvable [GJ79].

Within our reductions, we will refer to **3-dimensional matching**. Garey and Johnson explain 3-dimensional matching as follows:

> The 3-DIMENSIONAL MATCHING problem is a generalization of the classical "marriage problem": Given n unmarried men and n unmarried women, along with a list of all male-female pairs who would be willing to marry one another, is it possible to arrange n marriages so that polygamy is avoided and everyone receives an acceptable spouse? Analogously, in the 3-DIMENSIONAL MATCHING problem, the sets W, X, and Y correspond to *three* different sexes, and each triple in M corresponds to a *3-way marriage* that would be acceptable to all three participants. Traditionalists will be pleased to note that, whereas 3DM is **NP**-complete, the ordinary marriage problem can be solved in polynomial time. [GJ79, page 50]

Another family of problems that will be of interest is **SAT**. Papadimitriou defines, "SATISFIABILITY (or SAT, for short) then is the following problem: Given a Boolean expression ϕ in conjunctive normal form, is it satisfiable?"[Pap94, page 77] SATISFIABILITY is the best-known and, historically, the first **NP**-complete problem [GJ79]. Garey and Johnson write, "the **3-SATISFIABILITY** problem

[3]Formal descriptions of problems in complexity theory are traditionally given an identifier composed of a few brief descriptive words written in capital letters.

is just a restricted version of SATISFIABILITY in which all instances have exactly three literals per clause."[GJ79, page 48] 3-SAT is also **NP**-complete. **2-SAT** restricts instances to two literals per clause. This tighter restriction on the problem makes 2-SAT solvable in polynomial time [GJ79].

2.1.4 Polygon Reconstruction Problem

The following definitions are specific to our discussion of moment-based polygon reconstruction.

Given a graph $G = (V, E)$, at every vertex $v \in V$, we assign a single **colour** to each edge. We say G is an **edge-bicoloured** graph. This colouring is local; the colour at the head and tail of an edge may differ. Within our domain, a vertex may only have two different colours of edges; we refer to these locally as **red** and **blue**.[4] If every $v \in V$ contains no more than two red in-edges, two red out-edges, two blue in-edges, and two blue out-edges, then G is a **bounded-degree edge-bicoloured** graph. Furthermore, if edges of similar colour and direction align along an axis (see figure 2.1) then G is a **moment-derived edge-bicoloured** graph.

The angular constraints on potential edges, ϕ_1 and ϕ_2, impose an orthogonality between edges of similar colour. Thus, under a geometric interpretation of these constraints, edges of similar colour at a given vertex to belong to a common **framework** or **axis**. Each vertex has two such local axes with edges within an axis lying orthogonal to each other. We refer to a **global framework** whenever frameworks of common colour across all vertices in the graph lie within the same orthogonal axis. The **red-degree** of a vertex v is the total number of red edges incident upon

[4]We sometimes refer to **global colouring** in a graph, in which case, head and tail colours match for all edges $e \in E$. For presentation of graphs, we sometimes make use of a global colouring of edges. While locally, there are only two colours, globally, a graph may have more than two different colours.

v. Similarly, the **blue-degree** of v is the total number of blue edges at v. If a graph G has global colouring, then we may refer to the red-degree or blue-degree of G.

If a vertex v has at least one in-edge of a given colour and one out-edge of a different colour, then we say v is **valid**. If v does not have any in-edges, does not have any out-edges or only has edges of one colour, then we say v is **invalid**. When a vertex v has in-degree one, out-degree one, red-degree one, and blue-degree one, then we say v is **happy**. Otherwise, we say v is **unhappy**. If $H = (V', E')$ is a spanning subgraph of G with the property that every $v \in V'$ is happy, then H is a **solution** subgraph of G. We describe the state of H as **global happiness**.

Under the edge constraints of the reconstruction, a vertex v that is part of a solution may be in one of two **vertex states**: **red-in, blue-out** (RIBO) or **blue-in, red-out** (BIRO). This state directly corresponds to edge commitments. An edge e exists in one of three **edge states**: **undecided**, **committed**, or **dropped**. Whenever an edge is selected as being part of a partial solution, we say that we **commit** to that edge. We refer to this decision as an **edge commitment**.

A **fragment** is any sequence of vertices $v_1, \ldots, v_i$ such that $v_k v_{k+1} \in E$, for all $1 \leq k \leq i-1$ and and every vertex $v_2, \ldots, v_{i-1}$ is happy. A fragment constitutes a part of a possible solution. Several graphs embody more than one possible solution. Any such non-unique solution is an **ambiguous solution**.

2.2 Models Used and Problem Overview

The basis of all models we study will be that of an embedded graph in the Cartesian plane in which vertices are subject to very specific local restrictions; given vertex positions for the set V of vertices of P in the Cartesian plane, for every $v \in V$, we may derive two angles, ϕ_1 and ϕ_2, each allowing in-edges at angles $\phi_1 \bmod \pi$

and $\phi_2 \bmod \pi$, and out-edges at angles $\phi_1 \bmod \pi + \frac{\pi}{2}$ and $\phi_2 \bmod \pi + \frac{\pi}{2}$ [GMV99]. These restrictions have the following simple geometric interpretation. Each vertex has two independent axes assigned to it: a blue and a red axis. Each axis specifies two in-directions and two out-directions such that in-directions and out-directions within the axis are orthogonal (see Figure 2.1).

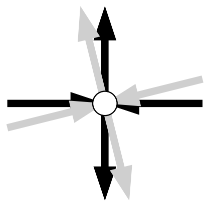

Figure 2.1: local blue and red axes for potential edges at a vertex

Within a solution, the reconstructed vertex positions form a set of vertices, V, for a graph $G = (V, E)$. An edge e from vertex u to vertex v belongs to the edge set of G if and only if its direction agrees with one of the axes specified at each of u and v. We will colour the head and tail of e with the colour of its associated axis. For example, say we have a graph G_1 with four vertices, $v_1, \ldots, v_4$, and each vertex

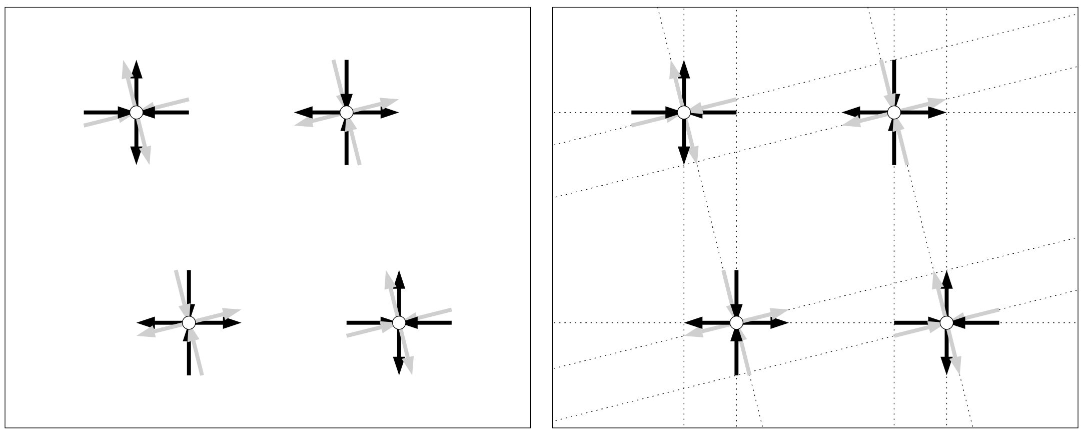

Figure 2.2: axes for potential edges

has red and blue axes given as in figure 2.2. The axes and their dotted extensions represent allowable edge directions.

Whenever the out-direction from vertex u aligns with the in-direction at another vertex v, we have an edge uv. Figure 2.3 displays the actual edge set E_1 that corresponds to this particular graph.

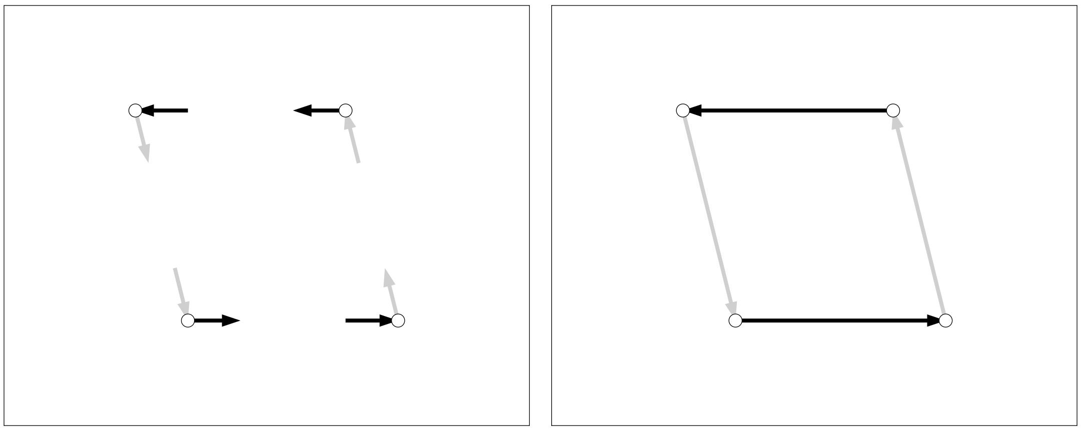

Figure 2.3: axes for actual edges

A solution to the polygon reconstruction problem is a spanning subgraph H of the directed and edge-bicoloured graph G in which every vertex has both in-degree and out-degree one and both red-degree and blue-degree one. Thus we are looking for a directed 2-factor of G that satisfies some local colour constraints, namely, global happiness.

At every vertex $v \in V$, a solution requires commitment to a single in-edge having some colour and a single out-edge of the opposite colour. Obviously, commitment at the head of an edge implies commitment at the tail and vice-versa.

For example, figure 2.4 displays a graph $G_2 = (V_2, E_2)$ for which we wish to find a globally-happy directed 2-factor. Every vertex $v \in V_2$ is valid. Several v, however, are unhappy. $H_2 = (V_2', E_2')$ is a spanning subgraph of G_2. Every $v \in V_2'$ is both valid and happy. Furthermore, H_2 represents a 2-factor of G_2 for which every

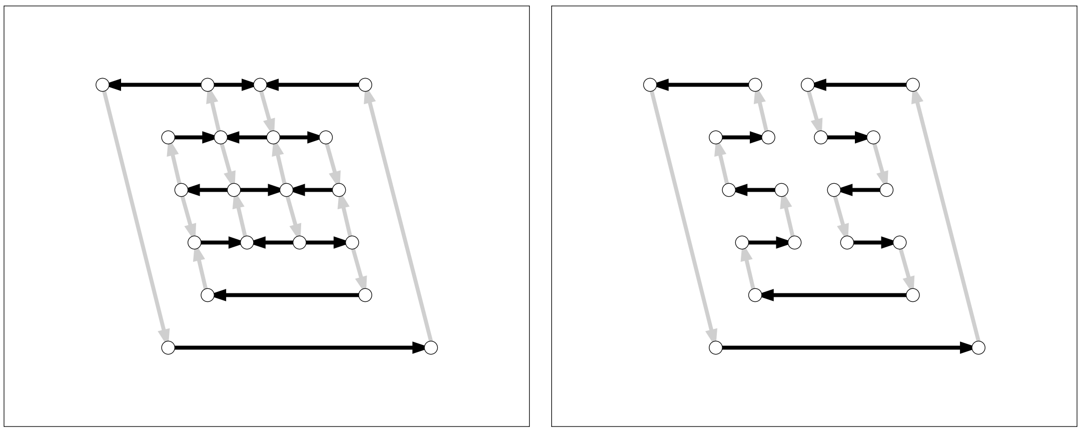

Figure 2.4: an initial graph G_2 and a solution subgraph H_2

vertex has both in-degree and out-degree one and both red-degree and blue-degree one. Thus, H_2 is a solution subgraph of G_2 for the polygon reconstruction problem. In this example, the edge set E_2 has two global frameworks: red and blue.

Whenever an odd-length cycle finds itself within a solution, then two colours are no longer sufficient for a representation of the graph with a global colouring. Since all vertices in a solution must be happy, at some point in any odd-length cycle, there must be an edge whose vertices do not agree on a global two-colouring (see figure 2.5a). Recolouring leaves the problem unchanged; we are searching for a globally-happy two factor within an edge-bicoloured graph. To facilitate the interpretation and presentation of such graphs, we introduce a new colour to allow global colouring. (see figure 2.5b).

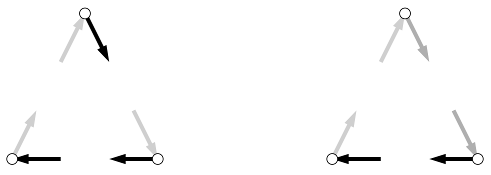

Figure 2.5: odd-length cycle edge colouring

Figure 2.6 displays a second example, a more involved graph, $G_3 = (V_3, H_3)$, and a solution subgraph, H_3. Unlike the previous graphs, in this case, edge set colouring can be made global only through the use of a third colour. Thus, G_3 has three global frameworks. Note that locally, however, every vertex $v \in V_3$ has exactly two colours of edges. Again, H_3 is a globally-happy directed 2-factor of G_3.

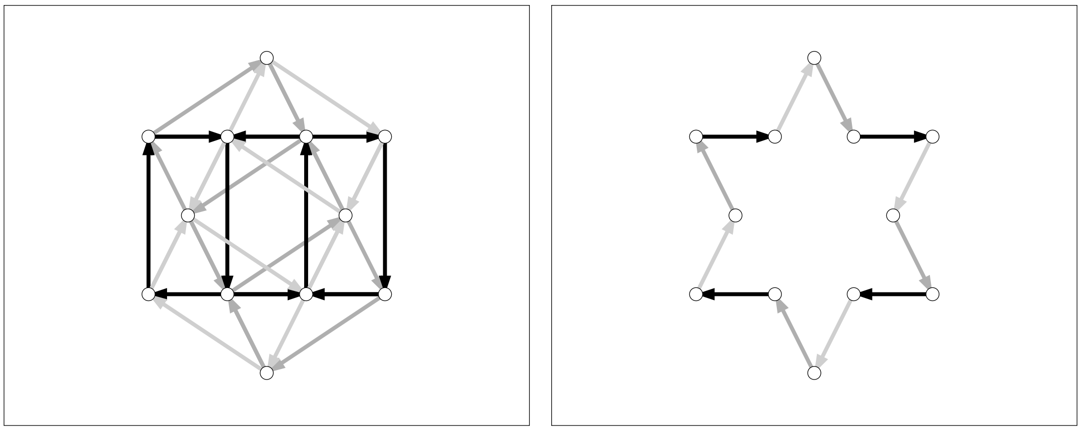

Figure 2.6: an initial graph G_3 and a solution subgraph H_3

We divide the problem of moment-based polygon reconstruction into three distinct phases:

1. Given complex moment data as input, the first step of the reconstruction provides approximations to the Cartesian coordinates of the vertices and the angles for two axes of potential in-edges and out-edges at each vertex.

2. Vertex positioning and potential edge information is mapped to an embedded edge-coloured directed graph G.

3. We look for a spanning subgraph $H \subseteq G$ that is a globally-happy 2-factor.

In our research, we examine the two latter steps with a focus on the last challenge: the problem of finding a globally-happy 2-factor.

Chapter 3

Restricted 2-Factors

The third phase in the problem of reconstructing a polygon from its complex moments involves searching for a spanning subgraph $H \subseteq G$ which is a globally-happy 2-factor. Thus, we are searching for a 2-factor with specific properties: a **restricted** 2-factor. In this chapter, we examine properties of various relevant restrictions to the general 2-factor problem. In Section 3.1, we examine the problem of finding a general 2-factor. In Section 3.2, we define three restrictions to the 2-factor problem. In Sections 3.3 and 3.4, we show **NP**-hardness for various restricted 2-factor problems, including, EDGE-BICOLOURED DIRECTED 2-FACTOR, BOUNDED-DEGREE DIRECTED 2-FACTOR, and NON-CROSSING 2-FACTOR. Finally, Section 3.5 summarizes our discussion of restricted 2-factors.

The unconstrained directed 2-factor problem is polynomial-time solvable [GJ79, LP86]. The problem of finding a globally-happy 2-factor in polygon reconstruction from moments involves the imposition of specific restrictions on the unconstrained 2-factor problem. We identify these additional properties by four specific restrictions to the input graph and output 2-factor:

1. We may constrain edge choice by requiring global happiness in the resulting 2-factor.

2. We may require that edges of the 2-factor be non-crossing.

3. We may require that each local blue axis and red axis of the vertices of the input graph be orthogonal.

4. We may limit in-degree and out-degree to two edges per colour at every vertex.

In this chapter, we examine these four constraints in isolation to determine their effect on the difficulty of the problem. We discuss the relationship between our colour-constrained 2-factor problem and the unconstrained 2-factor problem. We show **NP**-hardness for specific 2-factor restrictions which embody essential aspects of the globally-happy 2-factor problem.

3.1 2-Factors

The undirected 2-factor problem involves taking an undirected graph, $G = (V, E)$, and finding a spanning subgraph, $H = (V', E')$, such that every vertex $v \in V'$ has degree 2 (see figure 3.1). The directed 2-factor problem is similar to the undirected

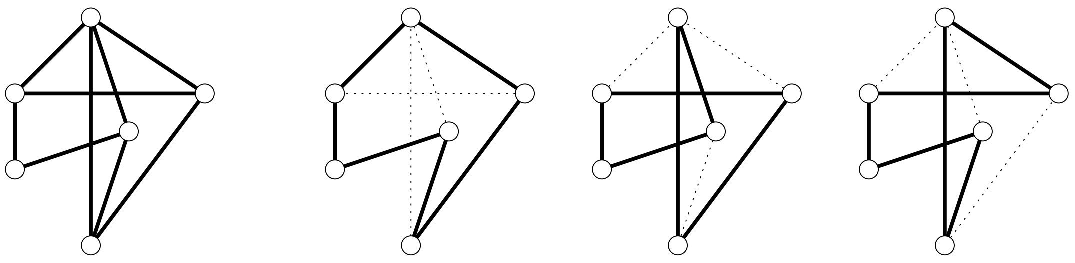
Figure 3.1: a graph G and three subgraphs which are 2-factors of G

problem but with the additional requirement that every vertex $v \in V'$ in the 2-factor must have in-degree 1 and out-degree 1.

The colour constraints on edges may be restated as a cycle-restricted 2-factor problem by replacing each vertex by a simple component. Figure 3.2 displays two com-

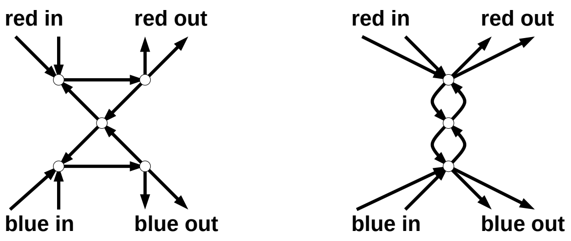

Figure 3.2: We replace vertices with one of the components in which 3-cycles and 2-cycles are disallowed, respectively.

ponents that embed colour constraints into the graph if 3-cycles are disallowed with use of the first component and if 2-cycles are disallowed with use of the second component. By replacing all vertices with either of these components, the problem may be restated as a k-cycle-restricted 2-factor problem, where k is either 2 or 3. Without these restrictions, a closed k-cycle within the component would allow two edges of similar colour to be included in a 2-factor.

Given any directed graph, $G = (V, E)$, an unconstrained 2-factor may be solved by mapping to a matching problem [LP86]. To do so, we make two copies of every vertex $v \in V$. One instance retains all in-edges and the other retains all out-edges (see figure 3.3). All in-edge vertices, v', are placed into a partition V' and

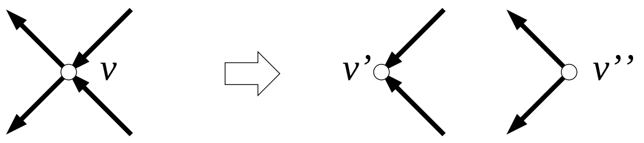

Figure 3.3: $v \Rightarrow v'$ and v''

all out-edge vertices, v'', are placed into a partition V''. We call this new graph $G' = (V' \cup V'', E')$. Whenever an edge uv appears in G, we include an edge from u'' to v' in E' (see figure 3.4). Therefore, any directed graph, G, may be mapped to

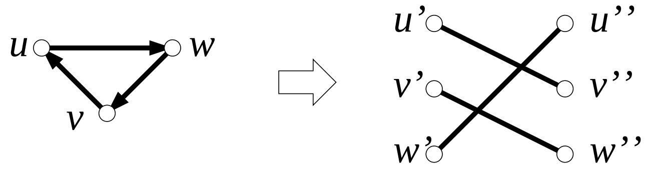

Figure 3.4: mapping a 2-factor problem to a matching problem

an undirected bipartite graph G' such that $uv \in E \Leftrightarrow u''v' \in E'$.

