CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

PROTEIN FOLDING: PLANAR CONFIGURATION SPACES OF DISC ARRANGEMENTS AND HINGED POLYGONS

A thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science in Applied Mathematics

by

Clinton Bowen

August 2014

The thesis of Clinton Bowen is approved:

Dr. Silvia Fernandez — Date

Dr. John Dye — Date

Dr. Csaba Tóth, Chair — Date

California State University, Northridge

Table of Contents

ABSTRACT

PROTEIN FOLDING: PLANAR CONFIGURATION SPACES OF DISC ARRANGEMENTS AND HINGED POLYGONS

By

Clinton Bowen

Master of Science in Applied Mathematics

Chapter 1

Background

In this section, we cover the background subjects needed to formally pose the problems and present solutions in this thesis. We start with two types of combinatorial structures, linkages and polygonal linkages. We then discuss the configuration spaces of linkages and polygonal linkages. We then look into an alternate representation of linkages, disk arrangements and state the disk arrangement theorem. Next we look at satisfiability problems and then review a framework, the logic engine, which can encode a type of satisfiability problem. Finally, we cover the basic definitions of algorithm complexity for **P** and **NP**.

Decidability problems study whether there exists a way to determine whether an element is a member of a set. In this paper we focus on four such decidability problems surrounding graph theory and geometry. The first set of problems involve a special type of graph called a tree and the second set of problems involve something called a polygonal linkage. In each problem, set membership is determined if the tree or polygonal linkage has a particular property when visualized in the plane.

This thesis first presents the preliminary information needed to pose our four problems, then we formally pose each problem and then provide solutions on decidability for each problem.

1.1 Graphs

A *graph* is an ordered pair $G = (V, E)$ comprising of a set of vertices V and a multiset of edges E. An edge is a 2 element subset of V and denoted as "V choose two", $\begin{pmatrix} V \\ 2 \end{pmatrix}$. Vertices are said to be *adjacent* if they form an edge in E. *Neighbors* of vertex v are the adjecent vertices of v. Vertices that are adjacent to themselves are *self-adjacent*, i.e. $u = v$ for $\{u, v\} \in E$. Edges are said to be *incident* if they share a vertex. When an edge is a member of E multiple times, then we say that the graph is a *multigraph*. A *simple graph* has no self-adjacent vertices. If $G' = (V', E')$ such that $V' \subset V$ and $E' \subset E$, then G' is a *subgraph* of G. Unless otherwise stated, this thesis will strictly work with simple graphs.

For graph equivalency, we need to define an isomorphism for graphs. Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, a *graph isomorphism* a bijective function $f : V_1 \mapsto V_2$ such that for any two vertices $u, v \in V_1$, we have $\{u, v\} \in E_1$, if and only if $(f(u), f(v)) \in E_2$.

)

Graph	Vertices	Edges
G_1	$\{a,b,c,d,e\}$	$\{(a,b),(b,c),(c,d),(d,e),(e,a)\}$
G_2	$\{1,2,3,4,5\}$	$\{(1,2),(2,3),(3,4),(4,5),(5,1)\}$

Table 1.1: Two graphs that are isomorphic with the alphabetical isomorphism $f(a) = 1$, $f(b) = 2$, $f(c) = 3$, $f(d) = 4$, $f(e) = 5$.

To visualize a graph, G, we create a *drawing* Γ ,of G. For a drawing, we use an injective mapping $\Pi : V \mapsto \mathbb{R}^2$ which maps vertices to distinct points in the plane and for each edge $\{u, v\} \in E$, a continuous, injective mapping $c_{u,v} : [0, 1] \mapsto \mathbb{R}^2$ such that $c_{u,v}(0) = \Pi(u)$ and $c_{u,v}(1) = \Pi(v)$. The *crossing number* of a graph is the smallest number of edge crossings for a graph, i.e. A drawing is said to be *planar* if no two

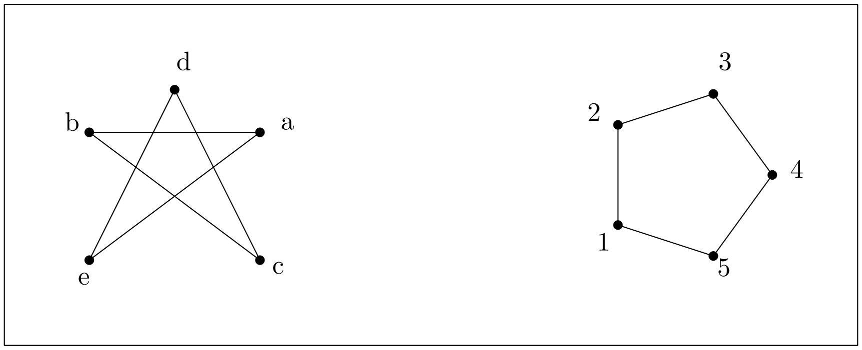


Figure 1.1: This figure depicts the graph isomorphism shown in Table (**??**) between V_1 and V_2 in the plane.

distinct edges.

A *crossing* is when two edges intersect. The crossing number $cr(\Gamma_G)$ of a drawing of G is the number of crossings in the drawing of the graph G. We can treat $cr()$ as a mapping from the space of drawings of G to the non-negative integers, i.e. $cr : \{\Gamma_G\} \mapsto \mathbb{Z}^+ \cup \{0\}$. One can ask if for any graph G, does there exists a drawing in which the crossing number is zero? Without loss of generality, this is a type of crossing number minimization problem, i.e.

$$\min_{\gamma \in \{\Gamma_G\}} cr(\gamma)$$

Kuratowski's theorem allows one to characterize finite planar graphs, i.e. a finite graph is planar if and only if it does not contain a subgraph that is a subdivision of K_5 or $K_{3,3}$ [8]. If no two distinct edges intersect

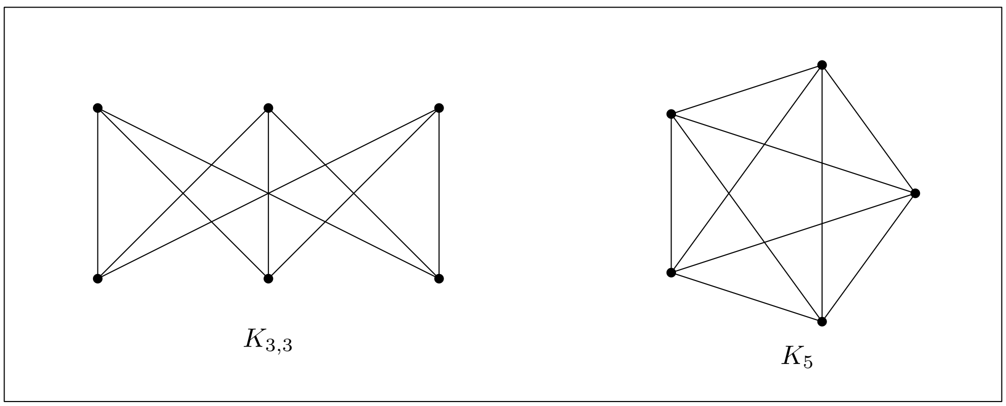


Figure 1.2: The K_5 and $K_{3,3}$ drawn in the plane.

in the drawing, the drawing is said to be *planar*. Two planar drawings of a graph G are equivalent if they determine the same circular orderings of the neighbor sets. A *planar embedding* is an equivalence class of planar drawings and is described by the circular order of the neighrs of each vertex. An equivalence class of a planar drawing ϕ is any combination of rigid transformations: rotation, translation, and reflection.

1.1.1 Trees

Some graphs can be classified by which properties they may have or not have. A *path* is a sequence of vertices in which every two consecutive vertices are connected by an edge. A *simple cycle* of a graph is a

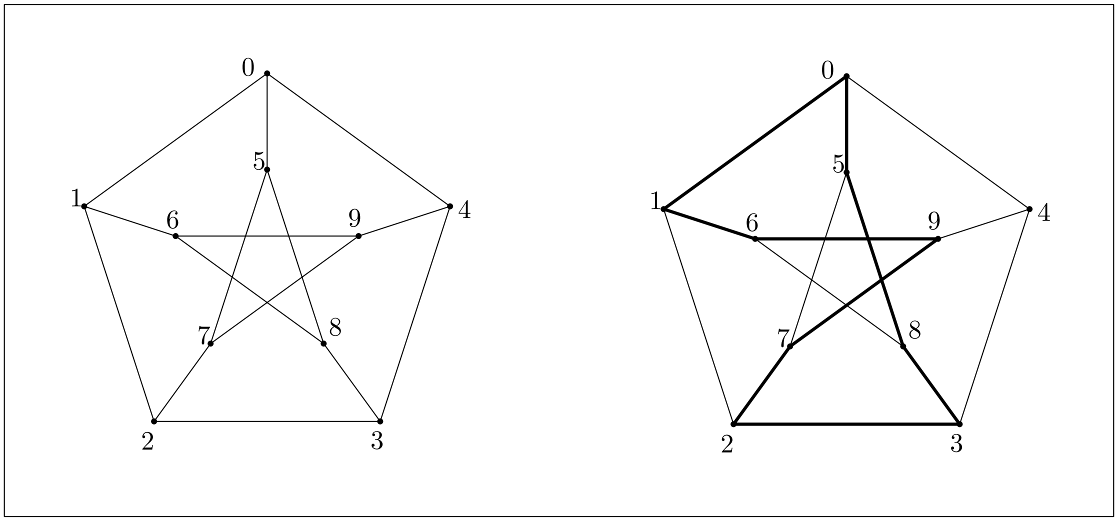


Figure 1.3: An embedding of the Peterson graph with a simple cycle of (2,7,9,6,1,0,5,8,3).

sequence, $(v_1, v_2, \ldots, v_{t-1}, v_t)$, of distinct vertices such that every two consecutive vertices are connected by an edge, and the last vertex, v_t, connects to v_1. A graph is *connected* if for any two vertices, there exists a path between the two points. A *tree* is a graph that has no simple cycles and is connected.

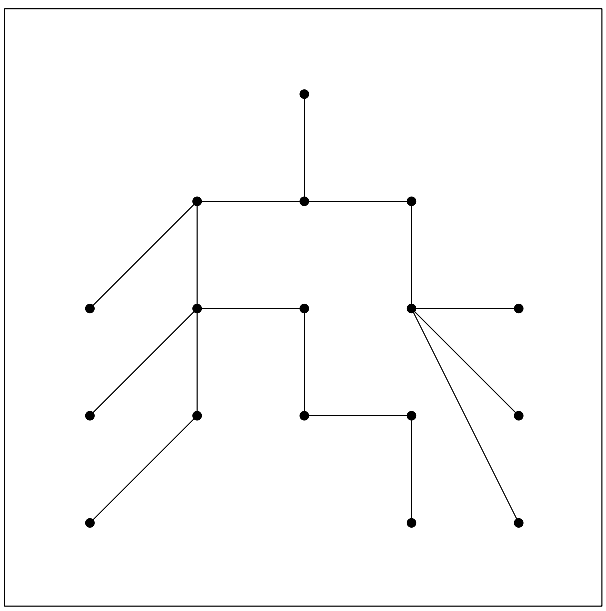


Figure 1.4: An example of a tree.