Thus, there exists a perfect matching in G' if and only if there exists a directed 2-factor in G. The bipartite matching problem is solvable in polynomial-time [LP86]. The 2-factor problem, therefore, is also solvable in polynomial-time.

Hell *et al.* show that disallowing k-cycles makes the undirected 2-factor problem polynomial-time solvable if $k \leq 4$ and **NP**-hard if $k > 4$ [HKKK88]. This polynomial-time solution to finding cycle-restricted undirected 2-factors provides motivating evidence for a possible polynomial-time solution to the globally-happy 2-factor problem.

It will follow later that disallowing k-cycles makes the directed 2-factor problem **NP**-hard for $k = 2$ and $k = 3$ (see Corollary 3.2). We go on to show that restricting the 2-factor problem with colour constraints or disallowing crossing edges renders the problem **NP**-hard. That is, we show that imposing properties 1, 2, or 4 render the problem **NP**-hard.

3.2 Restricting the 2-Factor Problem

The unconstrained directed 2-factor problem is polynomial-time solvable [GJ79, LP86]. Our polygon reconstruction problem differs from the basic 2-factor problem simply by the addition of one or more of the four additional constraints. The addition of these seemingly minor parameters significantly alters the complexity of the problem.

We know 2-FACTOR lies near the boundary between **P** and **NP**-complete since 2-FACTOR $\in$ **P** and HAM-CYCLE $\in$ **NP**-complete [GJ79]. The only difference between these two problems is that HAM-CYCLE requires the existence of a single spanning cycle, namely, a Hamiltonian cycle, whereas 2-FACTOR allows multiple disjoint cycles whose union spans the graph.

In addition to special colour constraints, one may reasonably require that polygonal regions be non-intersecting, that is, that their interior density be constant and non-negative. This corresponds to searching for a 2-factor in which no edges cross.

Thus, we define three restrictions of 2-FACTOR that model specific aspects of our globally-happy 2-factor problem; these restricted 2-factor problems impose properties 1 (Theorem 3.1) properties 1 and 4 (Corollary 3.3) and property 2 (Theorem 3.4) respectively. We examine the time complexities of each restricted problem.

3.3 Edge-Bicoloured Directed 2-Factor

3.3.1 Problem Instance and Question

When imposing the first property, that the resulting 2-factor respect global-happiness, our problem reduces to the following:

EDGE-BICOLOURED DIRECTED 2-FACTOR

INSTANCE: Directed graph $G = (V, E)$, where edges of E are coloured red or blue at each end.

QUESTION: Does G admit a directed 2-factor H with red-degree 1 and blue-degree 1 at each vertex?

3.3.2 EDGE-BICOLOURED DIRECTED 2-FACTOR is NP-hard

Using a component-based proof, we give a reduction from 3-DIMENSIONAL MATCHING [GJ79] to EDGE-BICOLOURED DIRECTED 2-FACTOR.

To facilitate colouring within this proof, we interchange the colours of out-edges at every vertex. In doing so, vertex happiness may be achieved by finding a directed

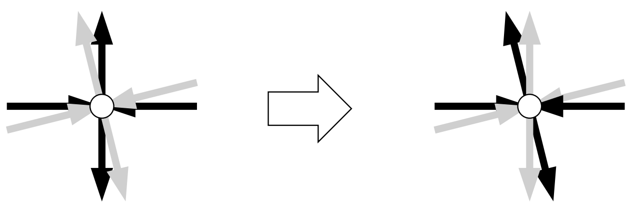

Figure 3.5: reversing out-edge colours at a vertex

2-factor such that, at every vertex, both edges have similar colour (see figure 3.5). Note that this does not alter the problem but simply alters the interpretation of the colour rules.

An instance of 3-DIMENSIONAL MATCHING consists of three sets of equal size, X, Y, and Z, along with triples of the form $T = \{(x_i, y_j, z_k)\} \subseteq X \times Y \times Z$, such that every x_i, y_j, and z_k appears in at least one triple. A solution to 3-DIMENSIONAL MATCHING consists of a subset $T' \subseteq T$ such that every x_i, y_j,

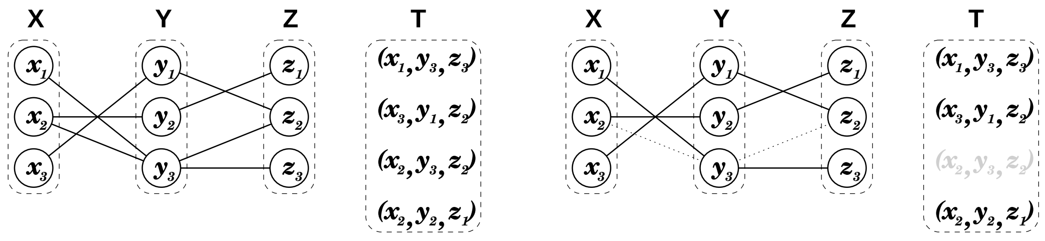

Figure 3.6: example of 3-DIMENSIONAL MATCHING instance and solution

and z_k is visited exactly once.

We represent a single point, x_i, y_j, or z_k by a pair of vertices connected by a single blue edge (see figure 3.7).

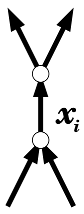

Figure 3.7: single point component

We create a component that connects three points, x_i, y_i, and z_i into a triple (see figure 3.8). Each triple has three pairs of blue edges that link the points to the component.

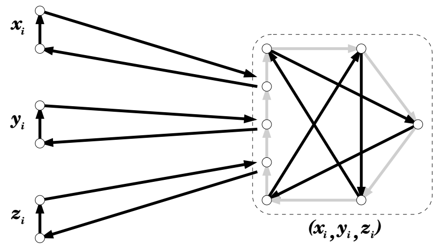

Figure 3.8: We represent a triple, (x_i, y_i, z_i), by this component.

In an edge-bicoloured directed 2-factor, every vertex must be met by exactly one in-edge and one out-edge of similar colour. Thus, either all three or zero outside blue paths may visit the construction (see figure 3.9). This forces all three or none of the points to be included in a edge-bicoloured directed 2-factor on the triple component.

A point x_i may be a member of several triples. x_i is linked up to every triple

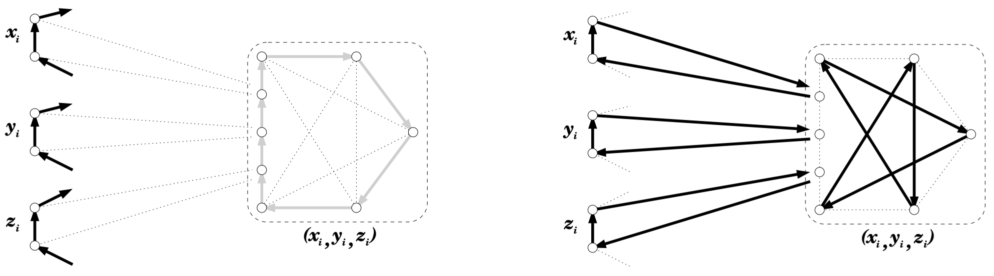

Figure 3.9: In an edge-bicoloured directed 2-factor, the triple component may be in one of two states.

component to which it belongs by a pair of blue edges. Thus, several pairs of blue edges may meet x_i. We call this entire graph of linked triples and points, G' (see figure 3.10). In any edge-bicoloured directed 2-factor, only a single pair of blue edges may visit any x_i. Thus, every x_i may only be included in exactly one triple.

Since individual points may only be visited by one pair of edges and since points must be included in given triples, a 3-dimensional matching in T corresponds to an edge-bicoloured directed 2-factor in G'. Given any X, Y, Z, and T, a 3-dimensional matching can be found if and only if there exists an edge-bicoloured directed 2-factor in G'. We have shown a polynomial-time reduction function, f_1, such that:

$$x \in \text{3DM} \Leftrightarrow f_1(x) \in \text{EDGE}-\text{BICOLOURED DIRECTED } 2-\text{FACTOR}$$

Therefore, we have proven:

Theorem 3.1 (Kirkpatrick-Durocher, 1999)

$$\text{EDGE}-\text{BICOLOURED DIRECTED } 2-\text{FACTOR} \in \mathbf{NP}-\text{hard}$$

Recall that we originally sought to show polynomial-time complexity for EDGE-BICOLOURED DIRECTED 2-FACTOR. In Section 3.1, we described a polynomial-time reduction, f_2, from EDGE-BICOLOURED DIRECTED 2-FACTOR to k-CYCLE-FREE DIRECTED 2-FACTOR for $k = 2$ and $k = 3$ such that:

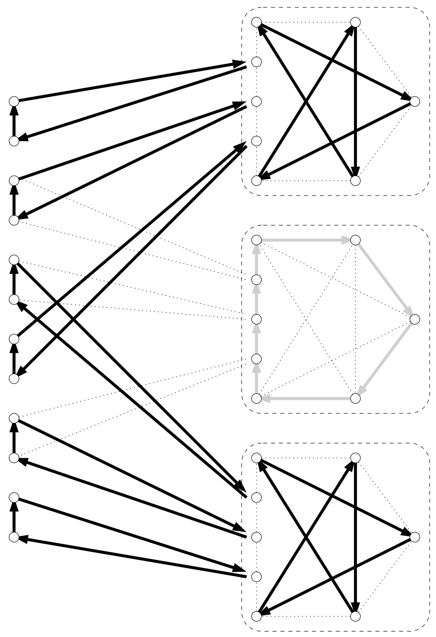

Figure 3.10: example of 3-dimensional matching within G' transformation

$$x \in \text{EDGE} - \text{BICOLOURED DIRECTED } 2-\text{FACTOR}$$

$$\Leftrightarrow f_2(x) \in k\text{–CYCLE}-\text{FREE DIRECTED } 2-\text{FACTOR}, k \in \{2, 3\}$$

If k-CYCLE-FREE DIRECTED 2-FACTOR were solvable in polynomial time for $k = 2$ or $k = 3$, then EDGE-BICOLOURED DIRECTED 2-FACTOR could also be solved in polynomial time. By contradiction, therefore, k-CYCLE-FREE DIRECTED 2-FACTOR cannot be solvable in polynomial time. Thus, we have shown:
Corollary 3.2 (Kirkpatrick-Durocher, 1999)

$$k\text{–CYCLE}-\text{FREE DIRECTED } 2-\text{FACTOR} \in \mathbf{NP}-\text{hard}, k \in \{2, 3\}$$

3.3.3 Bounded-Degree Edge-Bicoloured Directed 2-Factor

In our problem, the last property limits a vertex to at most two in-edges and two out-edges of any colour. Through a simple modification of the vertex component, we show that BOUNDED-DEGREE EDGE-BICOLOURED DIRECTED 2-FACTOR remains **NP**-hard.

We formally define BOUNDED-DEGREE EDGE-BICOLOURED DIRECTED 2-FACTOR as follows:

BOUNDED-DEGREE EDGE-BICOLOURED DIRECTED 2-FACTOR
INSTANCE: Directed graph $G = (V, E)$, where edges of E are coloured red or blue at each end and any $v \in V$ has blue in-degree, blue out-degree, red in-degree, and red out-degree ≤ 2.
QUESTION: Does G admit a directed 2-factor H with red-degree 1 and blue-degree 1 at each vertex?

Garey and Johnson write, 3-DIMENSIONAL MATCHING "also remains **NP**-complete if no element occurs in more than three triples, but is solvable in polynomial time if no element occurs in more than two triples." [GJ79, page 221]

We exploit this fact and replace the two-vertex component with six vertices (see figure 3.11). Exactly three pairs of exterior blue edges meet the component.

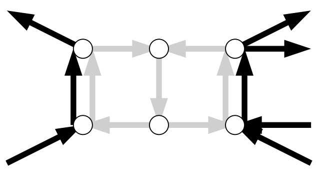

Figure 3.11: bounded-degree component for a vertex x_i, y_j, or z_k

Since every vertex may appear in at most three triples, this new vertex component requires connecting edges to, at most, three triple components.

Again, in an edge-bicoloured directed 2-factor, only a single pair of exterior blue edges may be followed (see figure 3.12). The difference, however, is that every vertex now has blue in-degree, blue out-degree, red in-degree, and red out-degree bounded by two.

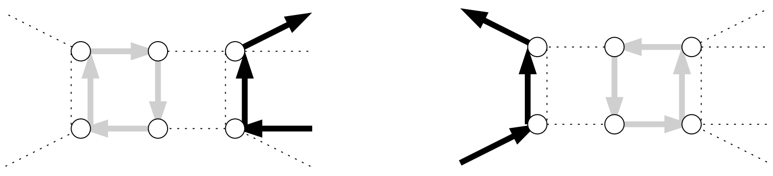

Figure 3.12: edge-bicoloured directed 2-factor on bounded-degree component

Following a proof identical to that from Theorem 3.1 except with use of this new bounded-degree component for points x_i, y_j, or z_k, we show that even with a single colour degree bound of two, the problem remains **NP**-hard. Thus, we have demonstrated:

Corollary 3.3 (Kirkpatrick-Durocher, 1999)

$$\text{BOUNDED}-\text{DEGREE EDGE}-\text{BICOLOURED DIRECTED 2}-\text{FACTOR} \in \mathbf{NP}\text{–hard}$$

Finally, in an attempt to bound degree as tightly as possible, we use a component that bounds red in-degree and out-degree to one (see figure 3.13). In doing so, we

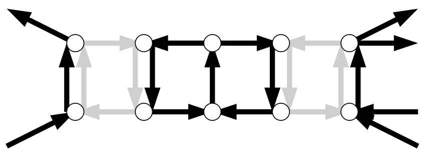

Figure 3.13: modified bounded-degree component

show that even if we limit blue in-degree and out-degree to two and red in-degree and out-degree to one, EDGE-BICOLOURED DIRECTED 2-FACTOR remains **NP**-hard.

3.3.4 Degree Four versus Degree Five

At this point, one may wonder about the boundary between **P** and **NP**-hard. Does the problem become easier if vertices have lower degree? What happens if we bound the degree any more tightly? In our final reduction, any vertex, v, has red in and out-degree ≤ 1, and, at most, three blue edges, only two of which have similar orientation. Thus, v has at most five edges.

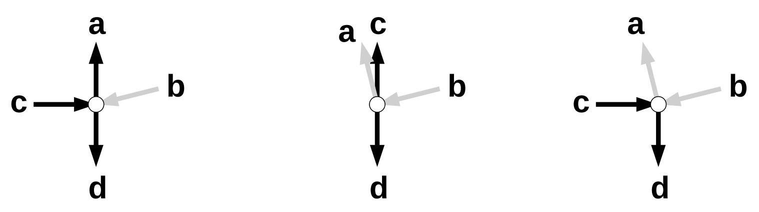

Figure 3.14: three cases when removing an additional edge

If we remove any additional edge, one of three cases may arise. We may remove a red edge in which case only a single red edge remains (figure 3.14a), we may remove a blue edge such that two blue edges of similar orientation remain (figure 3.14b) or we may remove a blue edge such that two blue edges of opposite orientation remain (figure 3.14c).

In each of these cases, the problem of finding an edge-bicoloured directed 2-factor can be described logically as a simple conjunction of disjunctions. The colour constraints on the above examples may be rewritten, respectively, as follows:

$$b \wedge (a \oplus d)$$

$$b \wedge (c \oplus d)$$

$$(a \oplus d) \wedge (b \oplus c) \wedge (a \oplus b) \wedge (c \oplus d)$$

Note that exclusive-OR is itself logically equivalent to a conjunction of disjunctions:

$$(x \oplus y) \equiv (x \vee y) \wedge (\bar{x} \vee \bar{y})$$

The vertices of any graph of degree ≤ 4, therefore, could be described by a conjunction of such disjunctions corresponding to local colour constraints. Every disjunctive clause contains at most two literals meaning that the problem reduces[1] to 2-SAT. 2-SAT is solvable in polynomial time [GJ79]. Reducing the number of edges to 4, therefore, brings the complexity down within range of polynomial-time solvability. When edges are orthogonal, Rendl and Woegingner show that reconstruction of sets of orthogonal line segments in the plane from the set of their vertices can also be done in polynomial time [RW93].

Thus, we have shown that with at least five edges at every vertex, EDGE-BICOLOU -RED DIRECTED 2-FACTOR is **NP**-hard. However, with at most four edges at any vertex, EDGE-BICOLOURED DIRECTED 2-FACTOR $\in$ **P**. Even under extensive pruning of edges, therefore, EDGE-BICOLOURED DIRECTED 2-FACTOR remains **NP**-hard.

3.4 Non-Crossing 2-Factor

3.4.1 Problem Instance and Question

If we ignore colour constraints and simply require that property 2 be respected, that polygonal regions be non-intersecting, our 2-factor problem reduces to a restricted 2-factor problem in which we disallow crossing edges. For simplicity, we first examine the problem of finding a non-crossing 2-factor on an undirected graph. We formalize the problem as follows:

NON-CROSSING 2-FACTOR

INSTANCE: Undirected embedded graph $G = (V, E)$.

[1]A reduction from problem p to problem q shows that q is "at least as hard" as p. Thus, when showing **NP**-hardness of q, we reduce a known **NP**-hard problem to q. When showing $q \in$ **P**, the argument is reversed and we reduce q to a known **P** problem.

QUESTION: Does G admit a 2-factor H, none of whose edges intersect except at their endpoints?

3.4.2 NON-CROSSING 2-FACTOR is NP-hard

Using a component-based proof, we give a reduction from 3-DIMENSIONAL MATCHING to NON-CROSSING 2-FACTOR. [2] We will present a reduction which is initially undirected but for which every component may be directed without any effect on functionality.

The first component essential to the reduction is one that may be inserted at the intersection of two crossing edges, A and B, to force a logical exclusive-OR within a non-crossing 2-factor (see figure 3.15). That is, either edge A or edge B may be

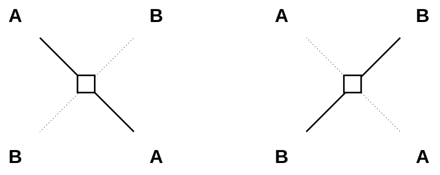

Figure 3.15: following either edge A or edge B

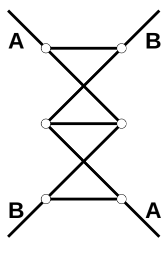

Figure 3.16: XOR component

[2]Jansen and Woeginger show that given an undirected graph G whose edges are axis-parallel, the problem of deciding whether G has a crossingfree two-factor is **NP**-complete [JW93]. Here we show that both the directed and undirected cases are **NP**-hard.

followed, but not both, nor neither, nor some fragment of either. The component construction is displayed in figure 3.16.

Note that any non-crossing 2-factor within this component must find itself in one of two states. Either either the path $A - A$ or the path $B - B$ is followed. Any other combination forces edges to cross (see figure 3.17).

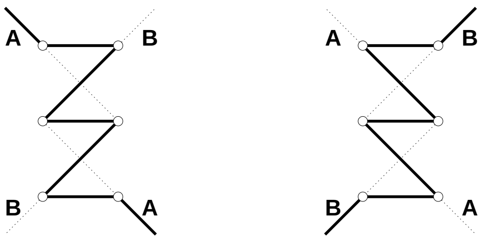

Figure 3.17: two states of XOR component in a non-crossing 2-factor

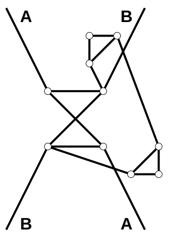

Figure 3.18: crossover component

We require a second component to eliminate unwanted edge-crossings. Inserting this component does not affect membership of edges within a 2-factor but eliminates actual crossing. In a non-crossing 2-factor, this construction forces one of $A - A$, $B - B$, neither or both edges to be followed (see figures 3.18 and 3.19). No other combination of edges is possible. For example, it is impossible to visit only the bottom B edge and the top A edge without forcing edges to cross.