An *ordered tree* is a tree T together with a cyclic order of the neighbors for each vertex, O. Embeddings of ordered trees are equivalent if for each node the counter-clockwise ordering of adjacent nodes are the same.

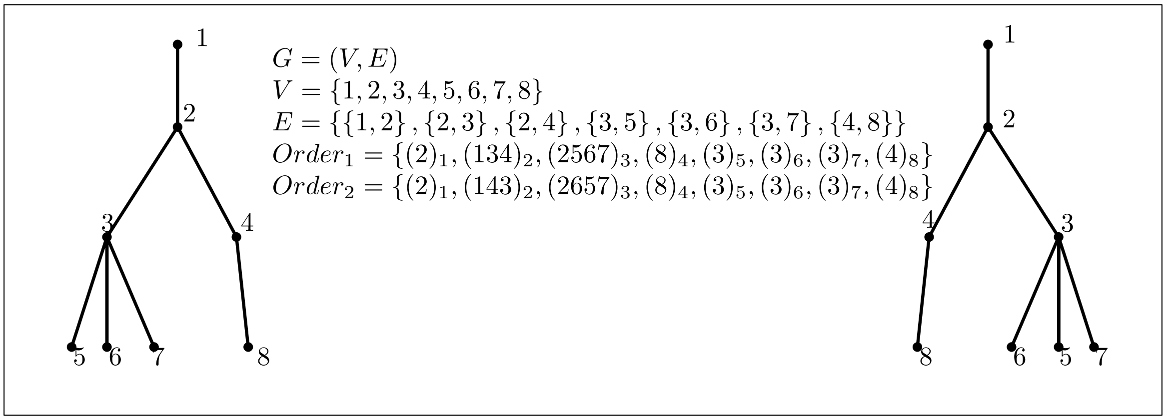


Figure 1.5: A tree with two embeddings with different cyclic orderings around vertices.

For embeddings of ordered trees, an equivalence class for an embedding ϕ is any combination of rotations and translations. Unlike embeddings for planar graphs where ordering of adjacent vertices is not a distinguishing condition, we do not consider reflection based transformations for embeddings of ordered trees for equivaliency as that can modify the ordering of adjacent vertices.

1.2 Linkages

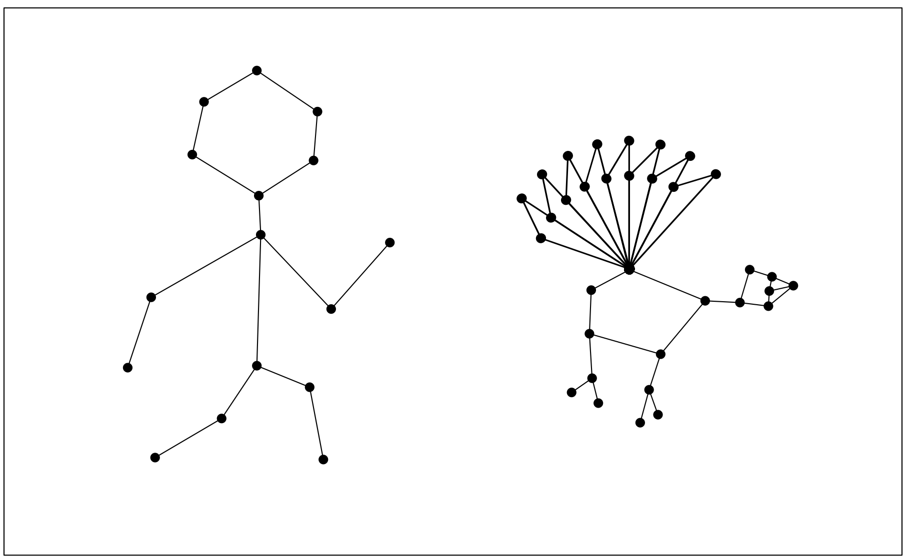


Figure 1.6: Here are skeleton drawings of a human and a turkey. When animating skeletons, one tends to make sure that the lengths of the skeleton segments are kept the same length throught the animation. Otherwise, the animation may depart from what is ideally understood of skeletal motions.

When graph drawings model physical objects, other qualities avout the graph can be contextualized in a geometric sense. Distance, angular relationstips and other geometric qualities of the drawings can be other useful properties of the drawing to perform analysis on. The *length assignment* of a graph $G = (V,E)$ is $\ell : E \mapsto \mathbb{R}^+$. For simple graphs, length assignment must be strictly positive, otherwise it may result in two distinct vertices with the same coordinates. A *linkage* is a graph $G = (V,E)$ with a length assignment $\ell : E \mapsto \mathbb{R}^+$. Length assignments can be thought of as a metric where $\ell(u,v) = \ell(v,u) > 0$.

Consider embeddings of a graph that respects the length assignment. A *realization* of a linkage, (G,ℓ), is an embedding of a graph, Π, such that for every edge $\{u,v\} \in E$, $\ell(\{u,v\}) = |\Pi(u) - \Pi(v)|$. A *plane realization* is a plane embedding with the property, $\ell(\{u,v\}) = |\Pi(u) - \Pi(v)|$. First let's define the space of realizations for a corresponding linkage, i.e.:

$$P_{(G,\ell)} = \left\{ \Pi_{(G,\ell)} \,|\forall\{u,v\} \in E,\, \ell(\{u,v\}) = |\Pi(u) - \Pi(v)| \right\}$$

With respect to P, we can establish the a *configuration space* that allows one to study problems of motion. For each vertex of G, the embedding of the vertex lies in the plane, i.e. $\Pi(v) \in \mathbb{R}^2$. By enumerating each vertex of G, e.g. $v_1, v_2, \ldots, v_k, \ldots, v_n$, we can create a projection mapping from $\mu : P \mapsto \mathbb{R}^{2|V|}$ where the corresponding coordinates of $\Pi(v_k)$ are in the $(2k)^{\text{th}}$ and $(2k+1)^{th}$ coordinates in $\mathbb{R}^{2|V|}$. The configuration space is $\mu(P)$.

By using real analysis, we can begin to pose problems about linkages with respect to a corresponding configuration space. We define a path $\gamma : [0,1] \mapsto \mu(P)$ where $\gamma(0)$ corresponds the the projection of a realization of a linkage Π_0 and $\gamma(1)$ corresponds to another realization of a linkage Π_1. If for any two elements $a, b \in \mu(P)$ that there exists a continuous path γ such that $\gamma(0) = a$ and $\gamma(1) = b$, $\mu(P)$ is said to be path connected. For γ to be continuous we would have that for every $\varepsilon > 0$, there exists a $\delta > 0$ such that if $x, y \in [0,1]$ and $|x-y|\delta$ then $||\gamma(x) - \gamma(y)|| < \varepsilon$. γ can be thought of as an animation of drawings that starts at $\gamma(0)$ and ends at $\gamma(1)$ To ask if $\mu(P)$ is a connected space, is to ask if $\mu(P)$ is connected in $\mathbb{R}^{2|V|}$.

1.2.1 Configuration Spaces of Linkages

NOTE THAT THIS SUBSECTION MAY HAVE REPEATED CONTENT

Let's focus on the space of embeddings of a linkage. If there are n vertices of a linkage, the *configuration space* of a linkage is said to be a vector space of dimension $2 \cdot n$ where edge length is preserved. A *configuration space* for a linkage G and corresponding proper embedding, L_1 is said to be for any other

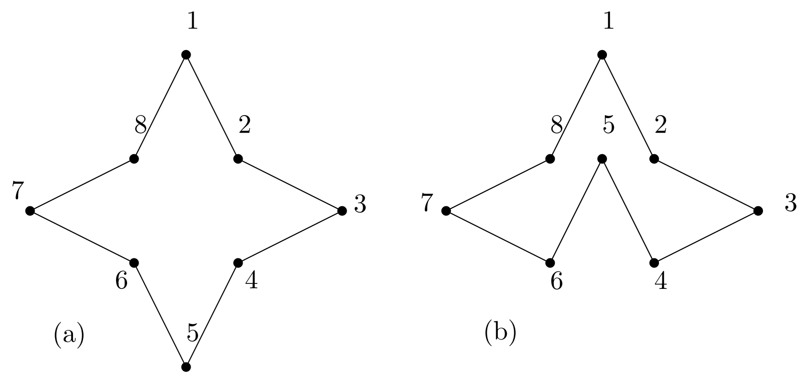


Figure 1.7: (a) and (b) show a linkage in two embeddings.

proper embedding of a linkage G, L_2, such that the lengths of every edge of G is preserved between the two embeddings, i.e.:

$$l((u,v)) = |L_1(u) - L_1(v)| = |L_2(u) - L_2(v)|$$

Equivalent embeddings include translations and rotations about the center of mass on $L(V)$. We further our embeddings by requiring that one vertice is pinned to the point of origin on the plane as well as a neighboring vertex.

Theorem 1 ([4, 10]Carpenter's Rule Theorem)**.** *Every realization of a linkage can be continuously moved (without self-intersection) to any other realization. In other words, the realization space of such a linkage is always connected.*

A *reconfiguration* of a linkage whose graph is $G = (V,E)$ and length assignment is ℓ is a continuous func-

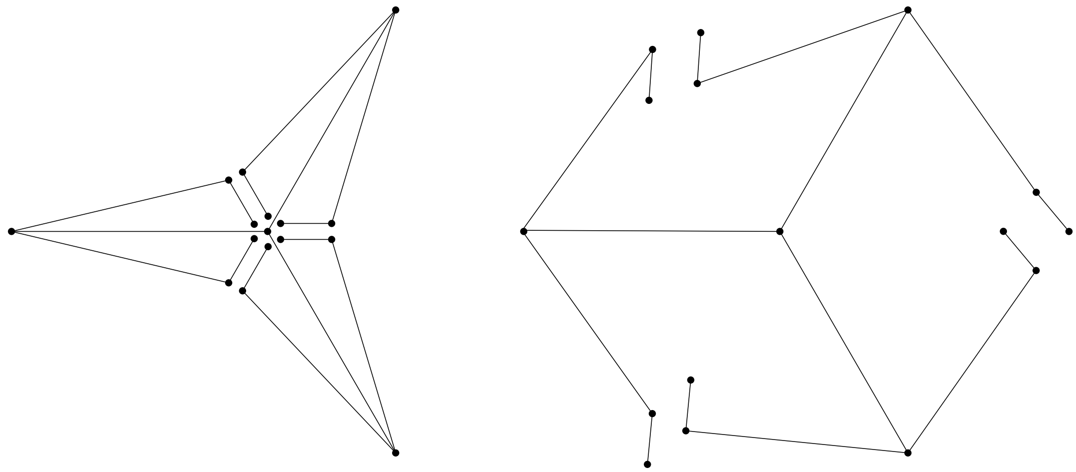


Figure 1.8: A linkage whose complete configuration space is discontinuous. These two examples above are two configurations of the same linkage that cannot continuously transform into the other without edge crossing.