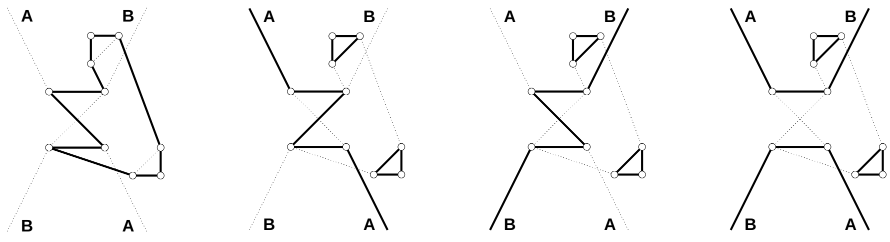

Figure 3.19: four states of crossover component in a non-crossing 2-factor

We may now begin the reduction from 3-DIMENSIONAL MATCHING. Let $S = X \cup Y \cup Z$. The sets S and T form a bipartite graph G (see figure 3.20a).

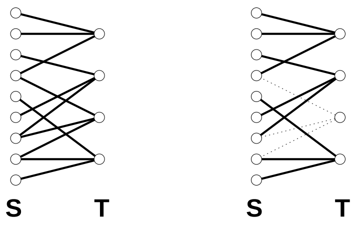

Figure 3.20: 3-dimensional matching instance and solution in bipartite graph

Finding a 3-dimensional matching amounts to finding a subset of edges of G which meet every vertex in the S partition exactly once and every vertex in the T partition exactly zero or three times (see figure 3.20b).

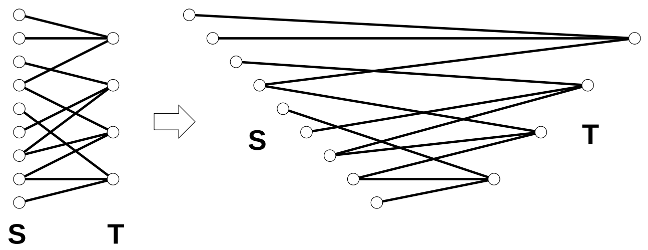

Figure 3.21: stretching G

We first stretch G onto diagonals (see figure 3.21).

We replace all crossovers using our crossover component (see figure 3.22). This implies that any 2-factor within the graph will be non-crossing.

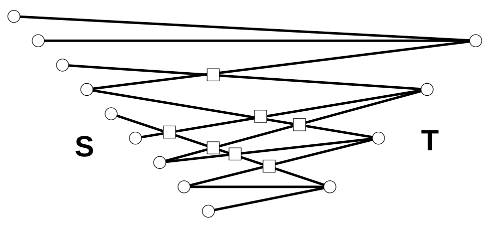

Figure 3.22: eliminate crossovers using crossover component

We add a mirror image of the entire graph and connect top vertices to their corresponding neighbours on the bottom. Call this graph G' (see figure 3.23).

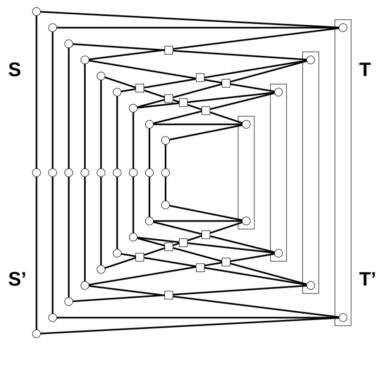

Figure 3.23: add mirror image and connect vertices

We require a special component to connect triples from T to T' (see figure 3.24).

In a non-crossing 2-factor, this component forces either all six or zero outgoing edges to be included (see figure 3.25). As required, therefore, either all three or none of the members of a triple are included in a non-crossing 2-factor.

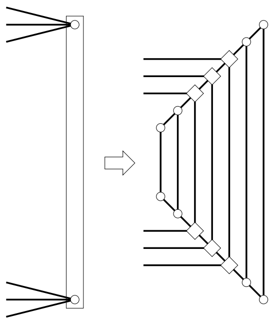

Figure 3.24: connecting triples from T to T'

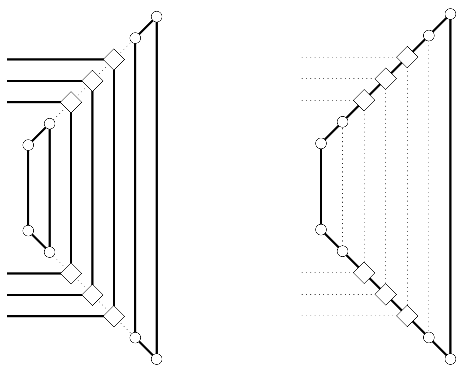

Figure 3.25: two states of component in non-crossing 2-factor

Since the edges from S to S' are forced and the edges in T and T' have the property that either all three or none may be followed, finding a 3-dimensional matching in G corresponds to finding a non-crossing 2-factor in G'. Given any S and T, a 3-dimensional matching can be found if and only if there exists a non-crossing 2-factor in G'. We have shown a polynomial-time reduction function, f_3, such that:

$$x \in \mathrm{3DM} \Leftrightarrow f_3(x) \in \mathrm{NON-CROSSING\ 2-FACTOR}$$

Therefore, we have proven,

$$\mathrm{NON-CROSSING\ 2-FACTOR} \in \mathbf{NP}-\mathrm{hard}$$

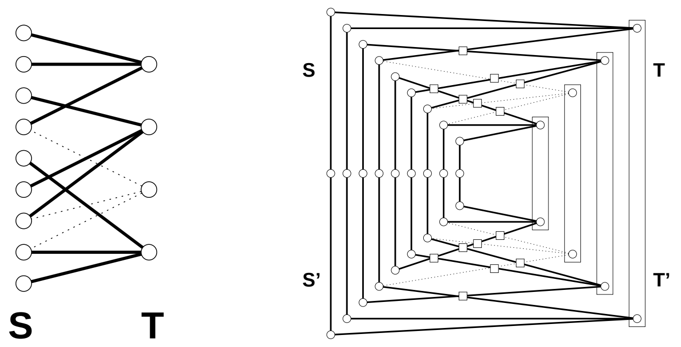

Figure 3.26: full reduction from bipartite S and T to G

In our input domain, graphs are directed. Each component within this construction may be directed without any alteration to its function (see figure 3.27).

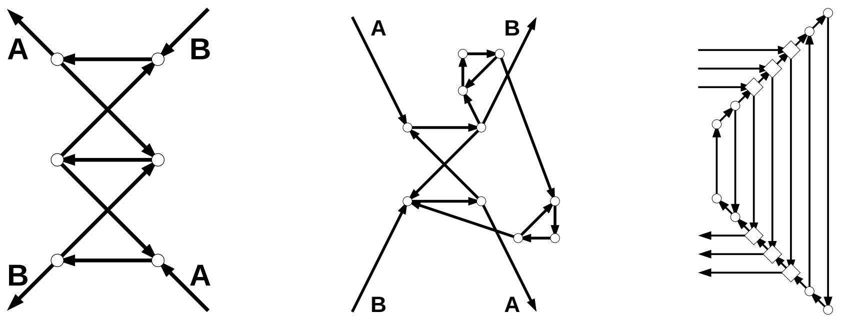

Figure 3.27: reduction components with directions added

Following a proof identical to that for NON-CROSSING 2-FACTOR, except with use of these new directed components, we show that DIRECTED NON-CROSSING 2-FACTOR remains **NP**-hard. We therefore derive the following theorem:

Theorem 3.4 (Kirkpatrick-Durocher, 1999)

$$\text{DIRECTED NON}-\text{CROSSING } 2-\text{FACTOR} \in \mathbf{NP}-\text{hard}$$

3.5 Summary

Using component-based reductions from 3-DIMENSIONAL MATCHING, we have shown that simple restrictions, such as edge-bicoloured constraints or non-crossing constraints, render the 2-factor problem problem **NP**-hard. These restrictions closely model various properties of our polygon reconstruction from moments problem. As demonstrated, this particular problem seems to lie very close to the boundary between **P** and **NP**-hard. The actual problem, encompassing colour constrains, non-crossing constraints, degree-bounds, and geometric orthogonal constraints remains to be shown to be polynomial-time solvable or **NP**-hard.

Chapter 4

Uniqueness of Moment Sets

The vertices in a moment-derived edge-bicoloured graph have special properties which allow for colour constraints to be propagated between neighbouring vertices. We discuss the basis of colour constraint propagation in Section 4.1. Certain graphs embed multiple distinct solutions. Constraints cannot be propagated within such graphs. We discuss the properties of non-unique solutions in Section 4.2.

4.1 Commitment Propagation

The global happiness requirement in edge selection forces any vertex v within a solution to be in one of two vertex states: red-in, blue-out (RIBO) or blue-in, red-out (BIRO) (see figure 4.1). Thus, v is **constrained** to having a single pair of

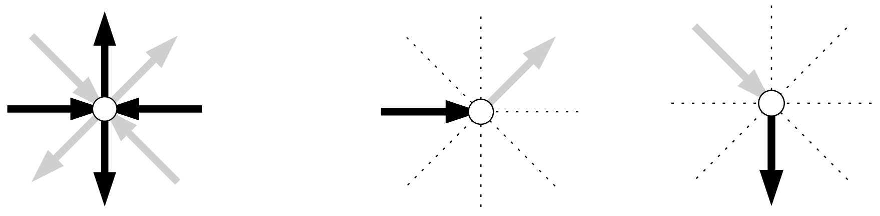

Figure 4.1: A vertex may be in one of two colour states.

edges: red in and blue out, or, blue in and red out. We refer to these restrictions as **colour constraints**. Since these constraints are upon the edges of a vertex, vertices at both endpoints may be affected by the restrictions; **constraint propagation** occurs whenever the constraints at a vertex v affect the constraints at a neighbouring vertex u. Finding a 2-factor solution involves committing to edges within the graph. Whenever a vertex v commits to an edge e, several implications arise from that commitment. All other edges of similar colour or similar direction at v must be dropped from the possible solutions space. For example, if e is a committed red in-edge at v, then all other red edges and all other in-edges incident on v are dropped; the only remaining type of edges allowed are blue out-edges. v now has vertex state RIBO.

A commitment applies both to the head and tail of an edge. Thus, whenever we commit to an edge uv, the vertex states of both u and v become fixed. For example, say vertex u is in vertex state RIBO and a blue edge connects u to vertex v. If we commit to uv, then the vertex state of v becomes BIRO (see figure 4.2). Similarly, dropping an edge applies to both the head and tail of an edge. Thus,

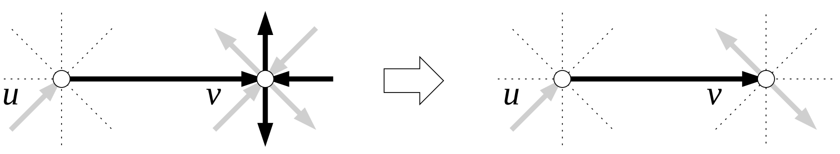

Figure 4.2: propagation of vertex states

whenever we drop an edge uv, the vertex states of both u and v may be affected.

If graph $H_2 \subseteq H_1$ is a spanning subgraph of H_1 and all edges $E \subseteq H_1 - H_2$ may be removed from H_1 by constraint propagation, then we say H_2 is a **reduced** graph of H_1. If no further edges may be removed from H_2 by constraint propagation, then H_2 is **maximally reduced**.

As we will see in Section 5.4, such commitment propagation forms the basis of our algorithm for finding solutions within a graph. There exist graphs, however, for which simple commitment propagation cannot be used to simplify a graph towards a solution subgraph. Some of these graphs embody more than a single solution subgraph. Commitment propagation alone will not provide a solution on such a graph.

4.2 Non-Unique Solutions

Interestingly, some graphs embed more than one globally-happy 2-factor. Under ordinary conditions, there would be no reason to expect such a graph to arise naturally; non-unique solutions graphs occur infrequently and require specific contrived constructions. Their properties, however, are quite fascinating to examine and provide insight into the mechanisms and obstacles of constraint propagation. In this section, we explore the uniqueness or non-uniqueness of solutions within edge-bicoloured graphs.

A solution to an edge-coloured graph $G = (V, E)$ is a globally-happy 2-factor spanning subgraph, $H \subseteq G$. For two distinct solutions to exist, there must exist two such subgraphs, H_1 and H_2, such that $H_1 \neq H_2$. Since H_1 and H_2 have a common vertex set, V, we must have that $E_1 \neq E_2$.

We distinguish between three types of graphs with non-unique solutions within a hierarchical classification. Graphs with **distinct state solutions** contain exactly two solutions, H_1 and H_2, such that for any vertex $v \in V$, v is in vertex state RIBO in H_1 and BIRO in H_2 or vice-versa. Graphs with **distinct edge solutions** may contain two or more solutions, $H_1, \ldots, H_k$, such that for any edge $e \in E$, e appears in at most one solution $H_1, \ldots, H_k$. That is, the intersection of the edge sets of any two solution subgraphs is empty. Graphs with **common edge solutions** have one or more edges e which remain present in two or more solutions.

If a graph embodies three or more solutions, then in at least two of the solutions, there must be some $v \in V$ whose vertex-state is unchanged. The solutions of such a graph cannot be distinct state solutions. We further classify solutions by examining connectedness (simple or multiple) and edge crossing (crossing or non-crossing).

With respect to common edge solutions, Milanfar *et al.* state the following proposition [MVKW95]:

Proposition 4.1 (Milanfar et al., 1995) *Consider n distinct points, $z_1, z_2, \ldots, z_n$ in the Cartesian plane. Let P and P' be simply connected, nondegenerate, n-gons generated by connecting these vertices in two distinct ways. If P and P' have at least one side in common, then for some $1 \leq j \leq n$, $a_j(P) \neq a_j(P')$, where $a_j(P)$ and $a_j(P')$ are, respectively, defined by:*

$$\int\int_P h''(z)\ dx\ dy = \sum_{j=1}^{n} a_j(P) h(z_j)$$

$$\int\int_{P'} h''(z)\ dx\ dy = \sum_{j=1}^{n} a_j(P') h(z_j)$$

with h denoting any analytic function in the closure of $P \cup P'$.

Proposition 4.1 claims that if P and P' are two distinct polygons which have at least one edge in common, then P and P' cannot be solutions to the same set of input constraints, otherwise P and P' must include degenerate vertices. Contrary to this proposition, in this section, we will demonstrate the existence of such common edge solutions, where all vertices are nondegenerate.

We first look at the octagon graph (see figure 4.3). The octagon is the simplest graph (least number of vertices) we have found which contains two simply-connected ambiguous solutions. Figure 4.3a displays the entire supergraph G_1. Figures 4.3b and 4.3c display the two possible globally-happy 2-factors of G_1. These 2-factors are

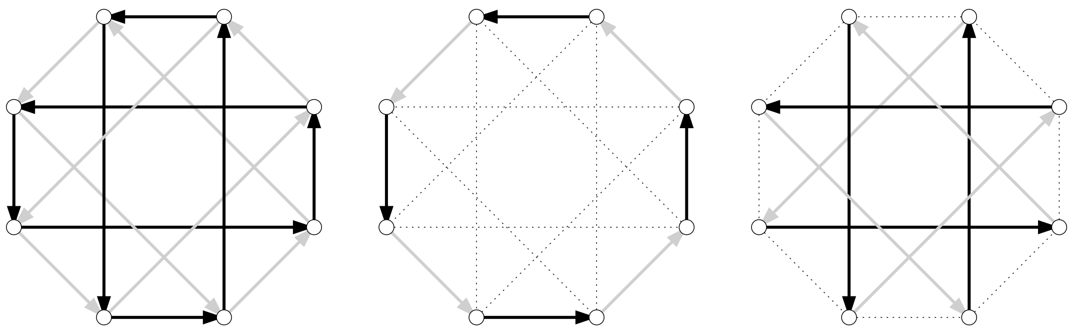

Figure 4.3: ambiguous solutions within the octagon graph

simply-connected but the second obviously involves edge-crossing. Note that at every vertex v, v is either in vertex state RIBO or BIRO in the different solutions (see figure 4.4). Furthermore, binding any single vertex to a fixed vertex state immedi-

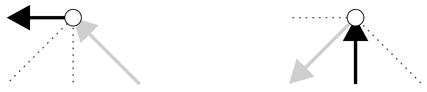

Figure 4.4: two vertex states within different solutions

ately determines the configuration of the remaining edges in the solution. Thus, G_1 is a graph with distinct-state solutions for which all solutions are simply-connected but not all are non-crossing.

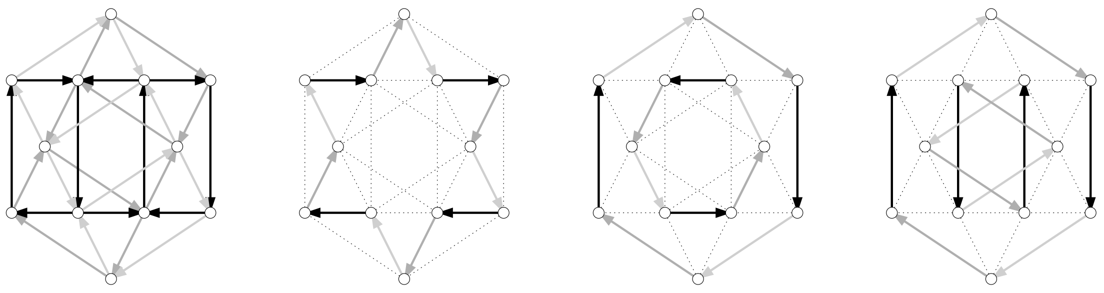

Figure 4.5: ambiguous solutions within the six-point star

Another simple graph which contains non-unique solutions is the hexagonal six-point star (see figure 4.5). This graph embodies three solutions. In this case, the star solution, figure 4.5b, is simply-connected, but the other two solutions, figures 4.5c and 4.5d, involve separate components. Again, edge-crossing occurs within some solutions.

Strakhov and Brodsky present a very cleverly constructed graph [SB86] which contains two solutions respecting simple-connectedness and non-crossing edge properties (see figure 4.6). The Strakhov-Brodsky graph is created by merging two

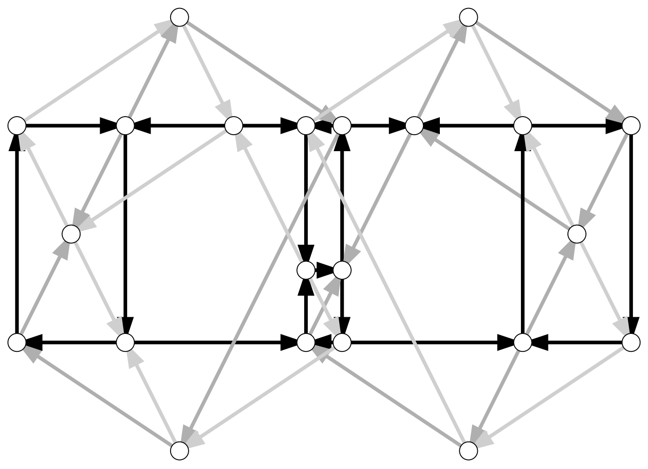

Figure 4.6: the Strakhov-Brodsky graph [SB86]

instances of the six-point star, removing a vertex from each, and displacing two others. To our knowledge, their construction is the simplest graph to embody multiple simply-connected, non-crossing solutions. The interesting behaviour in many ambiguous graphs derives from having outer vertices which allow convexity within two vertex states (see figure 4.4). The two solutions to the Strakhov-Brodsky graph are displayed in figure 4.7. They are symmetrical reflections of each other along a central vertical axis.

Within all examples presented up to this point, no two solutions share a

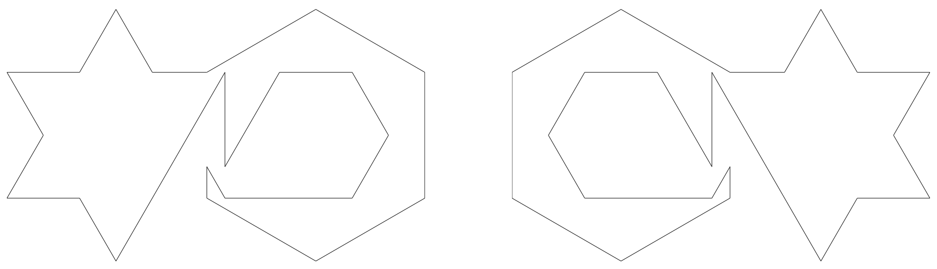
Figure 4.7: ambiguous solutions within the Strakhov-Brodsky graph

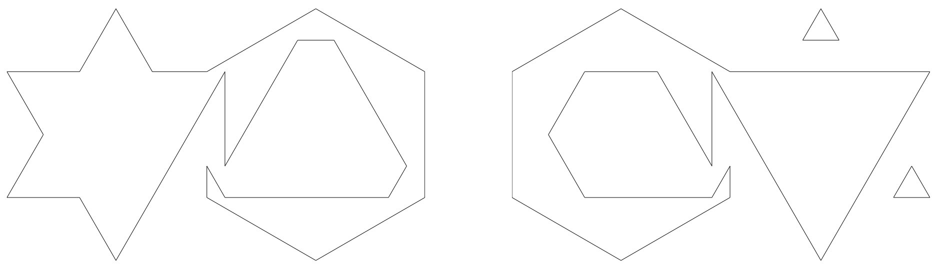
Figure 4.8: ambiguous solutions within the modified Strakhov-Brodsky graph [SB86]

common edge unless the common edge belongs to a static disconnected component (as is the case with the outer hexagon in the six-point star in figures 4.5c and 4.5d). Strakhov and Brodsky also present a graph which contains two solutions sharing two common edges [SB86]. This graph may be constructed by simple modification to the regular Strakhov-Brodsky graph (see figure 4.8).