tion $f : [0,1] \mapsto \mathbb{R}^{2\cdot|V|}$ specifying a configuration of the linkage for every $t \in [0,1]$ where length assignment ℓ is preserved, edges do not cross and for every $\varepsilon > 0$, there exists a $\delta > 0$ such that $|t_1 - t_2| < \delta$ implies

$$|f(t_1) - f(t_2)| < \varepsilon$$

1.3 Polygonal Linkages

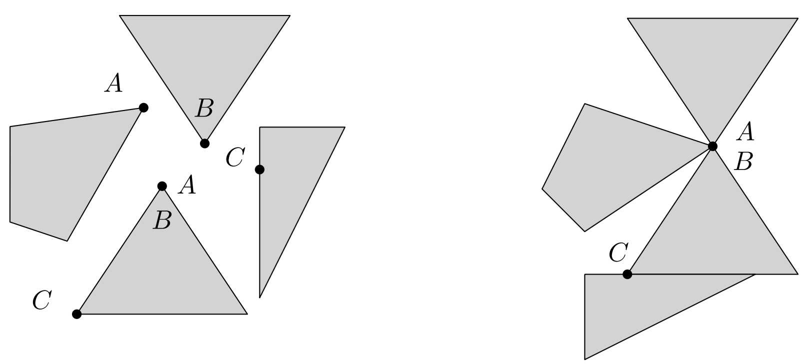


Figure 1.9: (a) A polygonal linkage with a non-convex polygon and two hinge points corresponding to three polygons. Note that hinge points correspond to two distinct polygons.(b) Illustrating that two hinge points can correspond to the same boundary point of a polygon.

A generalization of linkages are polygonal linkages where the edges of given lengths are replaced by rigid polygons. Formally, a *polygonal linkage* is an ordered pair $(\mathsf{PP}, \mathscr{H})$ where PP is a finite set of polygons and $\mathscr{H}$ is a finite set of hinges; a *hinge* $h \in \mathscr{H}$ corresponds to two points on the boundary of two distinct polygons in PP. A *realization* of a polygonal linkage is an interior-disjoint placement of congruent copies of the polygons in PP such that the points corresponding to each hinge are identified (Fig. **??**). A **realization with orientation** uses only translated or rotated copies of the polygons in PP (no reflections) and for each hinge, the cyclic order of incident polygons is given. The topology of a polygonal linkage can be represented by the **hinge graph**, a bipartite graph where the vertices correspond to polygons in PP and the hinges in H, and edges represent the polygon-hinge incidences. This definition of realization rules well known geometric dissections (e.g. Fig. 1.10).

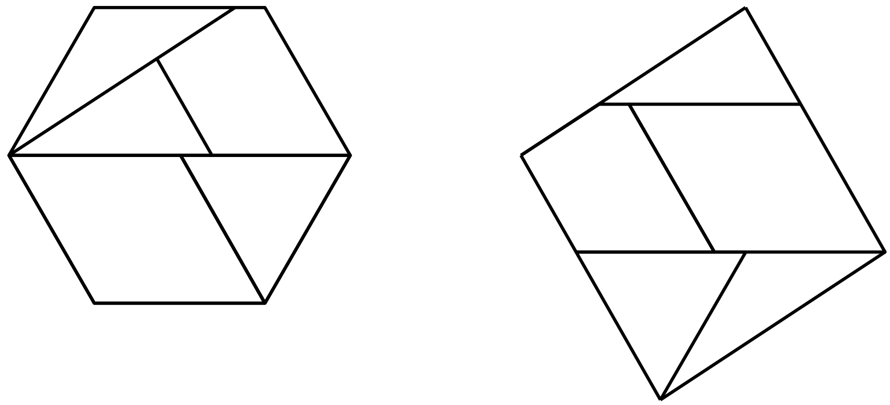


Figure 1.10: Two configurations of polygonal linkage where the polygons touch on boundary segments instead of hinges. These two realizations of the polygonal linkage are invalid to our definitions.

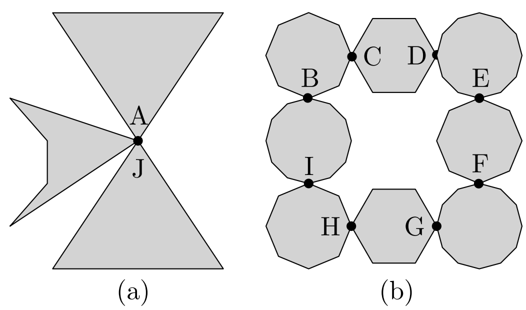


Figure 1.11: (a) A polygonal linkage with a non-convex polygon and a hinge point corresponding to three polygons. (b) A polygonal linkage with 8 regular polygons.

1.4 Disk Arrangements

It turns out the disk arrangements are an equivalent way to to represent plane graphs. By representing vertices as interioir disjoint disks and by representing edges as as points of intersections (contact), *kissing points* between two disks. The graph corresponding to a given disk arrangement, $\mathscr{D}$, is said to be the *contact graph*. A *disk arrangement* is a set, $\mathscr{D}$, of pairwise interior-disjoint disks in the plane, $\mathscr{D} = \{C_i\}_{i=1}^n$. $\{C_i\}_{i=1}^n$ such that for any circle $C \in \{C_i\}_{i=1}^n$, C

A classical result by Thurston and Koebe is that every disk arrangement embedded into the plane had a corresponding plane graph.