Note the two triangles in figure 4.8b. Their inside edge is also present in the first solution in figure 4.8a. Given any graph which contains ambiguous solutions H_1 and H_2, one may easily add constant edges by selecting any edge e, $e \in H_1$, $e \notin H_2$, and inserting three additional vertices to form a triangle along the edge (see figure 4.9a). We call this new graph G' and its solutions H_1' and H_2'.

Since $e \in H_1$, in solution H_1', we do not include edge c (see figure 4.9b). Conversely, since $e \notin H_2$, in solution H_2', we include edge c but not edges a or e (see

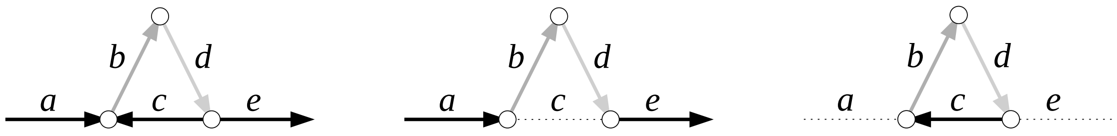

Figure 4.9: general pattern within common-edge solutions

figure 4.9c). The resulting graph, G', is such that, in both solutions, edges b and d remain constant.

The modified Strakhov-Brodsky graph has two non-crossing solutions, but one of the solutions is composed of disconnected components. Thus we examine the problem of finding a graph with common-edge ambiguous solutions such that all solutions are non-crossing and simply-connected. By extension of our observations about inserting constant edges, we can augment the regular Strakhov-Brodsky graph such that all solutions share common edges along the exterior of the graph and all solutions remain simply-connected and non-crossing.

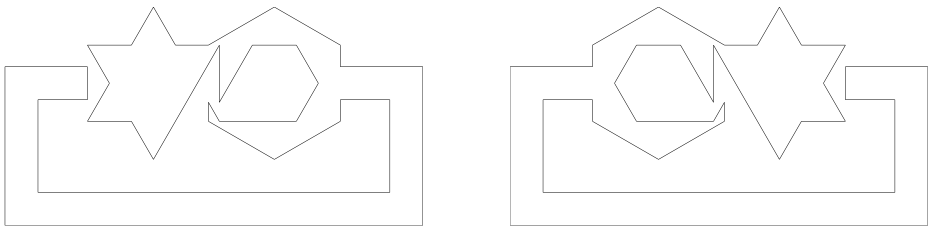

Figure 4.10: extended Strakhov-Brodsky graph

Let SB_1 and SB_2 be the two solutions to the regular Strakhov-Brodsky graph. We choose two edges e_1 and e_2 along the perimeter of the graph such that $e_1 \in SB_1$, $e_1 \notin SB_2$, $e_2 \notin SB_1$, and $e_2 \in SB_2$. We add a series of static edges to connect e_1 and e_2 (see figure 4.10).

These edges are such that only one of the two ends will connect to the graph within a solution. Thus, both solutions solutions, SB_1' and SB_2', remain simply-

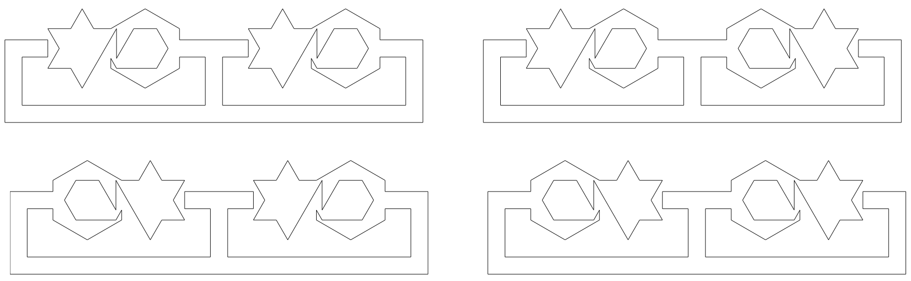

Figure 4.11: four instances of the extended Strakhov-Brodsky graph

connected and non-crossing. The difference between the regular and the extended Strakhov-Brodsky graphs is that SB_1' and SB_2' share common edges. Significantly, we can join two such extended Strakhov-Brodsky graphs together by linking their common edges to create a graph which has four simply-connected, non-crossing solutions (see figure 4.11).

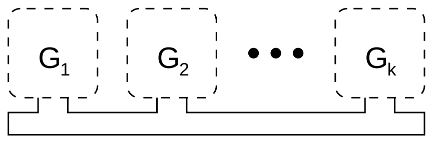

Figure 4.12: multiple connected instances of ambiguous graphs

One quickly observes that we may join k such extended Strakhov-Brodsky graphs together to form a graph which embeds 2^k simply-connected, non-crossing solutions (see figure 4.12). Thus, the number of solutions may be exponential with respect to the size of the graph.

Therefore, we have shown:

Theorem 4.2 (Durocher-Kirkpatrick, 1999) *For any $n \in \mathbf{Z}^+$, there exist 2^n distinct simple polygons in the Cartesian plane, each composed of $30n + 4$ vertices, such that all 2^n polygons have identical moment sets.*

Thus, for any $n \in \mathbf{Z}^+$, there exists an edge-bicoloured regular graph $G = (V, E)$ such that $|V| \leq kn$ and G contains 2^n distinct simply-connected non-crossing globally-happy 2-factors, where $k = 34$.

Our various extensions to the Strakhov-Brodsky graph demonstrate the existence of multiple common edge solutions to a single set of input vertices. All vertices within these solutions are nondegenerate, which contradicts proposition 4.1.

4.3 Summary

We have examined vertex properties which allow for the propagation of colour constraints. We use this propagation as a means of reducing a graph G towards a solution subgraph G'. Some graphs embed several non-unique solutions; the number of such solutions may be exponential with respect to the size of the graph. Within a multiple solution graph, constraint propagation alone cannot be used to reduce a graph to a solution. In Chapter 5, we discuss various methods which allow for the reduction of both unique-solution and non-unique-solution graphs.

Chapter 5

Reconstruction Algorithm

We have examined the various properties of input graphs and their embedded solutions. We now discuss algorithms that solve the problem of reconstructing the original polygon from which a set of moments were measured.

5.1 Reconstruction Process Overview

Recall the three-phase subdivision of moment-based polygon reconstruction:

1. Given complex moment data as input, approximate the Cartesian coordinates of the vertices and the angles for two axes of potential in-edges and out-edges at each vertex.

2. Map vertex positioning and potential edge information to an embedded edge-coloured directed graph G.

3. Find a spanning subgraph $H \subseteq G$ that is a globally-happy 2-factor.

Here we describe algorithms that were developed[1] to implement phases two and three of the reconstruction process. The algorithm modules form a pipeline and

[1]Each of the various algorithm modules presented here were implemented in MATLAB. All results and examples discussed were produced by running the MATLAB code. Some figures were redrawn from MATLAB plots for clearer presentation. The `recons` module is based on code written by J. Varah to implement the initial phase of reconstruction de-

may be broken down as follows:

1. `recons` takes k moment measurements and produces a set of n vertex positions and two sets of n angles each, for local red and blue edge axes at each respective vertex. Usually, $k = 2n$.

2. `buildGraph` takes n vertex positions[2] along with blue and red axis angles for each and builds a dense edge-bicoloured graph G_1 which includes edges for any matching combination of in-edge and out-edge angles (Section 5.2).

3. `edgeElim` takes G_1 as input and produces a moment-derived edge-bicoloured spanning subgraph G_2 such that every vertex has at most two edges of similar colour and orientation and these are at an angle π to each other (Section 5.3).

4. `commit` works recursively on G_2 using colour constraint propagation to produce a solution subgraph G_3 for which no further constraint propagation is possible (Section 5.4).

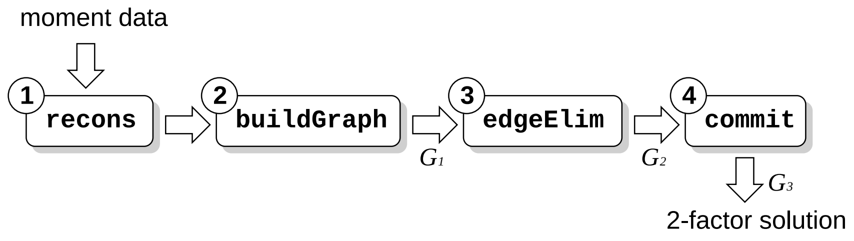

Figure 5.1: pipelined reconstruction modules

These four phases form the basic assembly line to transform moment data into reconstructed polygons (figure 5.1). As we will discuss, this solution process

scribed in his manuscript [GMV99]. All other modules were written by S. Durocher and D. Kirkpatrick.

[2]For examples that include significant numerical error and cause inaccurate position and angle input data to `buildGraph`, we fix the output of `recons` to provide perfect information (see Section 6.1).

may require extra manipulation and flow-control, and thus, we will describe three additional modules:

5. `treeSearch` works recursively down a search tree towards solutions whenever constraint propagation alone within module `commit` fails to reduce G_3 to a final solution (Section 5.5).
6. `fragmentSearch` is used whenever the input domain includes imperfect data. Partial fragmented solutions are researched as opposed to complete 2-factors (Section 5.6).
7. `momentDiff` allows for extremely noisy data to be reconstructed piecewise through re-iteration of the `recons` module on differences of moments (Section 5.7).

5.2 BuildGraph: Building the Initial Graph

After having passed through the initial phase of reconstruction, we are presented with a set of vertices, V. Each vertex $v \in V$ has an embedded position in the two-dimensional Cartesian plane and two associated angles for its local red and blue axes.

We wish to form a graph G_1 which includes an edge for any two vertices u and v such that u and v are compatible. We use the following definition for vertex compatibility:

Definition 5.1 *Given two vertex positions, u and v, an in-edge angle β' at u, an out-edge angle γ' at v, and a deviation tolerance angle α, let ϕ be the angle of the straight line between u and v, let $\beta = |\phi - \beta'|$, and let $\gamma = |\phi - \gamma'|$. v is* **compatible** *with u under the relation $C(v, u)$ if and only if $\beta \leq \alpha$ and $\gamma \leq \alpha$.*

Basically, two vertices u and v are compatible[3] whenever they have corre-

[3]The compatibility relation is asymmetric: $C(u, v) \nLeftrightarrow C(v, u)$, for $u, v \in V$.

sponding in-edge and out-edge angles which lie within a window of tolerance. Figure 5.2a displays two such vertices with the tolerance angle α and the line segment uv superimposed. In figure 5.2b, the edges at u and v are added and we see the two differences of angles, β and γ. Since both β and γ lie within the window α, we create an edge vu (figure 5.2c).

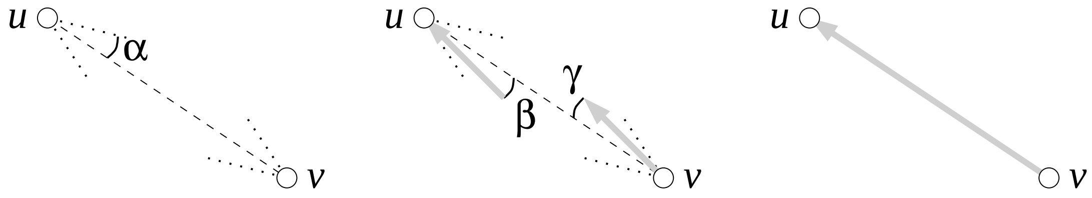

Figure 5.2: determining whether vertex v is compatible with vertex u

In this way we build the preliminary graph G_1; for every pair of vertices u and v, we check to see if u and v are compatible according to Definition 5.1. Whenever u is compatible with v, we include the edge uv in G_1.

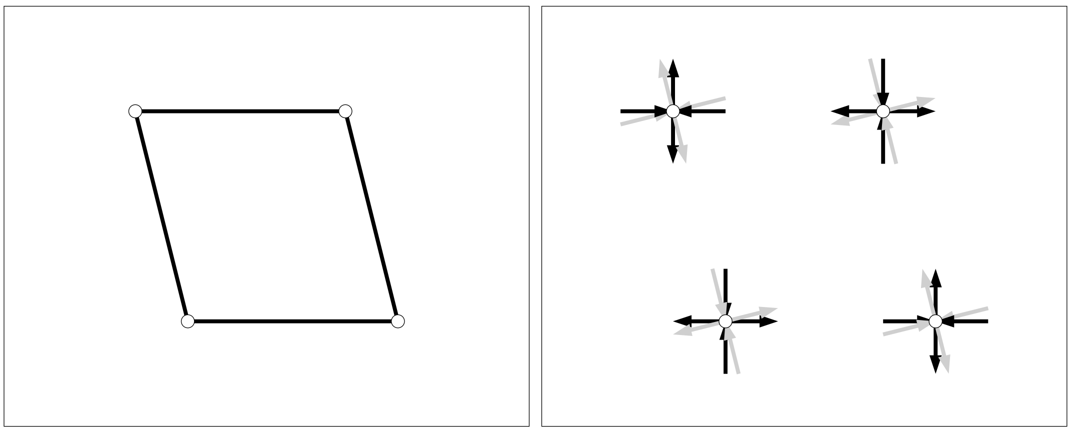

Figure 5.3: a parallelogram and its associated input position and axis data

For example, figure 5.3a displays a parallelogram. The vertex position and axis angle data which would provide input to `buildGraph` are displayed in figure 5.3b.

The `buildGraph` module proceeds to check all pairs of vertices for possible edges (figure 5.4a) to produce the graph G_1 (figure 5.4b) before passing it along the pipeline to the `edgeElim` module.

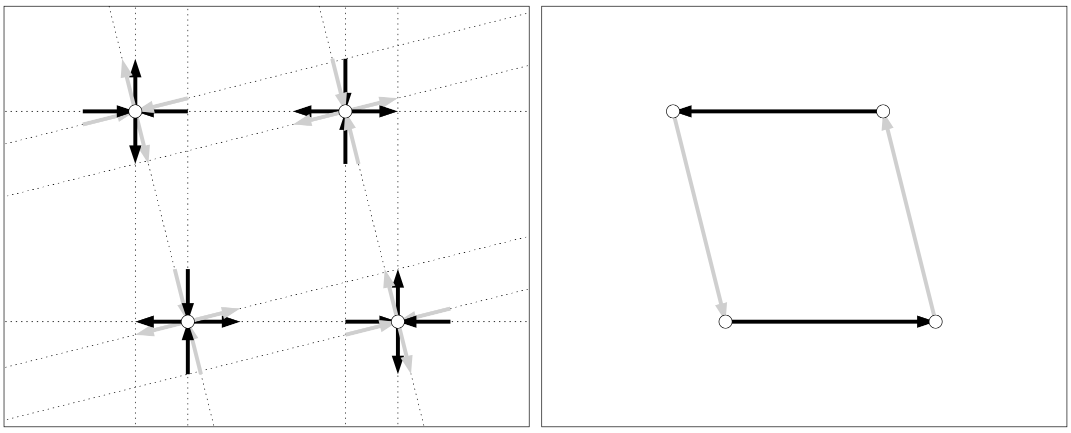

Figure 5.4: checking for possible edges between vertices

Thus, we complete the first phase of the pipeline, having constructed an edge-bicoloured graph G_1 of all compatible vertex pairs, $(u, v) \in V \times V$.

5.3 EdgeElim: Simplifying the Graph

The initial graph G_1 produced by `buildGraph` may be very dense. Any possible pairs of vertices which align are granted an edge between them. Whenever three or more vertices align in a common direction, if the in-edge and out-edge angles fall within the i tolerance, then a semi-complete interconnection may be produced. We would like to reduce the degree of G_1 to form a moment-derived edge-bicoloured spanning subgraph.

For example, figure 5.5a displays six co-linear vertices whose in-edge and out-edge

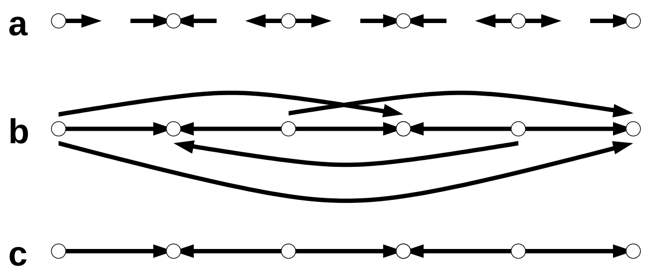

Figure 5.5: applying `edgeElim` to reduce edge density to constant degree

angles align. In this case, `buildGraph` will construct the high-degree graph G_1 displayed in figure 5.5b. We use `edgeElim` to reduce the graph to the desired subgraph G_2 displayed in figure 5.5c.

For every vertex $v \in G_1$, `edgeElim` reduces the degree of v such that there may be at most one edge of any given orientation and colour per hemisphere, where a hemisphere is defined to be half a rotation about the vertex or an angle of π radians. Thus, v will have at most one red in-edge, one red out-edge, one blue in-edge, and one blue out-edge per hemisphere. As required, v will conform to having no more than four edges per colour axis.

Our heuristic keeps only the shortest edge and eliminates all others along a common direction with similar orientation and colour (see figure 5.6).

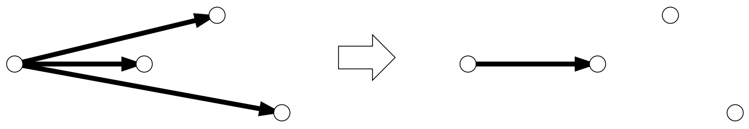

Figure 5.6: edge elimination heuristic

Obviously, depending upon the tolerance angle α in `buildGraph`, there will be instances when the heuristic eliminates the true edge and retains an invalid edge. This usually results in an equivalent solution. For example, figure 5.7a displays the

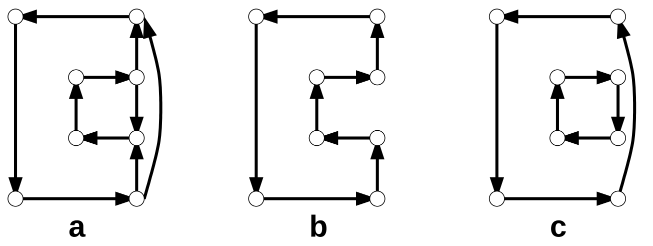

Figure 5.7: two different edge eliminations which produce equivalent results

initial reconstruction G_1 of an eight-vertex polygon which has four co-linear vertices on its right side. `buildGraph` has included additional edges along these co-linear vertices. The original polygon may have been one of two possible shapes: a simply-connected closed polygon (figure 5.7b) or a multiple-component rectangle with a square hole within it (figure 5.7c). In the latter case, the square hole is positioned very close to the right exterior edge of the rectangle. The two figures differ only by the additional thin strip which links the top and bottom to form a closed hole. The area of this thin strip is quite small with respect to the rest of the polygon, some small ϵ which depends upon the tolerance α and the scale of the figure. Reconstructing the former or the latter shape, therefore, gives a solution which is very close to the desired polygon. Furthermore, given the input information, we have no way of choosing a best solution between figures 5.7b and 5.7c. Thus, we apply our heuristic and eliminate all but the shortest edge along a common direction with similar orientation and colour to produce the second graph, G_2. In our example, this results in figure 5.7b being passed along to the next phase of reconstruction.

Thus, we complete the second phase of the pipeline, having simplified the graph G_1 such that each vertex has constant red-degree, blue-degree, in-degree, and out-degree within a hemisphere; `edgeElim` produces a moment-derived edge-bicoloured graph spanning subgraph, G_2.

5.4 Commit: Colour Constraint Propagation

After having passed through `recons`, `buildGraph`, and `edgeElim`, we are given an input graph G_2 whose vertices have bounded degree and whose edges fall locally into two geometrically-restricted axes as displayed in figure 5.8.

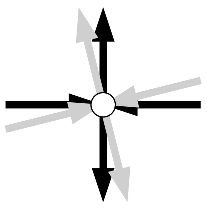

Figure 5.8: axes at a vertex from the graph G_2

We now begin the final phase of the reconstruction. Here, we are given a moment-derived edge-bicoloured input graph G_2 within which we wish to find a globally-happy 2-factor. The first method employed to achieve this end is through the use of the `commit` module.

The commitment algorithm performs a recursive walk through the graph, visiting each vertex no more than a constant number of times.[4] The walk propagates colour constraints between neighbouring vertices to eliminate edges and to reduce the graph in a greedy manner. After successive edge eliminations, constraints may no longer be propagated and three types of output graphs, G_3, may be produced:

1. G_3 may be a **solution** graph, that is, a globally-happy 2-factor.
2. G_3 may be an **invalid** graph containing one or more invalid vertices (see definition in Section 2.1.4). This case only arises when the original input graph, G_2, does not contain a solution subgraph.

[4]As we will see, vertices may only be revisited after some action, be it edge-commitment or edge-elimination, has occurred at a neighbouring vertex. Since vertices have a maximum degree of eight, no more than eight such visits may take place and, thus, the constant is exactly eight. Therefore, commit has linear worst-case runtime: $\Theta(|V|)$.

3. G_3 may be in a state where colour constraints can no longer be propagated. The algorithm requires two or more recursive search branches (see Section 5.5).

As mentioned, the essence of the commitment algorithm involves colour constraint propagation. The constraints arise from our requirement for global-happiness. For a vertex to be happy, it must have in-degree one, out-degree one, red-degree one, and blue-degree one. Initially, all edges are in the undecided state. After finding a 2-factor solution, all edges will either be committed edges or they will have been dropped. Thus, we work towards a solution by altering the state of an undecided edge to being committed or dropped. We do so by applying two types of rules: local elimination rules and new commitment rules.

Local elimination rules follow a basic format: at a vertex v, we check for vacancies and eliminate edges accordingly. If v does not have any blue in-edges, then v can never be in vertex state BIRO. Thus, v has no need for red out-edges and we eliminate all red out-edges at v (see figure 5.9a). Similarly, if v does not have

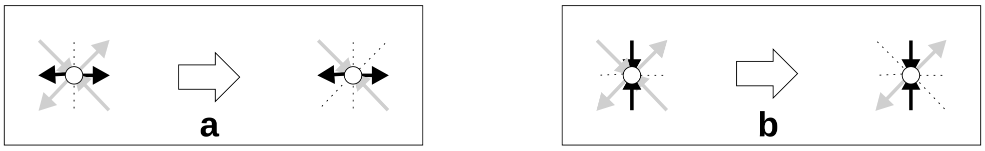

Figure 5.9: local elimination rules

any blue out-edges, then v can never be in vertex state RIBO, v has no need for red in-edges, and we eliminate all red in-edges at v (see figure 5.9b). We apply respective rules for absent red in-edges and red out-edges.

The local elimination rules involve checking for dropped edges. Conversely, we may check for committed edges and, hence, we develop **new commitment rules**. Recall that a committed edge is one which belongs to the final solution subgraph. Whenever a vertex has only one edge of a given colour or a given orientation, then that edge must be contained within a solution and, therefore, we commit to the edge. Edge-commitment affects the remaining edges of vertices at both endpoints of the newly-committed edge.

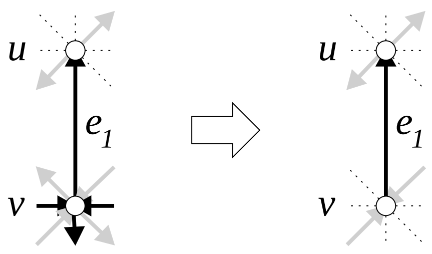

Figure 5.10: new commitment rule

For example, vertex u in figure 5.10 has a single blue edge, e_1. Since there are no other blue edges, we make e_1 a committed edge. Commitment at the head of an edge implies commitment at the tail. Thus, vertex v inherits a committed blue out-edge. This new commitment fixes the vertex state of v to RIBO. Vertex v can never be in state BIRO and no longer needs blue in-edges or red out-edges. All blue in-edges and red out-edges are dropped. We apply respective rules for all four combinations of committed edges.

Rules are applied locally at a vertex. Whenever the application of a rule involves a change in edge state, the neighbouring vertex along the affected edge must be alerted; its local edge set has been altered either by having lost an edge or by having one of its edges committed and, consequently, should be revisited.

Since the application of a single rule at a vertex can trigger changes at many neighbouring vertices, commitment propagation occurs quite rapidly in a highly reactive recursive fashion. We visit vertices one at a time. Upon visiting a vertex, we check to see if any of the local elimination or commitment rules apply and perform actions accordingly. In order to keep track of affected vertices which still need to be visited, we maintain a set of active vertices.

Thus, our algorithm falls into place. Initially, we know nothing about vertices and we must visit every vertex at least once. Therefore, our initial active set is simply the set of all vertices. We may visit any vertex in the set[5] and apply the colour rules to it. After having visited the vertex, we remove it from the active set and add any affected neighbours to the set. Once the set is empty, colour constraints have been propagated as much as the graph allows, and we return the current graph, G_3.

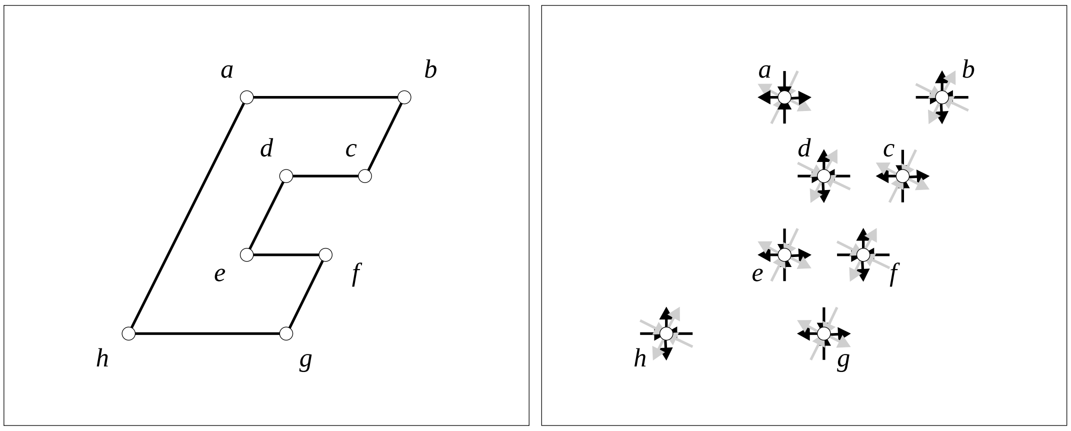

Figure 5.11: original polygon and input data after applying `commit`

Let us trace a simple example through the four phases of reconstruction while paying special attention to commitment propagation. The original shape from which moment data has been obtained is an eight-vertex, simply-connected closed figure (figure 5.11a). Given 16 moment measurements for this shape, `recons` returns a set of 8 vertex positions along with a blue and red axis orientation for each (figure 5.11b).

[5]To improve performance, we sort the active set by non-decreasing degree such that vertices of low degree are visited first. Vertices of lower degree d, $d \geq 3$, have greater potential to trigger propagation "reactions."

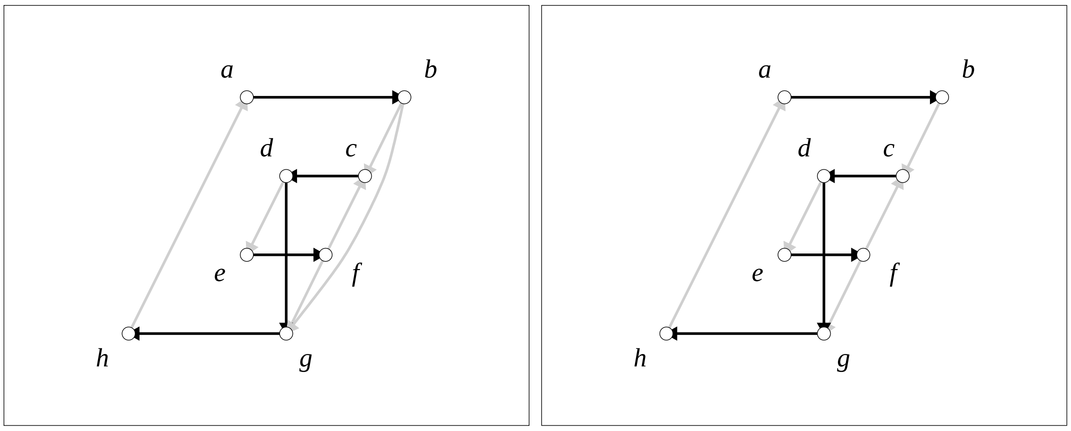

Figure 5.12: graphs G_1 and G_2 produced by `buildGraph` and `edgeElim`, respectively

Given these 8 positions and 8 pairs of axes, `buildGraph` produces the initial graph, G_1 (figure 5.12a). This graph includes edges for every possible pair of compatible vertices. `edgeElim` takes G_1 as input, simplifies the edge set by reducing the degree at every vertex, and produces the bounded-degree edge-bicoloured graph, G_2 (figure 5.12b).

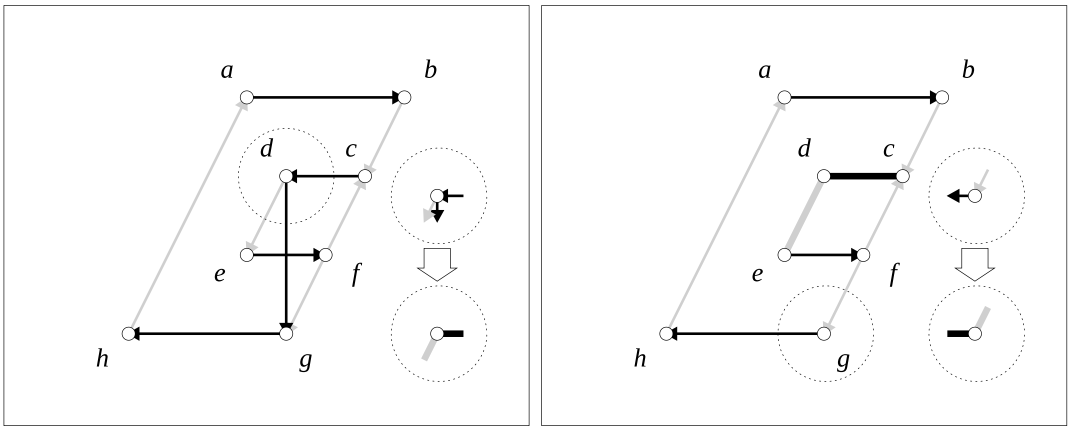

Figure 5.13: constraint propagation

The graph G_2 is passed to `commit` where commitment propagation begins. Initially, the active set, A, is the set of all vertices: $A = \{a, b, c, d, e, f, g, h\}$. We begin by visiting vertex d. Since d only has one red edge, a red out-edge de, then the vertex state at d becomes fixed to BIRO. We can therefore apply local elimination rules to remove all blue out-edges (see figure 5.13a). Since there are only two edges remaining at d, cd and de, we commit to each of them (represented by bold edges). Vertex d is then removed from A and affected neighbours, g in this case, are added to A: $A' = (A - \{d\}) \cup \{c, e, g\} = \{a, b, c, e, f, g, h\}$.

We then visit vertex g. Since g only has two edges, a red in-edge fg and a blue out-edge gh, we commit to each of these (see figure 5.13b). Once again, we update A: $A' = (A - \{g\}) \cup \{f, h\} = \{a, b, c, e, f, h\}$.

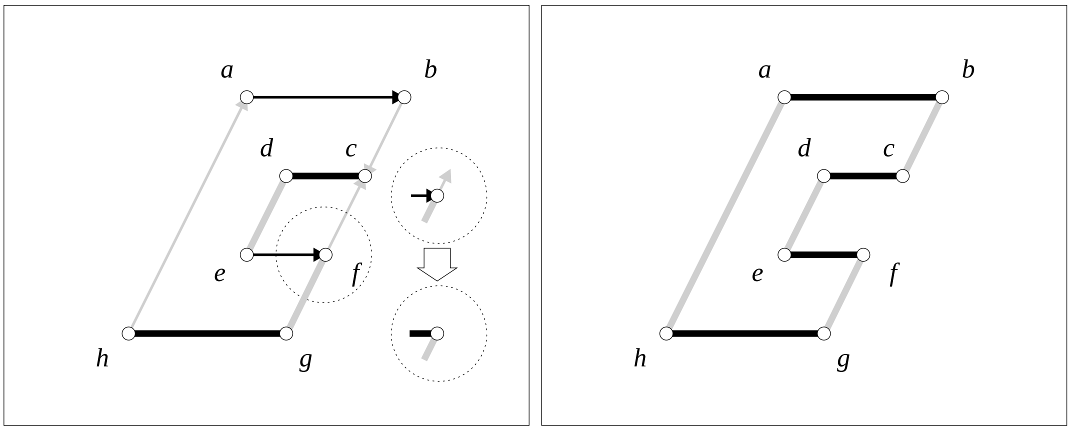

Figure 5.14: Constraint propagation continues on to produce a solution graph G_3.

We now visit vertex f. Vertex f has a committed red out-edge, fg. Vertex f also has another red out-edge, fc, which is undecided. Obviously, edge fc can be dropped (see figure 5.14a). Vertex f now only has two edges remaining, and we commit to each of these. We update A: $A' = (A - \{f\}) \cup \{e\} = \{a, b, c, e, h\}$.

The remaining edges form a globally-happy 2-factor (see figure 5.14b). The algorithm still requires visits to each of $\{a, b, c, e, h\}$ to commit to their edges. Note that a happy vertex whose edges are both committed cannot be added to the active set again since the state of its edges may no longer be altered. The algorithm terminates and the graph G_3 is returned as a solution.

For most graphs which embed a single unique solution, `commit` successfully finds and returns the solution subgraph. Figures 5.15 and 5.16 display more complex examples which were solved by commit. In each figure, the first plot is the graph G_2 returned by `edgeElim` and the second plot is the graph G_3 returned by `commit`. In

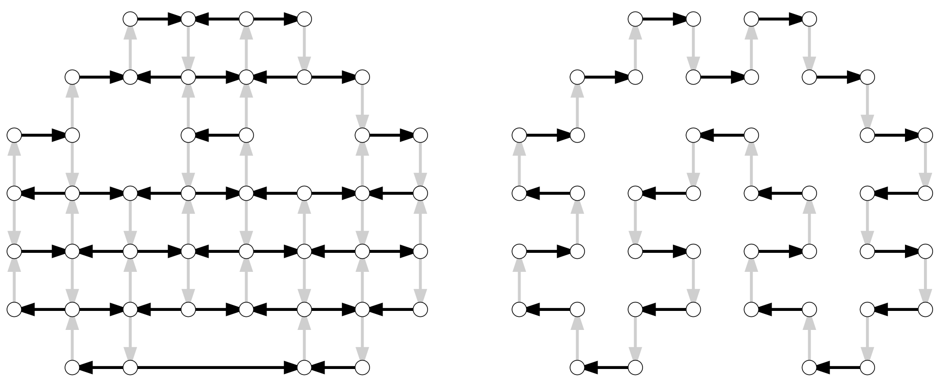

Figure 5.15: complex rectilinear polygon reconstructed by `commit`

figure 5.16a, only six vertices have initial edge configurations which may propagate constraints. All other 30 vertices require constraints to be passed to them.

The module `commit` searches for a globally-happy 2-factor through the propagation of colour constraints. This technique often succeeds in finding a solution. Sometimes, however, constraint propagation alone comes to a halt before having found a solution. For these cases, we require a recursive search tree.

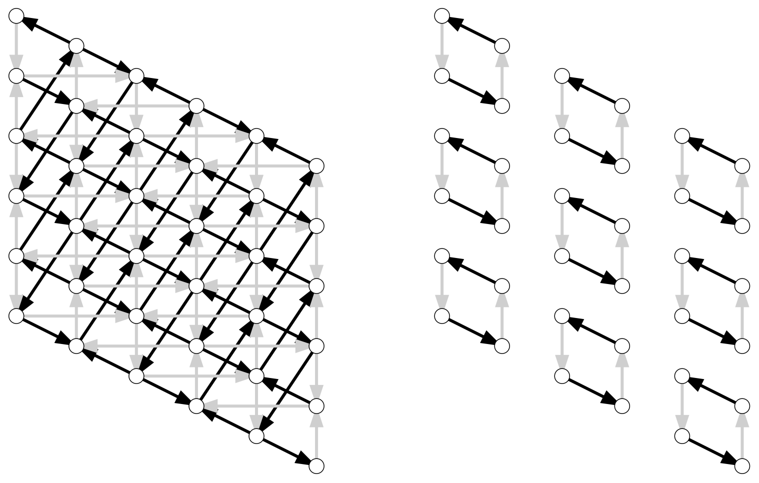
Figure 5.16: set of 9 boxes reconstructed by `commit`

5.5 TreeSearch: Recursive Search Trees

When the subgraph G_3 produced by `commit` is not a globally-happy 2-factor, constraints can no longer be propagated at any of the vertices of G_3. Hence, we require another mechanism to decide upon which edges belong to a solution.

Given a graph G_3 on which constraint propagation has been exhausted, the `treeSearch` module manages control of a simple backtracking approach which allows propagation to resume. We choose an undecided edge $e \in G_3$ and create two new graphs, G_3' and G_3'', where edge e is dropped in G_3' and committed in G_3''. We then proceed to solve the two separate problems of searching for solutions within G_3' and G_3'' using `commit`.

Again, we have the same three possible outcomes at the end of both of these branches. Either, we have a solution, the resulting graph is invalid, or constraint propagation has again been exhausted. If the last case arises, we simply repeat the process, branch again, and continue. All valid solutions are accumulated and returned as a set of possible solutions to the original input graph.

In selecting the candidate edge e, we search for a vertex of lowest degree greater than two and choose one of its edges. We do so because any vertex of degree two requires both of its edges in a solution. Forcing a new edge state at a vertex of degree three or greater, however, should trigger some constraint propagation, with lower degree vertices being more reactive in general. This basic heuristic for choosing edges works efficiently on most examples examined. Even if only minimal commitment propagation results from a decision, each vertex will only have been visited a constant number of times, and successive branching will eventually lead to a solution.

Let us examine the search tree for a graph which contains three ambiguous solutions. Branching within the search tree is displayed in figure 5.18.

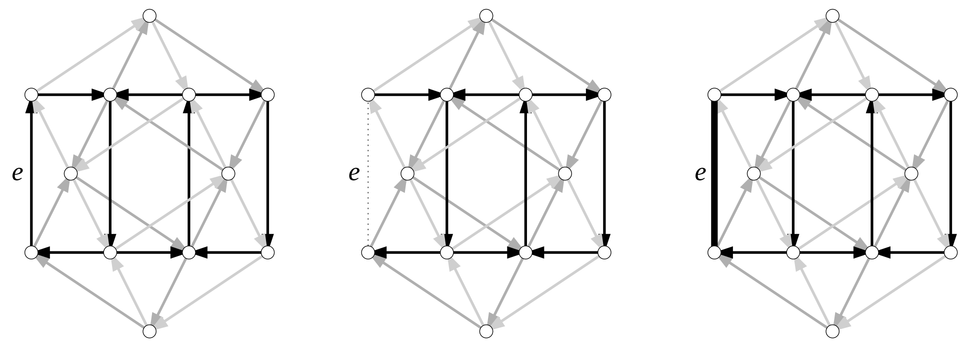

Figure 5.17: ambiguous input graph G_1 and two subgraphs, G_2' and G_2''.

After having passed through `recons`, `buildGraph`, and `edgeElim`, we enter the module `treeSearch`. The input graph, G_1, is displayed in figure 5.17a. G_1 is the hexagonal six-point star which we saw in Section 4.2. `treeSearch` begins by calling `commit` with G_1 as input (step 1). Every vertex $v \in G_1$ has at least one in-edge and one out-edge of each colour. Thus, an initial pass of `commit` fails to simplify G_1 and outputs an unaltered graph, $G_2 = G_1$.

At this point, `treeSearch` creates two modifications of G_2 (step 2). All vertices along the exterior of G_2 have lowest degree, in this case, 4. Thus, we select an undecided edge at one of these vertices, say edge e. G_2' is set to be G_2 except with edge e dropped (figure 5.17b) and G_2'' is set to be G_2 except with edge e committed (figure 5.17c).