Theorem 2 (**??**Disk Packing Theorem)**.** *For every graph G, there is a disk arrangement in the plane whose contact graph is isomorphic to G.*

Proposition 1. *For every linkage L, there is a disk arrangement in the plane whose contact graph is isomorphic to L.*

1. Show the relation between polygonal linkages and disk arrangengements.

Suppose we're given a tree. By the disk packing theorem we can ascertain a sense of order for the isomorphic

Figure 1.12: This example represents a disk arrangement transformed to and from its corresponding graph G_2

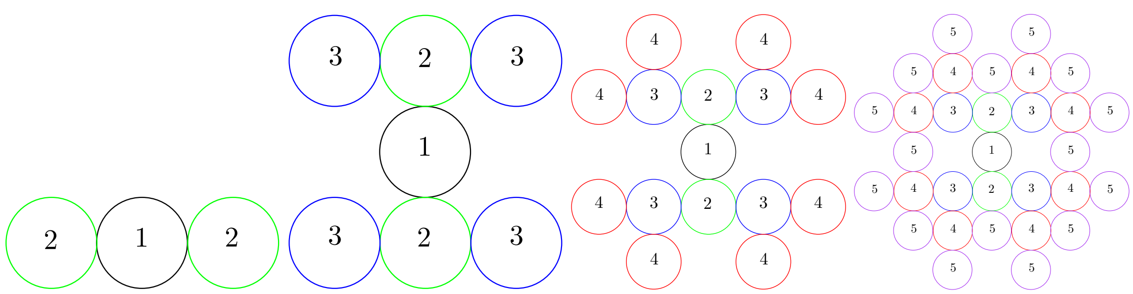
disk packing. An *ordered disk arrangement* is a rooted tree in which the counter-clockwise ordering of adjacent vertices.

Given inputs of radii By adding constraints to the embeddings of disk arrangements, we can devise realizability problem by a volume argument.

1. Round 1: Start with a disk of unit radius.
2. Round 2: Add two kissing disks, each of diameter 2, that do not intersect with any other disk (they may kiss other disk).
3. Round 3 and Higher: For each new kissing disk added, add two more non-intersecting kissing disks of diameter 2 to it.

For each round i we are adding $2^{(i-1)}$ disks, each with an of π. The area that the disk arrangement is bounded by at round i is a box of length $2 \cdot (2 \cdot (i-1)+1)$ totalling to an area of $(4 \cdot i^2 - 4 \cdot i + 1)$. Meanwhile the total area of the disk arrangement at round i is $\pi \cdot (2^i - 1)$. The exponential growth rate of the disk packing will exceed its bounded area for sufficiently large i.

Figure (1.13) illustrates the iterative problem. The problem with this is that the area in which is necessary to contain this disk growing disk arrangement will exceed the area needed to contain it.



(a) A disk arrangement with two layers of disks
(b) A disk arrangement with three layers of disks
(c) A disk arrangement with four layers of disks
(d) A disk arrangement with five layers of disks

Figure 1.13: The gradual growth of disk arrangements by adding two kissing disks to each of the previously generated disks. By continuing this arrangement growth, the space needed to contain the kissing disks will exceed the area containing the disk arrangements.

1.5 Configuration Spaces

Just as one can compose colors or forms, so one can compose motions.

Alexander Calder, 1933

We'd like to describe motions and range of motions of embedded graphs, linkages, polygonal linkages, and disk arrangements. Table **??** provides the definition of *reconfiguration* for each type of object covered so far:

Object Type	Definition of Reconfiguration
Graph Embeddings	a continuous motion of the vertices that never causes the edges to intersect.
Linkage	a continuous motion of the vertices that preserves the lengths of the edges and never causes the edges to intersect.
Polygonal Linkage	a continuous motion of polygons that preserves shapes of polygons, hinge point pairings, and never causes the polygonal sides to intersect.
Disk Arrangement	a continuous motion of disks that preserves disk radii, pairs of contact points, and never causes disks to intersect.

1.5.1 Configuration Spaces of Polygonal Linkages

Some text goes here

1.5.2 Configuration Spaces of Disk Arrangements

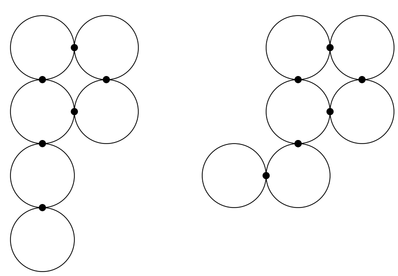


Figure 1.14: A linkage whose complete configuration space is discontinuous. These two examples above are two configurations of the same linkage that cannot continuously transform into the other without edge crossing.

1.6 Algorithm Complexity

Algorithms are a set of procedural calculations. When an algorithm executes its procedure it can be measured in terms of units of consumed resources (in computers, that is memory) and the time it takes to complete the procedure of calculations. Ideally, a desirable algorithm would run quickly and utilizes a small amount of resources.

Determining the time and space that algorithms use determine their efficiency. The *worst-case* running time is the largest possible running time that an algorithm could have over all inputs of a given size N. *Brute force* is when an algorithm tries all possibilities to see if any formulates a solution. An algorithm is said to be *efficient* if it achieves qualitatively better worst-case performance, at an analytical level, than brute force search.

For combinatorial problems, as the number of inputs of the problem grows, the solution space tends to grow exponentially. In general, as problems grow, it is desirable to minimize the *running time*, time take to run an algorithm that solves a problem. Formally, we quantify running time with Big O notation.