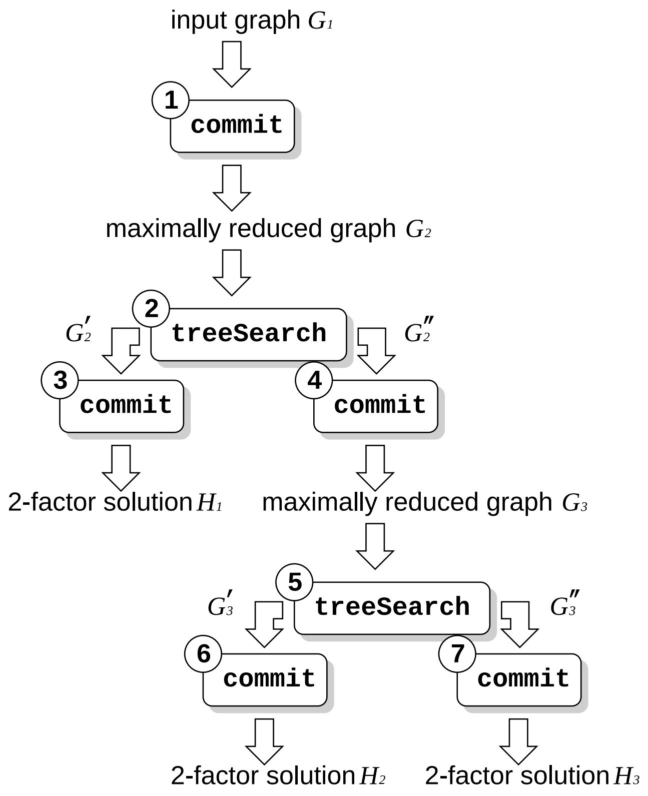

Figure 5.18: recursive search tree descent

We now branch for the first time. Two parallel recursive calls are made to `commit`, one on input graph G_2' (step 3) and the other on G_2'' (step 4).

Within the left branch (step 3) constraint propagation reduces the graph to a solution subgraph, H_1, displayed in figure 5.19a. Within the right branch (step 4) however, we reach another standstill: the maximally reduced subgraph, $G_3 \subseteq G_2$, displayed in figure 5.19b.

`treeSearch` creates two modifications of G_3: G_3' and G_3'' (step 5). Again, we look for a vertex of lowest degree greater than two. Every exterior vertex has degree

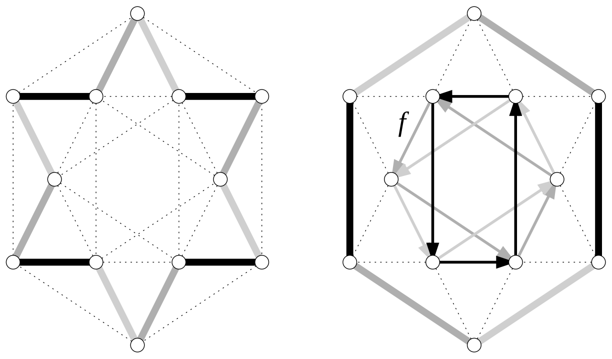

Figure 5.19: solution H_1 and maximally reduced subgraph G_3

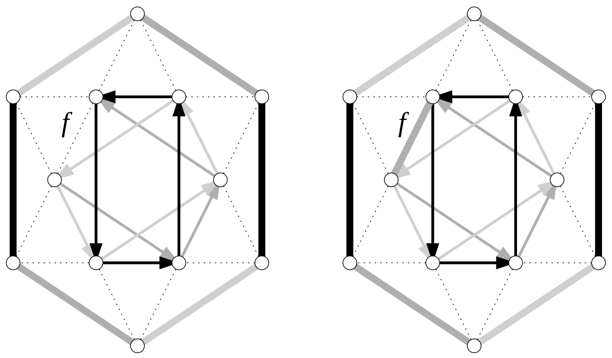

Figure 5.20: two subgraphs of G_3: G_3' and G_3''

two and every interior vertex has degree four. Therefore, we select an undecided edge from one of the interior vertices, say edge f. G_3' is set to be G_3 except with edge f dropped (figure 5.20a) and G_3'' is set to be G_3 except with edge f committed (figure 5.20b). We now branch for the second time. Again, two parallel recursive calls are made to `commit`, one on input graph G_3' (step 6) and the other on G_3'' (step 7).

After propagating constraints, these two calls emerge with solutions H_2 and H_3, respectively (see figure 5.21). All three solutions produced are valid, globally-happy 2-factors of the original graph. Given our input information, any of these

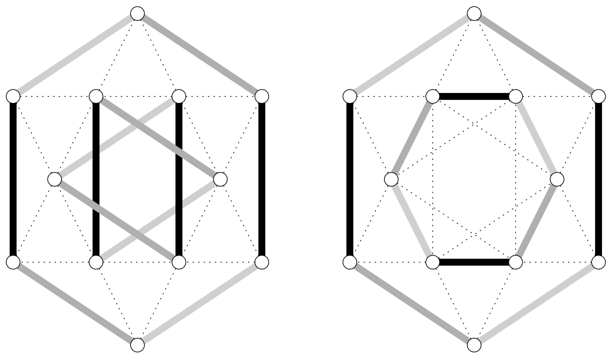

Figure 5.21: two solutions H_2 and H_3

three is as good a solution as the other. When discerning between ambiguous possibilities in `edgeElim`, alternative cases had solutions which differed in area only by some small ϵ. Non-unique solutions produced by `treeSearch`, however, may differ greatly in area, connectedness and edge-crossing. In our example, solution H_1 is a simply-connected, non-crossing polygon, solution H_2 is a multiple-component, crossing polygon, and solution H_3 is a multiple-component, non-crossing polygon.

Whenever constraint propagation alone fails to provide a solution, `treeSearch` successfully reconstructs all solutions, regardless of the complexity of the input graph. For example, figure 5.22 displays the graphs G_1 and G_2 which are produced by `buildGraph` and `edgeElim`, respectively, when given the input vertices for a two-connected, extended Strakhov-Brodsky graph. In this case, `treeSearch` produced the actual output plots displayed in figure 4.11 (see Section 4.2).

Thus, through recursive calls to `commit` down a search tree, our `treeSearch` module allows the reconstruction of all solutions contained within a subgraph.

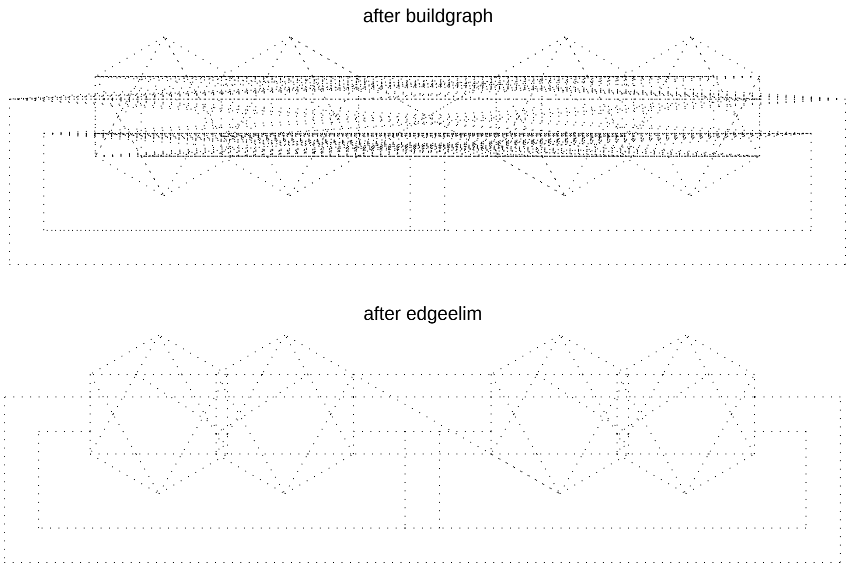

Figure 5.22: `buildGraph` and `edgeElim` results for two-connected Strakhov-Brodsky extension

5.6 FragmentSearch: Finding Fragments of Solutions

The reconstruction modules `commit` and `treeSearch` run under the assumption that all required information lies embedded within the input graph. That is, all edges of a solution 2-factor must be edges of the graph G_2. Imperfect data, however, may provide a less than perfect input graph for G_2. A single missing edge can easily trigger an erroneous chain of constraint propagation which results in an invalid graph.

For domains which involve imperfect input data, we need to relax our colour constraint rules and modify our search strategy. We have investigated two such techniques which were implemented in `fragmentSearch` and `momentDiff`. We begin by discussing module `fragmentSearch`.

Instead of looking for a globally-happy 2-factor, we search for fragments of happy vertices. Thus, we seek portions of a solution which, when pieced together with missing edges, form potential reconstruction solutions. Unlike solutions found by `commit` and `treeSearch`, solutions constructed by `fragmentSearch` are of a more subjective nature; `treeSearch` will return every globally-happy 2-factor contained within a graph, whereas `fragmentSearch` must heuristically decide between possible collections of fragments contained within a graph, and return those which might form pieces of a solution. Furthermore, in `fragmentSearch`, eliminating an edge e may cause a vertex v to become invalid. Thus, unlike `commit` and `treeSearch`, in which ordering of events does not affect constraint propagation in the solution process, solutions produced by `fragmentSearch` are subject to the ordering of elimination and commitment actions.

An input graph to `fragmentSearch` may include invalid vertices, that is, vertices which may be without a given colour or orientation of edge (see figure 5.23). In this new algorithm, constraint propagation must account for such vertices. Thus,

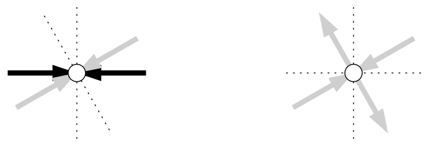
Figure 5.23: two invalid vertices

propagation is altered as follows: whenever we visit a vertex v from the active set A, we apply commitment and elimination rules as before, but only if v is valid. Upon visiting an invalid vertex u, we simply remove u from A and continue. We retain our search tree approach and branch recursively on two calls to `commit` until the output graph embeds a collection of happy vertex fragments.

When comparing two solutions, we use a simple heuristic and compare the lengths[6] of the longest fragment contained within each. If these have equal length,

[6]Here, "length" refers to the number of vertices in a fragment and not to geometric length.

we compare the next longest, and so on. Obviously, the solution with the longer fragment may not always be the better one; generally, however, a solution composed of one long fragment of k vertices provides a closer partial reconstruction than a different solution composed of several shorter fragments.

In most examples, the fragment approach efficiently locates correct portions of solutions. For example, figure 5.24a displays the reconstructed vertices of an E-shaped polygon produced by `recons`. Note the irregularities in vertex position and potential angle data, especially near the middle of the figure. Figures 5.24b and 5.24c display the graphs G_1 and G_2 produced by `buildGraph` and `edgeElim` respectively. Neither G_1 nor G_2 contain a globally-happy 2-factor. Thus, as expected, when passed through `commit` and `treeSearch`, no solution is found. The `fragmentSearch` module, however, finds the fragment displayed in figure 5.24d. Upon observation, we note that G_2 does not embed any longer fragment. Thus, given our limited input constraints, `fragmentSearch` provides a valid partial solution.

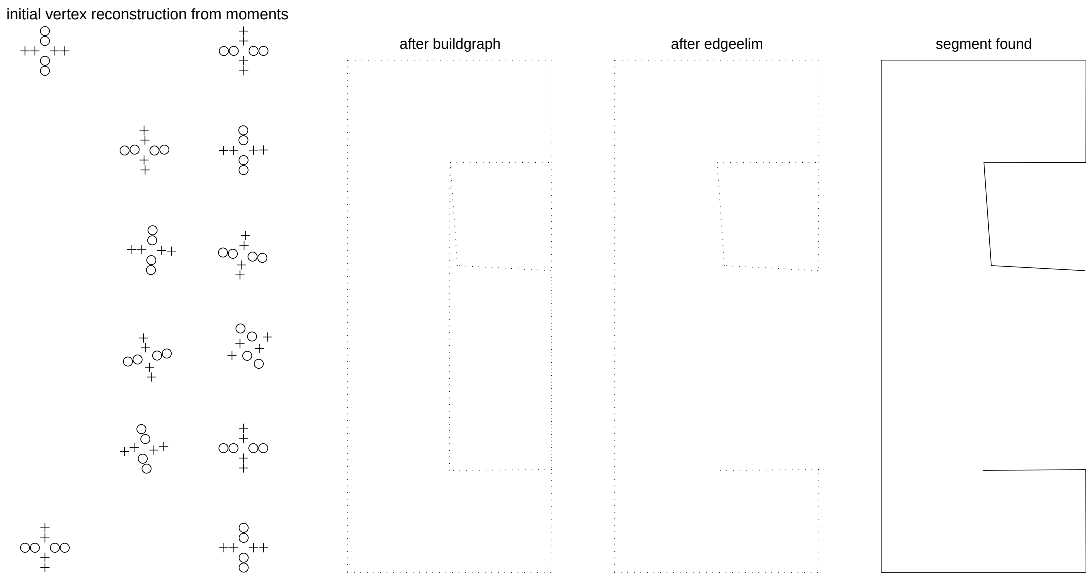

Figure 5.24: 9-edge fragment found in 12-vertex polygon

Whenever the input domain includes imperfections in the data such that a globally-happy 2-factor is not contained within the input graph, we may use `fragmentSearch` to reconstruct partial solution composed of fragments of happy vertices. Such a partial solution often provides a close approximation to the original polygon and allows the reconstruction to overcome some degree of error arising from numerical computation or input data.

5.7 MomentDiff: Component Reconstruction by Difference of Moments

The partial solution provided by `fragmentSearch` identifies portions of a graph in which data is more likely to be accurate. A long reconstructed fragment usually follows a string of vertices whose position and angle data closely match those of the original polygon. Long fragments of accurate vertices can be used along with the original moment input data in subsequent iterations of the reconstruction pipeline to obtain better position and angle approximations for inaccurate vertices.

Given a polygon, P, and its moment vector, m_P, for any subdivision of P into non-intersecting subcomponents, $P_1, \ldots, P_k$, the sum of the moment vectors of the subcomponents of P is exactly equal to the moment vector of P: $m_P = m_{P_1} + \ldots + m_{P_k}$. For example, figures 5.25a and 5.25b display a simple polygon P and a two-piece decomposition of P into P_1 and P_2. If we separately measure n moments for P, n moments for P_1, and n moments for P_2, we find that $m_P = m_{P_1} + m_{P_2}$.

Module `momentDiff` implements the following algorithm. Using a difference of moments, we can reconstruct a polygon, one component at a time. The reunion of all reconstructed components forms the complete original reconstructed polygon. Given a set of moments, m_P, which correspond to an n vertex polygon P, $n \geq 4$, we use `recons`, `buildGraph`, and `edgeElim` to create an n-vertex input graph, G_1.

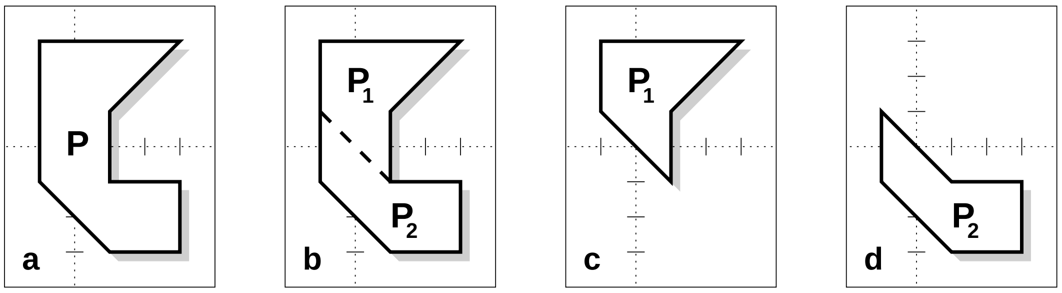

Figure 5.25: sum of subcomponent moments: $m_P = m_{P_1} + m_{P_2}$

A call to `fragmentSearch` on input G_1 returns a partial solution composed of edge fragments. Let us assume the longest fragment, S, has k vertices, $s_1, \ldots, s_k$, $3 \leq k < n$. We form a new polygon P_1, by taking the endpoints s_1 and s_k, and creating a new edge $s_1 s_k$. Polygon P_1 is a closed cycle of edges which corresponds to the closure of the edges of fragment S. If the vertices of S are accurate with respect to corresponding vertices within the original polygon P, then P_1 is a subcomponent of P. We find the moments of P_1, m_{P_1}, and a new difference of moments, $m_\Delta = m_P - m_{P_1}$.

We call `recons` again with this new difference of moments, m_Δ, as input. The output vertices are run through `buildGraph` and `edgeElim` to produce a reconstructed graph G_2 of $n - k + 2$ vertices which correspond to the component $P_2 = P - P_1$. We search within G_2 using `fragmentSearch` for another longest partial solution. If G_2 contains a globally-happy 2-factor, then we have successfully reconstructed P_2 and we may assemble P by joining P_1 and P_2. If not, then we take the longest fragment within G_2, and repeat this process recursively.

Let us examine a simple example. As we saw in Section 5.6, the reconstruction of a 12-vertex, E-shaped polygon P includes some inaccurate vertex positions and angles which initially prevent a complete solution from being discovered. `fragmentSearch` successfully finds the 9-edge fragment S_1 displayed in figure 5.26a.

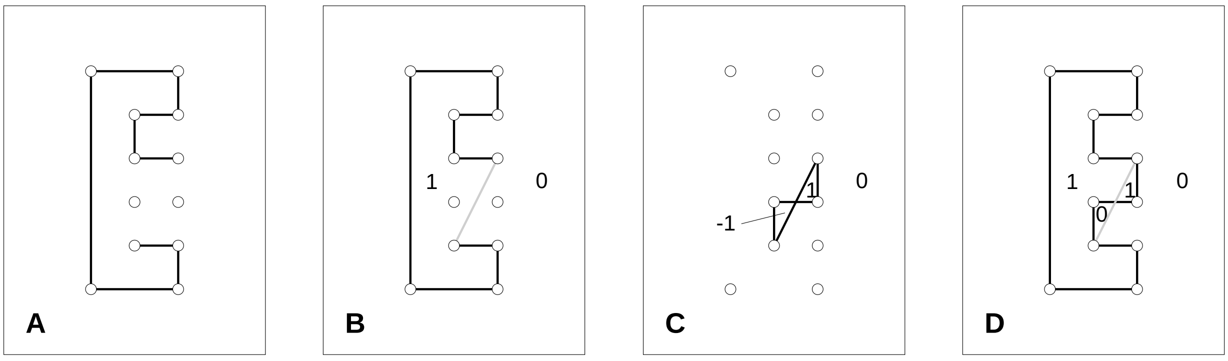

Figure 5.26: reconstruction by component decomposition

Module **`momentDiff`** takes the closure of S_1 (figure 5.26b) finds its moments, m_{S_1}, and the difference of moments, $m_\Delta = m_P - m_{S_1}$.

Upon a second pass through the pipeline, the reconstruction of m_Δ forms the four-vertex polygon P_2 displayed in figure 5.26c. If we adopt the premise that clockwise fragments of closed edges bound areas of positive density and counter-clockwise fragments of closed edges bound areas of negative density, then we find that the union of P_1 and P_2 forms an exact reconstruction of P (figure 5.26d).

We must note, however, that at this point in time, our experiments using differences of moments as a method for polygon reconstruction have produced successful results only under very controlled conditions. Errors in moment differences produce "ghost" vertices which correspond to the differing regions between the perimeters of the two polygons (see Section 7.4). These additional vertices easily fool the algorithm when searching for the second polygon, P_2.

For imperfect input domains, we successfully reconstruct partial solutions by searching for fragments of happy vertices. Reiteration of the reconstruction process on differences of moments provides interesting possibilities for piecewise polygon reconstruction, and for complete solutions on these input domains.

5.8 Summary

The process of reconstructing a polygon from its complex moments involves a multiple-phase pipeline of algorithms. Given a set of moments, we build vertex positions and potential edge angles using module `recons`. We then construct a dense edge-bicoloured graph G_1 of all possible pairs of compatible vertices using module `buildGraph`. We simplify G_1 into a moment-derived edge-bicoloured graph G_2 using an edge elimination heuristic in module `edgeElim`. Given that the input domain assumes accurate and complete information, modules `commit` and `treeSearch` allow the reconstruction of all globally-happy 2-factor solutions, G_3. Whenever the input domain includes inaccurate or incomplete information, modules `fragmentSearch` and `momentDiff` provide partial solutions and attempts at piecewise reconstructions of complete solutions. Together, these algorithms present various approaches to the polygon reconstruction from moments problem.