Definition 1 (Big O Notation)**.** Let f and g be defined on some subset of $\mathbb{R}$. $f(x) = O(g(x))$ if and only if there exists a positive real number M and x_0 such that

$$|g(x)| \leq M\,|f(x)|$$

for all $x \geq x_0$

An algorithm has a *polynomial running time* if there is a polynomial function p such that for every input string s, the algorithm terminantes on s in at most $O(p(|s|))$ steps.

To categorize problems [7], we ask the following:

Problem 1. Can arbitrary instances of problem Y by solved using a polynomial number of standard computational steps, plus a polynomial number of calls to an algorithm that solves X?

The class of problems that can be solved in polynomial running time is called the *polynomial time* class, P. A second property of problems is whether if its solution can be verified efficiently. This property is independent of whether it can be solved efficiently. B is said to be an efficient certifier for a problem X if the following properties hold:

(i) B is a polynomial-time algorithm that takes two inputs s and t.

(ii) There exists a polynomial function p such that for every string s, we have $s \in X$ if and only if there exists a string t such that $|t| \leq p(|s|)$ and $B(s,t) =$ 'yes'.

The class of problems which have an efficient certifier is said to be the *nondeterministic polynomial time* class, NP. Before we continue with the definitions for NP and NP complete, we will look into a type of problem, a reduction of a problem, and what an efficient certification is. This facilitates the reader for the definitions and illustrate complexity better.

A *polynomial time reduction* is when arbitrary instances of problem Y be solved using a polynomial number of standard computational steps, plus a polynomial number of calls to a black box that solves problem X. subsubsectionIndependent Sets and Vertex Covers To illustrate what a reduction is, we cover an example of independent sets and vertex covers. Given a graph $G = (V,E)$, a set of vertices $S \subset V$ is *independent* if no two vertices in S are joined by an edge. A *vertex cover* of a graph $G = (V,E)$ is a set of vertices $S \subset V$ if every edge $e \in E$, has at least one end corresponding in S.

Theorem 3. *Let $G = (V,E)$ be a graph. Then S is an independent set if and only if its complement $V - S$ is a vertex cover.*

Proof 1. *If S is an independent set. Then for any pair of vertices in S, the pair are not joined by an edge if and only if for any $v_1, v_2 \in S$, $e = (v_1, v_2) \notin E$. We have two cases. The first case is if $v \in S$, then any vertex $u \in V$ that forms an edge $e = (v,u) \in E$ must reside in $V - S$. The second case is if there is an edge which no pair of vertices is in S, then both vertices are in $V - S$. Both cases together imply that every edge has at least one end corresponding in $V - S$.*

If $V - S$ is a vertex cover. Every edge $e \in E$ has at least one vertex in $V - S$. The two possible cases, the first case is that the second vertex is in $V - S$, and the second case is that the second vertex is in S. The first *case would yield $S = \emptyset$. The second case implies that the edge $e \in E$ has exactly one vertex in $V - S$ and exactly one vertex in S. $V - S$ is a vertex cover would disallow S to have a pair of vertices to form an edge in the graph.*

Theorem 3 allows for problem reductions for independent set and vertex cover problems.

There are two problems for the independent set: an optimization problem and a decision problem.

Problem 2 (Optimization of an Independent Set in G). Given a graph G, what is the largest independent set in G?

Problem 3 (Decision of an Independent Set of Size k). Given a graph G and a number k, does G contain an independent set of size at least k?

[NEEDS TO PROVE REDUCIBILITY IN THIS PARAGRAPH BELOW] An algorithm that solves the optimization problem automatically solves the decision problem of the independent set. An algorithm that solves the decision problem for all size k solves the optimization problem where the decision is "yes" for the largest value of k. This establishes a reduction of the optimization problem to the decision problem and vice versa. [NEEDS TO PROVE REDUCIBILITY IN THIS PARAGRAPH ABOVE]

1.7 Satisfiability

Let $x_1, \ldots, x_n$ be boolean variables. A boolean formula is a combination of conjunction, disjunctions, and negations of the boolean variables $x_1, \ldots, x_n$. A boolean formula is *satisfiable* if one can assign true or false value to each variable so that the formula is true.

Problem 4 (Satisfiability Problem (SAT)). Given a boolean formula, decide whether it is satisfiable.

[9] It is known that every boolean formula can be rewritten in *conjunctive normal form* (CNF), a conjunction of clauses. A *clause* is a disjunction of distinct literals. A *literal* is a variable or a negated variable, x_i or $\bar{x}_i$, for $i = 1, \ldots, n$. Furthermore, it is also known that every boolean formula can be written in CNF such that each clause has exactly three literals. This form is called 3-CNF. Given a boolean formula in 3-CNF, decide whether it is satisfiable is a *3-SAT problem*.

The problems we focus on in this thesis have a geometry. A special geometric 3-SAT problem is that Planar 3-SAT Problem. Given a 3-CNF boolean formula, B, we define the associated graph as follows: the vertices correspond to the variables and clauses in B, when a variable or its negation appears in a clause there is an edge between the corresponding vertices.

Problem 5 (Planar 3-SAT). Given a boolean formula B in 3-CNF such that its associated graph is planar, decide whether it is satisfiable is a *3-SAT problem*.

Problem 6 (Not All Equal 3 SAT Problem (NAE3SAT)). Given a boolean formula in 3-CNF, decide whether it is satisfiable so that each clause contains a true and a false literal.