Chapter 6

From Models to Real Data

Upon being presented real moment data, the reconstruction of vertex positions and local angle axes often acquires some inaccuracies from numerical computation error and imperfect input moment data. Certain types of vertices seem to be affected more than others upon encountering inaccuracies. In this chapter, we examine the various levels of control we may impose on input data (Section 6.1) we briefly discuss the success of reconstruction on specific families of graphs (Section 6.2) and, finally, we discuss the effect of varying the number of moments provided as input to the reconstruction (Section 6.3).

6.1 Varying Control of the Input Data

Within the reconstruction process, four decreasing levels of control may be imposed on input data to module `buildGraph`. These determine the degree to which we fix the vertex position and potential angle information being passed on from module `recons`. Varying the level of control allows us to test the functionality of the reconstruction algorithms under various environments ranging from ideal to realistic conditions. We differentiate between the degrees of control as follows:

1. **exact vertex and angle information**: We may fix both position and angle data such that information passed onto `buildGraph` is exact.

2. **exact vertex locations and exact moments**: We may fix only vertex positions and reconstruct potential angle axes in `recons` using this positioning. Thus, half of the input data passed onto `buildGraph` is exact and half is reconstructed.

3. **exact moment data**: We may use perfect moment data as input to `recons` and reconstruct both position and angle data to be passed onto `buildGraph`.

4. **actual moment data**: We may acquire actual moment data, containing measurement imperfections, to be used as input to `recons`. Position and angle data reconstructed from these moments is then passed onto `buildGraph`.

Within our research, we examined the performance of our algorithms on the input domains of exact vertex and angle information and exact moment data.

6.2 Examining the Behaviour of Various Graphs

By nature of the quadrature formula, vertices within a polygon that are co-linear or interior often trigger numerical computation errors in the reconstruction; positioning and potential edge angle reconstruction for these vertices often includes significant inaccuracy.

Inaccuracies arise quite differently in slightly different graphs. For example, figure 6.1 displays the vertex positions and potential edge angles reconstructed by module `recons` for two graphs, G_1 and G_2, where G_1 is a six-point star and G_2 is an eight-point star. In each plot, the original polygon whose complex moments provided input to module `recons` is displayed in solid lines overtop the reconstructed data. Both G_1 and G_2 contain sets of four co-linear vertices as well as many interior

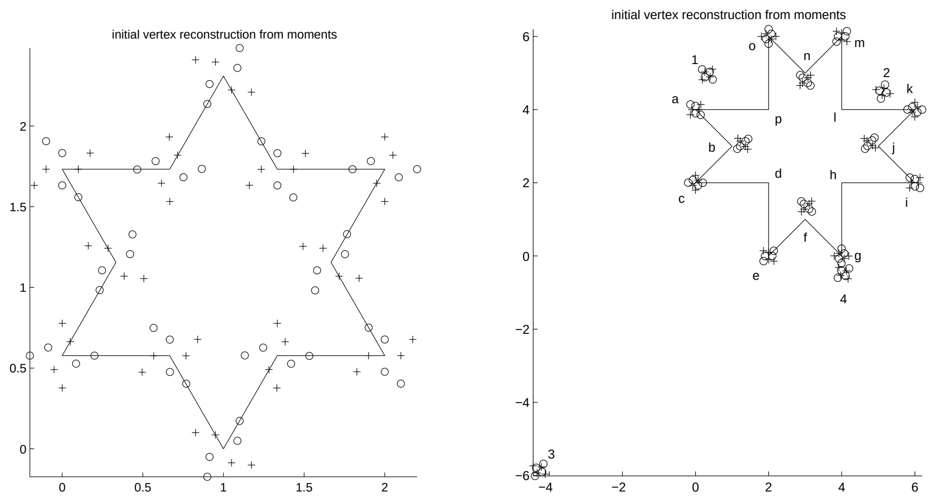

Figure 6.1: position and angle reconstructions for two graphs G_1 and G_2

vertices. Reconstructed vertex positions and potential angles are reconstructed accurately in graph G_1. In graph G_2, however, only exterior vertices are reconstructed accurately: $\{a, c, e, g, i, k, m, o\}$. Interior vertices either include minor inaccuracies, $\{b, f, j, n\}$, or extreme, irrecoverable error, $\{d, h, l, p\}$. In the case of the last four vertices, their reconstructed positions appear at points 1, 2, 3, and 4.

Thus, when working specifically with the graph-theoretic aspects of the reconstructions algorithm on graphs such as G_1, we fix the output from module `recons` and input perfect information to module `buildGraph`. When examining error-recovery abilities and partial solution search heuristics within the reconstruction algorithms, we input actual position and angle information from module `recons`, allowing inaccuracy to enter the input domain.

Certain families of shapes demonstrate interesting behaviour. The success of reconstruction directly depends upon the relative location from which moments are measured within the plane.

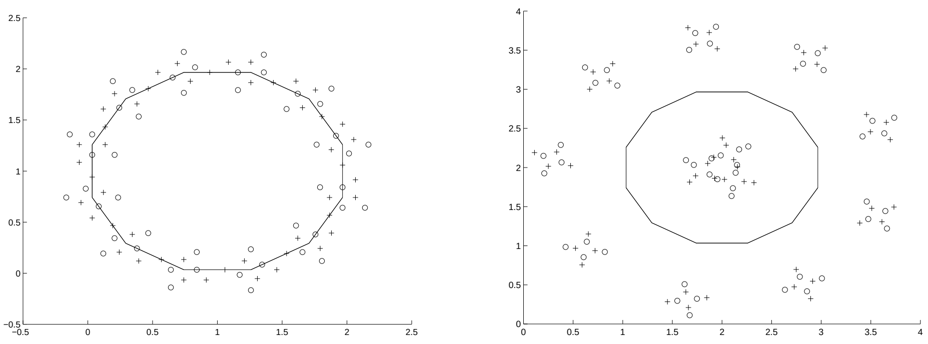
Figure 6.2: a northeast translation of polygon P by $+(1, 1)$

For example, given a 12-sided regular polygon of radius 1, P, figure 6.2 displays two reconstructions of the vertices of P. In figure 6.2a, P is centered at $(1, 1)$. In figure 6.2b, P is centered at $(2, 2)$. In both figures, the original polygon is plotted in solid lines. When centered at $(1, 1)$, vertices of P are accurately reconstructed. When centered at $(2, 2)$, however, the reconstruction becomes indecipherable.

If we center a regular polygon P at the origin, the reconstruction remains perfect. Upon attempting to reconstruct a subcomponent of this same P, however, we again notice the introduction of inaccuracies. Figure 6.3a displays the reconstructed positions of the upper right 12 vertices of a regular 48-sided polygon centered at the origin. Figure 6.3b displays the results after processing by `buildGraph` and fig-

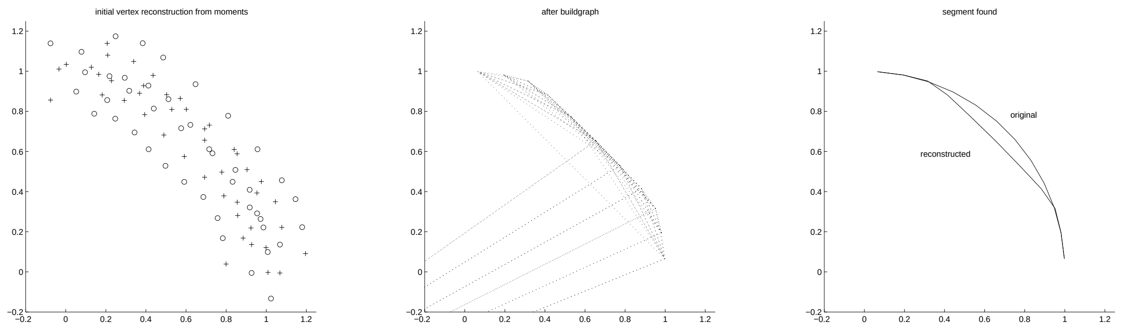

Figure 6.3: top right corner of a regular 48-sided polygon

ure 6.3c displays the results after processing by `edgeElim` and `fragmentSearch`. Note that one outlying vertex lies out of the displayed domain. Interestingly, a polygon including all 48 vertices can be reconstructed accurately, whereas as subset of size 12 cannot. Thus, subsets of sets of accurately reconstructible vertices are not necessarily accurately reconstructible themselves.

6.3 Dealing with Numerical Error in the Reconstruction

Polygons that present challenges to the reconstruction require us to consider alternative approaches to reconstruct their vertex positioning and potential angles. Experimentation demonstrates that reconstruction inaccuracies may be reduced by increasing the number of moment measurements input to module `recons`.

For example, figure 6.4 displays our familiar 12 vertex, E-shaped polygon P_1 whose vertices are independently reconstructed three times using an increasing number of moments. In figure 6.4a, module `recons` is given 8 complex moments and produces 8 vertex positions and axis angles. Undersampling provides an idea as to the general positioning of the polygon within the plane. In figure 6.4b, module `recons` is given 12 complex moments and produces 12 vertex positions and axis angles. In this case, reconstructed vertices lie close to actual positions but some

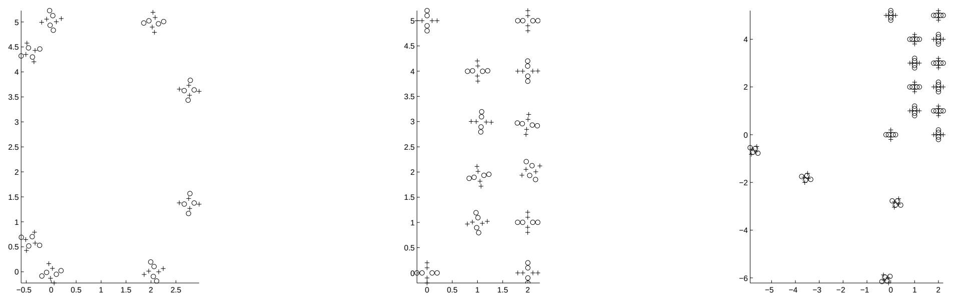

Figure 6.4: varying the number of sample points: n = 8, 12, and 16

angles still contain significant inaccuracies. Finally, in figure 6.4c, module `recons` is given 16 complex moments and produces 16 vertex positions and axis angles. Thus, by oversampling, all vertices and angles of P_1 are accurately reconstructed.

Increasing the number of samples usually improves reconstruction, but the results are not always as successful as for P_1. For example, figure 6.5 displays an extended E-shaped polygon, P_2. Here, we display the vertex reconstruction for increasing numbers of moments: $n = 16$, 24, 32, and 64. Note that extra reconstructed vertices lie close to a circle centered at origin.

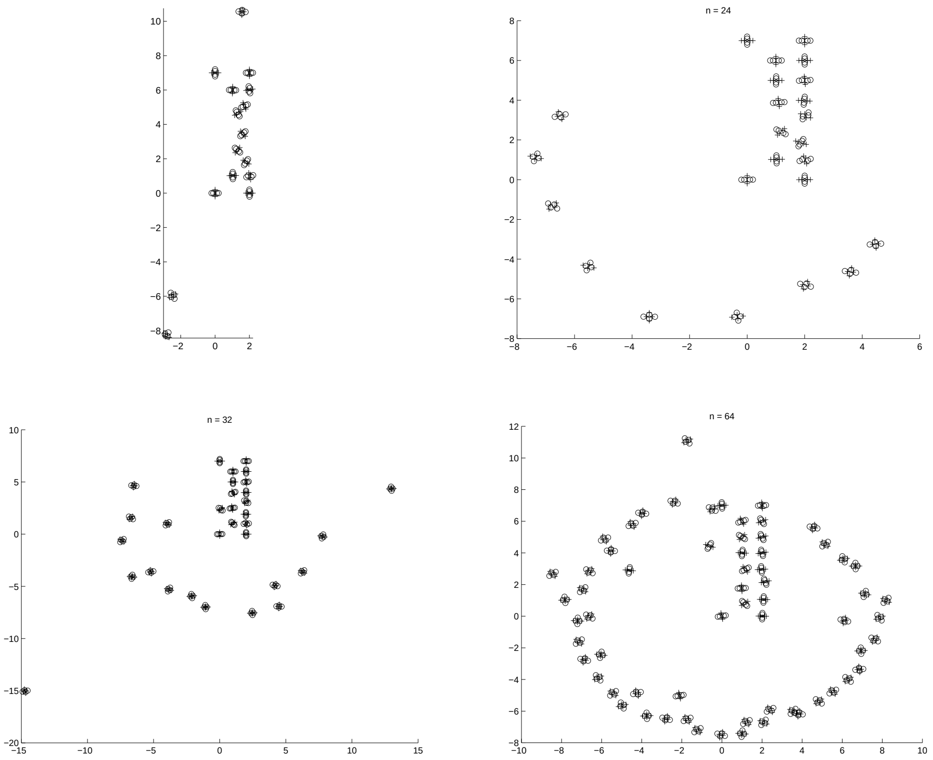

Figure 6.5: varying the number of sample points: $n = 16$, 24, 32, and 64

6.4 Summary

Given complex moments, reconstruction of vertices sometimes includes inaccuracies. For some vertices, these imperfections remain recoverable and may be dealt with from within the domain of graph theory. In these instances, inaccurate vertices are left unaltered and error tolerance within the reconstruction algorithms allows successful reconstruction. Other vertices, however, acquire significant position or angle error. For these cases, we require alternative methods of reconstructing position and angle, since we are provided too little information from the initial reconstruction.

Chapter 7

Further Research

Various partially explored and even completely unexplored questions branch out from the problems analyzed within this thesis. Among these possible research paths are the following interesting areas:

- Within our research, density is assumed to be uniform. We may consider the reconstruction of polygons within domains of variable density (Section 7.1).
- When given two solutions to a reconstruction problem, we may wish to certify the validity of each answer, measure the closeness of the moment vectors of the two solutions, and choose a best solution between the two (Section 7.2).
- Upon encountering inaccurate information, iteration of the initial reconstruction formula could be used to create additional constraints using partial results. This added input information might provide more accurate position and angle data (Section 7.3).
- Block reconstruction by differences of moments as currently implemented only allows very limited functionality. Greater error-tolerance within module `momentDiff` could provide improved reconstruction of inaccurate input data (Section 7.4).
- Current theorems about **NP**-hard restricted 2-factors could be used to show

NP-hardness for the problem of searching for a globally-happy 2-factor within a moment-derived edge-bicoloured graph (Section 7.5).

In this chapter, we briefly visit each of these potential research areas and describe the motivating ideas from which they derive.

7.1 Variable Internal Density

All models examined include polygons of constant density. Area within the polygon is assumed to be bipartite, such that, for any point x in the Cartesian plane, only two states are possible: either $x \in P$ or $x \notin P$. Real-world polygons, or 2-dimensional polygonal slices of 3-dimensional objects, most definitely include varying degrees of density. Exploring reconstruction issues among such polygons might provide insight into additional realistic and application-oriented examples. Figure 7.1 displays three shapes, A, B, and C, which have components of different density. These represent three models that may be assumed. Densities may be assumed to be discrete units, or low positive integer values, as in example A. Alternatively, densities may be assumed to be real positive values, as in example B. Finally, reversed loops within a polygon may be interpreted as negative densities, as in example C.

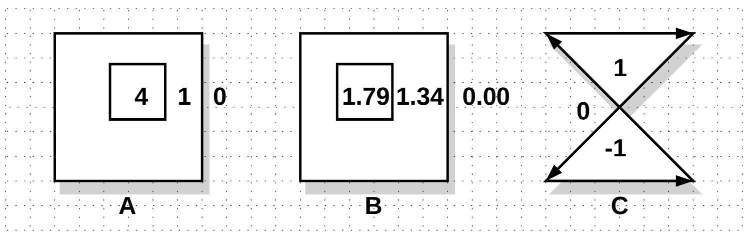

Figure 7.1: two shapes with multiple densities

Thus, issues of varying or non-uniform density remain to be considered within polygon reconstruction from moments.

7.2 Certifying Solutions

Given two solutions, A and B, how do we measure which is a better solution? Within module `fragmentSearch`, we discussed a heuristic that chose the solution with the longer fragment of happy vertices. Certainly, many other comparisons are possible. How do we differentiate between levels of validity? For example, figure 7.2 displays four solutions discovered by module `fragmentSearch` on $\alpha = .15\pi$ and moments for an E-shaped polygon. Which is the better solution?

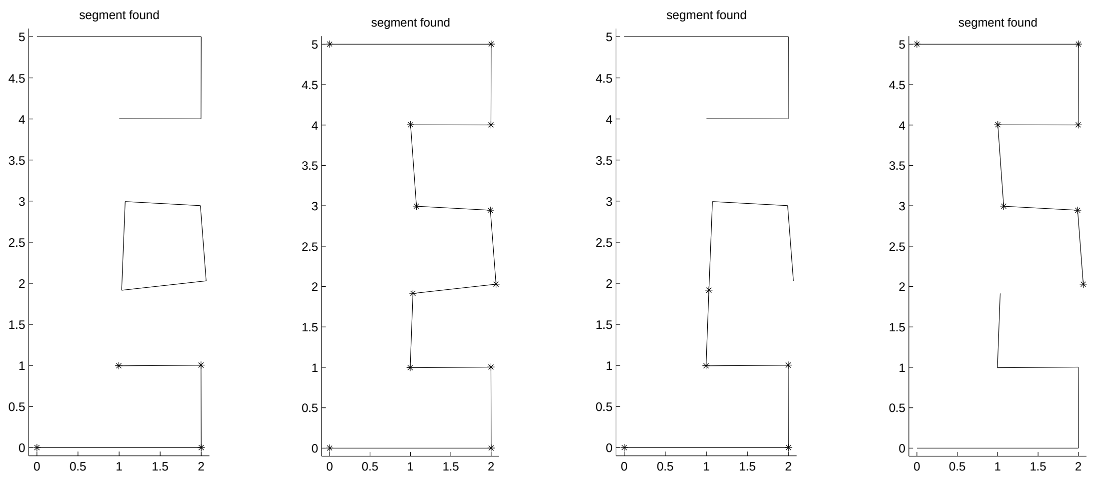

Figure 7.2: differentiating between partial solutions

One may hypothetically derive some measure of closeness by comparing moments of the reconstructed polygon to the original input moments. Perhaps statistical measurement of the differences between moments can provide a numerical description of this closeness. Thus, comparative certification of solutions, remains a mostly unexplored domain within polygon reconstruction from moments.

7.3 Iterative Reconstruction

Reiteration of the quadrature formula on a greater number of constraints may provide a reconstruction that more closely resembles the original polygon. After an initial run of `recons`, upon encountering inaccurate information, vertices deemed accurate or inaccurate could be tagged and divided into corresponding partitions. Positions of vertices from the partition of accurate vertices could then be used as additional constraints in a subsequent iteration of the reconstruction formula. The motivation being that additional information in a reiteration on a greater number of constraints may provide more accurate reconstruction for those vertices initially in the inaccurate partition.

For example, figure 7.3 displays a polygon P and its reconstruction P'. Assuming we have a mechanism for identifying inaccurate vertices, we separate vertices

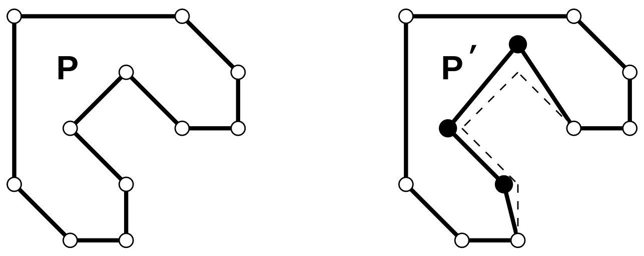

Figure 7.3: identifying accurate and inaccurate vertices

into two partitions: accurate (white) and inaccurate (black). The positions and edge angles of vertices in the accurate partition are then used as additional input constraints within a subsequent iteration of `recons`. This second call to `recons` could hypothetically provide a better reconstruction for the vertices of the inaccurate partition.

7.4 Improving Error Tolerance using Differences of Moments

Instead of attempting to improve the original vertex and angle positions reconstructed by `recons`, we may attempt to improve error tolerance within module `momentDiff`. Currently, differences between moments of the original polygon and subcomponents of the reconstructed polygon result in noisy errors within moments causing ghost vertices.

For example, figure 7.4a displays a polygon Q. Let us assume `fragmentSearch` successfully finds the partial solution fragment in figure 7.4b. Module `momentDiff` takes the closure of this fragment, Q_1, and searches for a reconstruction matching the difference of moments. Note that the position of vertex v is inaccurate. In attempting to reconstruct the six-vertex polygon Q_2, shadows of the difference Q_3 (figure 7.4c) cause error in the reconstruction. Hypothetically, ghost polygonal re-

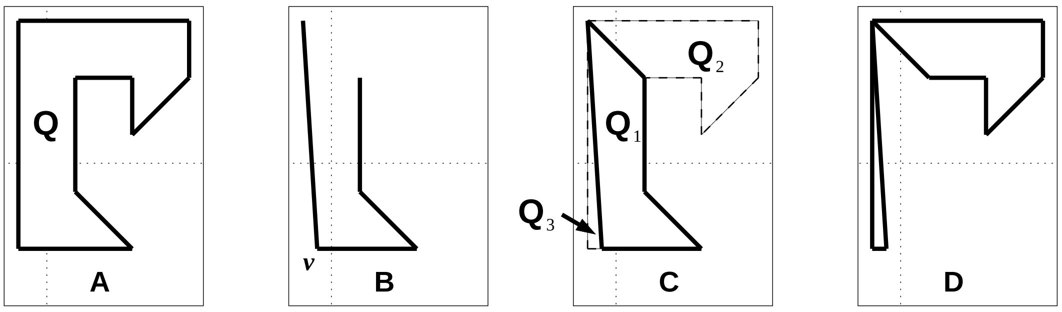

Figure 7.4: error in differences of moments

gions such as Q_3 may be identified and reconstructed. In our example, if vertex v were known to have inaccurate position, then module `momentDiff` could search for two subcomponent polygons, Q_2 and Q_3 (figure 7.4d).

7.5 Additional Complexity Issues

In Chapter 3, we proved that restricting the 2-factor problem in ways which resemble aspects of the polygon reconstruction problem are **NP**-hard. The actual problem involves finding a globally-happy 2-factor within a moment-derived edge-bicoloured graph. Our proof demonstrated that searching for a globally-happy 2-factor within a bounded-degree edge-bicoloured graph is **NP**-hard. Thus, we ask ourselves the question, can we incorporate geometric orthogonality constraints into a reduction to prove that searching for a globally-happy 2-factor within a moment-derived edge-bicoloured graph is also **NP**-hard?

Modifications to current components may provide a reduction from 3-DIMENSIONAL MATCHING. Alternatively, new components imposing orthogonality of frameworks may allow a mapping from EDGE-BICOLOURED 2-FACTOR. Perhaps the exploitation of planarity, connectedness, and orthogonality within solutions might provide assistance in designing a reduction.

These complexity questions remain unanswered and invite future examination from the complexity theoreticians.

7.6 Summary

Many avenues within polygon reconstruction from moments, including varying density, solution validation, iterative reconstruction, error tolerance within differences of moments, and problem of complexity, remain open and unexplored. These potential research domains present very tangible and genuine possibilities for improving the success of polygon reconstruction algorithms and for deepening our understanding of the nature of this problem.

Chapter 8

Conclusion

Previous work by Davis [Dav64, Dav77] and Strakhov and Brodsky [SB86], along with recent work by Milanfar *et al.* [MVKW95] and Golub *et al.* [GMV99] has inspired interest in the problems associated with the reconstruction of polygonal shapes from moments. Through their techniques, one may gather potential position and vector information about the vertices which lie on the perimeter of a polygon being reconstructed.

This preliminary acquisition process finds an approximation to the Cartesian coordinates of the vertices along with two axes of possible in-edges and out-edges for each vertex. Each axis has two possible out-edges along a line perpendicular to two possible in-edges. A reconstructed polygon must pass through every vertex exactly once visiting one out-edge and one in-edge from each axis.

Our work consists of reconstructing the polygon based on this input information. We develop and analyze reconstruction algorithms with respect to their dependence on numerical error and spatial variation, their time complexity, and their ability to recognize ambiguous solutions when more than one exists.

The main results of this thesis include:

- We have proved that some restricted 2-factor problems relating to the reconstruction problem are **NP**-hard. These include:
  - EDGE-BICOLOURED DIRECTED 2-FACTOR
  - k-CYCLE-FREE DIRECTED 2-FACTOR ($k \in \{2, 3\}$)
  - BOUNDED-DEGREE EDGE-BICOLOURED DIRECTED 2-FACTOR
  - NON-CROSSING 2-FACTOR
  - DIRECTED NON-CROSSING 2-FACTOR
- We have examined the occurrence and properties of non-unique solutions within graphs. We have proven that there exist graphs that embed an exponential number of distinct, simply-connected, non-crossing solutions with respect to the size of the input graph.
- We have formulated edge-elimination rules which can be applied to reduce a graph towards a solution subgraph. Based on these rules, we have developed a linear-time reconstruction algorithm using constraint propagation. As an extension to this algorithm, we have implemented a tree-based search routine which finds all possible solutions contained within a graph.
- We have examined techniques to find partial solutions within domains of imperfect or inaccurate input data. These include searching for segments of solutions, iterative differences of moments, and use of additional input moments.
- We have observed the behaviour of the reconstruction algorithms on various classes of graphs, both those which reconstruct successfully and unsuccessfully.

Until now, research on the polygon reconstruction problem did not address the latter phase of reconstruction, namely, the problem of taking vertex positions and potential angles, and reconstructing the final solutions. As demonstrated within this thesis, polygon reconstruction involves a multiple-phase geometric and graph-theoretic reconstruction process. Issues of non-uniqueness, ambiguity, position and angle inaccuracy, and erroneous input data must be considered when searching for solutions. Algorithms and theorems presented here address these issues and present methods at deriving complete and partial solutions.

These advances, along with other current developments, have revealed many unexplored branches within polygon reconstruction from moments, in turn providing avenues for potential future research.

Bibliography

[CLR92] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. McGraw-Hill, New York, 1992.

[Dav64] P. J. Davis. Triangle formulas in the complex plane. *Mathematics of Computation*, 18:569–577, 1964.

[Dav77] P. J. Davis. Plane regions determined by complex moments. *Journal of Approximation Theory*, 19:148–153, 1977.

[DK99] S. Durocher and D. Kirkpatrick. Resctricted 2-factor problems arising from moment-based polygon reconstruction. In *Workshop on Computational Graph Theory and Combinatorics*, pages 55–57. Pacific Institute for the Mathematical Sciences, 1999.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability.* W. H. Freeman and Company, New York, 1979.

[GMV99] G. H. Golub, P. Milanfar, and J. Varah. A stable numerical method for inverting shape from moments. *SIAM Journal of Scientific Computation,* to appear, 1999.

[HKKK88] P. Hell, D. Kirkpatrick, J. Kratochvíl, and I. Kříž. On restricted two-factors. *SIAM Journal of Discrete Mathematics*, 1(4):472–484, 1988.

[JW93] K. Jansen and G. J. Woeginger. The complexity of detecting crossingfree configurations in the plane. *BIT*, 33(4):580–595, 1993.

[LP86] L. Lovász and M. D. Plummer. Matching theory. In *Annals of Discrete Mathematics*, volume 29. North-Holland Mathematics Studies, 1986.

[MVKW95] P. Milanfar, G. C. Verghese, W. C. Karl, and A. S. Willsky. Reconstructing polygons from moments with connections to array processing. *IEEE Transactions on Signal Processing*, 43:432–443, 1995.

[O'R88] J. O'Rourke. Uniqueness of orthogonal connect-the-dots. In G. T. Toussaint, editor, *Computational Morphology*, pages 97–104. Elsevier Science Publishers B.V. (North-Holland), 1988.

[Pap94] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, New York, 1994.

[RW93] F. Rendl and G. Woeginger. Reconstructing sets of orthogonal line segments in the plane. *Discrete Mathematics*, 119:167–174, 1993.

[SB86] V. Strakhov and M. Brodsky. On the uniqueness of the inverse logarithmic potential problem. *SIAM Journal of Applied Mathematics*, 46:324–344, 1986.

[ST43] J. Shohat and J. Tamarkin. The problem of moments. In *Mathematical Surveys*, volume 1. Waverly Press, Baltimore, 1943.

Appendix A

Glossary of Terms

A.1 General Graph Theory

graph A graph, $G = (V, E)$, is a collection of vertices, V, and edges, E.

vertex A vertex, $v \in V$, is a single point or node in a graph G.

edge An edge, $e \in E$, represents a relationship between two vertices in G. We say edge $uv \in E$ if and only if there exist vertices $u, v \in V$ such that u and v are related under some relation $R \subseteq V \times V$. Graphically, we represent e by drawing a straight line from u to v.

directed graph If the edge relation is asymmetric, then G is a directed graph or **digraph**. uv and vu are distinct possible edges between vertices v and u.

subgraph Given graphs $G = (V, E)$ and $H = (V', E')$, H is a subgraph of G if and only if H is a graph, $V' \subseteq V$, and $E' \subseteq E$. If $V' = V$ then H is a **spanning** subgraph of G.

in-edge, out-edge An edge coming from a vertex u into vertex v is called an in-edge locally at v. Conversely, an edge going from vertex v out to a vertex u is described as an out-edge locally at v.

adjacency matrix We may represent vertex adjacency through a boolean adjacency matrix, M, where $M_{u,v} = true$ if and only if there exists an edge from vertex u to vertex v.

degree The degree of a vertex v is the total number of edges incident upon v. Similarly, the **in-degree** of v is the number of in-edges at v and the **out-degree** of v is the number of out-edges at v. The degree of a graph, G, is defined as the maximum degree amongst all vertices $v \in V$.

embedded graph If we are given a pre-determined fixed positioning for the vertices of a graph G, then we say that G is an embedded graph. Any fixed positioning of the vertices of G is described as an **embedding** of G.

planar If there exists a two-dimensional embedding of graph G in the plane in which none of the edges of G cross, then we say G is planar. Such and embedding is defined as a **non-crossing** embedding.

simply-connected A graph G is simply-connected if for any two vertices $u, v \in V$ we can find some sequence of vertices $\{v_1, \ldots, v_k\} \subseteq V$ such that $v_1 = u$, $v_k = v$, and for all $1 \leq i \leq k-1$, $v_i v_{i+1} \in E$ or $v_{i+1} v_i \in E$.

connected component A connected component $G' \subseteq G$ is a maximal simply-connected subgraph of G. There cannot exist vertices $v \in V - V'$ and $u \in V'$ such that $uv \in E$ or $vu \in E$.

cycle A cycle is a collection of neighbouring vertices, $\{v_1, \ldots, v_k\} \subseteq V$, such that $v_i v_{i+1} \in E$ for all $1 \leq i \leq k-1$ and $v_k v_1 \in E$. A k-**cycle** is a cycle of length k.

interior, exterior A vertex v is an interior vertex within an embedded graph G, if there exists a cycle $C \subset G$ such that $v \notin C$ and C encloses v within it. Conversely, v is an exterior vertex, if there does not exist such a cycle C.

bipartite A graph $G = (V, E)$ is **bipartite** if there exists a partition of its vertices, $V = V_1 \cup V_2$, such that, $V_1 \neq \emptyset$, $V_2 \neq \emptyset$, $V_1 \cap V_2 = \emptyset$, and every edge $v_1v_2 \in E$ has one endpoint in each partition, $v_1 \in V_1$ and $v_2 \in V_2$ or $v_1 \in V_2$ and $v_2 \in V_1$.

rectilinear All of the edges in a rectilinear embedding of a graph lie along a single orthogonal north-south, east-west axis.

A.2 Classical Complexity

P Garey and Johnson [GJ79, page 27] formally define the class **P** as follows:

$$\begin{aligned} \mathbf{P} = \{ & L : \textit{there is a polynomial time deterministic} \\ & \textit{Turing machine program } M \textit{ for which } L = L_M\} \end{aligned}$$

Informally, **P** unites all problems which have deterministic polynomial-time algorithmic solutions. That is, their solutions can be found deterministically in time proportional to some polynomial function in terms of the input size of the problem.

NP Garey and Johnson [GJ79, page 31] formally define the class **NP** as follows:

$$\begin{aligned} \mathbf{NP} = \{ & L : \textit{there is a polynomial time nondeterministic} \\ & \textit{Turing machine program } M \textit{ for which } L = L_M\} \end{aligned}$$

Informally, the class **NP** unites all problems which have nondeterministic polynomial-time algorithmic solutions.

reduction function To compare the difficulty of two decision problems, we make use of a reduction function, R, which allows us to reduce one problem to another. Papadimitriou writes:

> That is, we shall be prepared to say that problem A is at least as hard as problem B if B reduces to A. Recall what "reduces" means. We say that B reduces to A if there is a transformation R which, for every input x of B, produces an equivalent input $R(x)$ of A. Here by "equivalent" we mean that the answer to $R(x)$ considered as an input for A, "yes" or "no," is a correct answer to x, considered as an input of B. In other words, to solve B on input x we just have to compute $R(x)$ and solve A on it. [Pap94, page 159]

NP-hard If we can show a polynomial-time reduction from some problem q to a problem p, then we say p is at least as hard as q. We define a special class of problems, the class of **NP-hard** problems. If q is **NP**-hard and q is reducible to p in polynomial time, then we say p is also **NP**-hard. This special class of **NP** problems have the property that, for any **NP**-hard problem q other than SAT, there is another **NP**-hard problem p, such that, p is reducible to q in polynomial time. Thus, problem q is "at least as complex" as problem p. Therefore, **NP**-hardness is a lower bound of complexity in a hierarchy of problems for which no deterministic polynomial-time algorithmic solution is known.

A.3 Graph Theoretic and Complexity Problems

n-factor Lovász and Plummer write, "a spanning subgraph regular of degree n is called an n-factor."[LP86, page xxx] N-FACTOR is solvable in polynomial time [LP86, GJ79].

2-factor A 2-factor is an n-factor of degree 2. A **directed 2-factor** of G is a spanning subgraph $H \subseteq G$ for which every vertex $v \in V'$ has exactly one in-edge and one out-edge. The problem finding an unconstrained 2-factor in a general graph G is polynomial-time solvable [LP86, GJ79].

matching A matching is an n-factor of degree 1. Every vertex is met by exactly one edge. The problem of finding a matching in a general graph G is polynomial-time solvable [GJ79].

Hamiltonian cycle Lovász and Plummer define, "a cycle which includes every point of a graph G is called a Hamilton cycle of G."[LP86] The problem HAMILTONIAN CIRCUIT is **NP**-complete [GJ79, page xxx].

3-dimensional matching Garey and Johnson explain 3-DIMENSIONAL MATCHING as follows.

> The 3-DIMENSIONAL MATCHING problem is a generalization of the classical "marriage problem": Given n unmarried men and n unmarried women, along with a list of all male-female pairs who would be willing to marry one another, is it possible to arrange n marriages so that polygamy is avoided and everyone receives an acceptable spouse? Analogously, in the 3-DIMENSIONAL MATCHING problem, the sets W, X, and Y correspond to *three* different sexes, and each triple in M corresponds to a *3-way marriage* that would be acceptable to all three participants. Traditionalists will be pleased to note that, whereas 3DM is **NP**-complete, the ordinary marriage problem can be solved in polynomial time. [GJ79, page 50]

SAT Papadimitriou defines, "SATISFIABILITY (or SAT, for short) then is the following problem: Given a Boolean expression ϕ in conjunctive normal form, is it satisfiable?"[Pap94, page 77] SATISFIABILITY is the best-known and, historically, the first **NP**-complete problem [GJ79].

3-SAT Garey and Johnson write, "the 3-SATISFIABILITY problem is just a restricted version of SATISFIABILITY in which all instances have exactly three literals per clause."[GJ79, page 48] 3-SAT is also **NP**-complete.

2-SAT 2-SAT restricts instances to two literals per clause. This tighter restriction on the problem makes 2-SAT solvable in polynomial time [GJ79].

A.4 Polygon Reconstruction Problem

polygon reconstruction from complex moments Given a finite set of complex moments measured from a two-dimensional polygon P embedded within the plane, we attempt to reconstruct the original shape of P.

moment The moments of a region are the integrals of the powers of the independent variables over that region. The kth harmonic moment of a 2-dimensional polygon P is given by [GMV99]:

$$m_k = \int\int_P z^k \, dx \, dy$$

colour At a vertex v, we assign a single colour to each edge. This colouring is local; the colour at the head and tail of an edge may differ. We sometimes refer to **global colouring** in a graph, in which case, head and tail colours match for all edges $e \in E$. Within our domain, a vertex may only have two different colours of edges; we often refer to these locally as **red** and **blue**. For presentation of graphs, however, a graph, may have several different colours globally.

edge-bicoloured A graph $G = (V, E)$ is edge-bicoloured if, for every $v \in V$, every edge at v has local blue or red colour.

bounded-degree edge-bicoloured Given an edge-bicoloured graph $G = (V, E)$, if every $v \in V$ contains no more than two red in-edges, two red out-edges, two blue in-edges, and two blue out-edges, then G is a bounded-degree edge-bicoloured graph.

moment-derived edge-bicoloured Given a bounded-degree edge-bicoloured graph $G = (V, E)$, if edges of similar colour and direction align along an axis (see figure 2.1 in Section 2.3) then G is a moment-derived edge-bicoloured graph.

framework The angular constraints on potential edges, ϕ_1 and ϕ_2, impose an orthogonality between edges of similar colour. Thus, under a geometric interpretation of these constraints, edges of similar colour at a given vertex to belong to a common framework or **axis**. Generally, edges within an axis lie orthogonal to each other. We refer to a **global framework** whenever frameworks of common colour across all vertices in the graph lie within the same orthogonal axis.

commitment Whenever an edge is selected as being part of a partial solution, we say that we **commit** to that edge. We refer to this decision as an edge commitment.

vertex state A vertex v which is part of a solution may be in one of two states: **red-in, blue-out** (**RIBO**) or **blue-in, red-out** (**BIRO**). This state directly corresponds to edge commitments.

edge state An edge e exists in one of three states: **undecided**, **committed** or **dropped**.

valid, invalid If a vertex v has at least one in-edge of a given colour and one out-edge of a different colour, then we say v is valid. If v does not have any in-edges, does not have any out-edges or only has edges of one colour, then we say v is invalid.

red-degree, blue-degree The red-degree of a vertex v is the total number of red edges incident upon v. Similarly, the blue-degree of v is the total number of blue edges at v. If a graph G has global colouring, then we may refer to the red-degree or blue-degree of G.

happy, unhappy When a vertex v has in-degree one, out-degree one, red-degree one and blue-degree one, then we say v is happy. If v is not happy, then we say v is unhappy.

fragment A fragment is any sequence of vertices $v_1, \ldots, v_i$ such that $v_k v_{k+1} \in E$, for all $1 \leq k \leq i-1$ and and every vertex $v_2, \ldots, v_{i-1}$ is happy.

solution If $H = (V', E')$ is a spanning subgraph of G which has the property that every $v \in V'$ is happy, then H is a solution subgraph of G. We describe the state of H as **global happiness**.

ambiguous solution Several graphs embody more than one possible solution. Any such non-unique solution is an ambiguous solution.

constraint Vertex happiness requires a single pair of edges: red in and blue out, or, blue in and red out. We refer to these restrictions as **colour constraints**.

constraint propagation Constraint propagation occurs whenever the constraints at a vertex v affect the constraints at a neighbouring vertex u.

reduction If graph $H_2 \subseteq H_1$ is a spanning subgraph of H_1 and all edges $E \subseteq H_1 - H_2$ may be removed from H_1 by constraint propagation, then we say H_2 is a **reduced** graph of H_1. If no further edges may be removed from H_2 by constraint propagation, then H_2 is **maximally reduced**.

A.5 Reconstruction Algorithm Modules

recons Module `recons` takes k moment measurements and produces a set of n vertex positions and two sets of n angles each, for local red and blue edge axes at each respective vertex. Usually, $k = 2n$.

buildGraph Module `buildGraph` takes n vertex positions along with blue and red axis angles for each and builds a dense edge-bicoloured graph G_1 which includes edges for any matching combination of in-edge and out-edge angles (Section 5.2).

edgeElim Module `edgeElim` takes G_1 as input and produces a regular edge-bicoloured spanning subgraph G_2 such that every vertex has at most two edges of similar colour and direction which are geometrically opposite to each other (Section 5.3).

commit Module `commit` works recursively on G_2 using colour constraint propagation to produce a solution subgraph G_3 for which no further constraint propagation is possible (Section 5.4).

treeSearch Module `treeSearch` works recursively down a search tree towards solutions whenever constraint propagation alone within module `commit` fails to reduce G_3 to a final solution (Section 5.5).

fragmentSearch Module `fragmentSearch` is used whenever the input domain includes imperfect data. Partial fragmented solutions are researched as opposed to complete 2-factors (Section 5.6).

momentDiff Module `momentDiff` allows for extremely noisy data to be reconstructed piecewise through re-iteration of the `recons` module on differences of moments (Section 5.7).