1.8 Problem

The *realizability* problem for a polygonal linkage asks whether a given polygonal linkage has a realization (resp., orientated realization). For a weighted planar (resp., plane) graph,, it asks whether the graph is the contact graph (resp., ordered contact graph) of some disk arrangement with specified radii. These problems, in general, are known to be NP-hard. Specifically, it is NP-hard to decide whether a given planar (or plane) graph can be embedded in $\mathbb{R}^2$ with given edge lengths [3, 5]. Since an edge of given length can be modeled by a suitably long and skinny rhombus, the realizability of polygonal linkages is also NP-hard. The recognition of the contact graphs of unit disks in the plane (a.k.a. coin graphs) is NP-hard [2], and so the realizability of weighted graphs as contact graphs of disks is also NP-hard. However, previous reductions crucially rely on configurations with high genus: the planar graphs in [3, 5] and the coin graphs in [2] have many cycles.

In this thesis, we consider the above four realizability problems when the union of the polygons (resp., disks) in the desired configuration is simply connected (i.e., contractible). That is, the contact graph of the disks is a tree, or the "hinge graph" of the polygonal linkage is a tree (the vertices in the *hinge graph* are the polygons in PP, and edges represent a hinge between two polygons). Our main result is that realizability remains NP-hard when restricted to simply connected structures.

Problem 7 (Unordered Realizibility Problem for the Tree). For a tree with positive weights for the verticies, it asks whether it is a contact graph of some disk arrangement where the radii are equal to the vertex weights.

Problem 8 (Unordered Realizibility Problem for Polygonal Linkages). The *realizability* problem for a polygonal linkage asks whether a given polygonal linkage has a realization.

Problem 9 (Ordered Realizibility Problem for the Tree). For a tree with positive weights for the vertices, it asks whether its corresponding graph is the ordered contact graph of some disk arrangement where the radii equal the vertex weights.

Problem 10 (Ordered Realizibility Problem for Polygonal Linkages). labelproblem:OrderedPolygonal The *realizability* problem for a ordered polygonal linkage asks whether a given polygonal linkage has a realization with respect to order.

Theorem 4. *It is NP-Hard to decide whether a polygonal linkage whose hinge graph is a tree can be realized (both with and without orientation).*

Theorem 5. *It is NP-Hard to decide whether a given tree (resp., plane tree) with positive vertex weights is the contact graph (resp., ordered contact graph) of a disk arrangements with specified radii.*

The unoriented versions, where the underlying graph (hinge graph or contact graph) is a tree can easily be handled with the logic engine method (Section **??**). We prove Theorem 4 for *oriented* realizations with a reduction from PLANAR-3SAT (Section **??**), and then reduce the realizability of ordered contact trees to the oriented realization of polygonal linkages by simulating polygons with arrangements of disks (Section **??**).

Chapter 2

Decidability Problems for Hinged Polygons and Disks

2.1 The Logic Engine

The *logic engine* is a planar, mechanical device that simulates an instance NAE3SAT problem. It was introduced in [1].

2.1.1 Construction of the Logic Engine

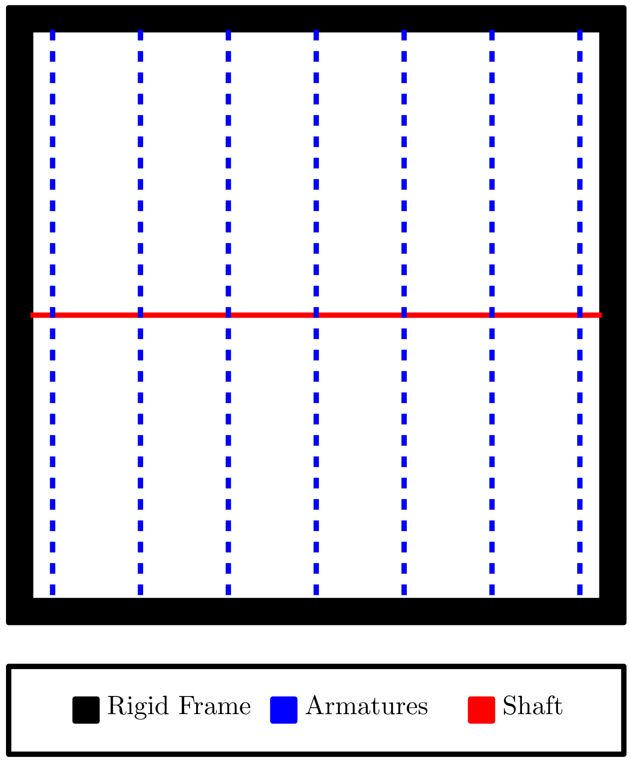


Figure 2.1: A logic engine frame with veritical armatures and a horizontal shaft.

For a given a boolean formula, Φ, in 3-CNF with n variables and m clauses, we construct a logic engine. The logic engine has a *rigid frame* which houses the mechanical components of the the logic engine. The rigid frame is the boundary of which the logic engine can operate within. The *shaft* is a horizontal line segment that is placed at mid-height of the rigid frame. The *armatures* are veritical line segments whose midpoints are on the shaft. Each armature has two orientations with respect to the shaft.

Each armature corresponds to a variable in Φ. There are two literals for each variable, i.e. the negated literal $\bar{x}_j$ and non-negated literal x_j. Flagging arragement indicates the relationship of the boolean literal's existence within a clause. Partition each armature into $2m$ unit, vertical line segments. Label the segments on the j^{th} armature starting from the shaft by $\ell_{j,1}, \ldots, \ell_{j,n}$ on one side and $\bar{\ell}_{j,1}, \ldots, \bar{\ell}_{j,n}$ on the other side of the shaft. Attach regular triangles, called *flags*, to some of these segments. Each segment is either flagged one or zero flags, i.e. *flagged* or *unflagged*.

1. If the literal x_j is found in clause C_k, then $\ell_{j,k}$ is unflagged.

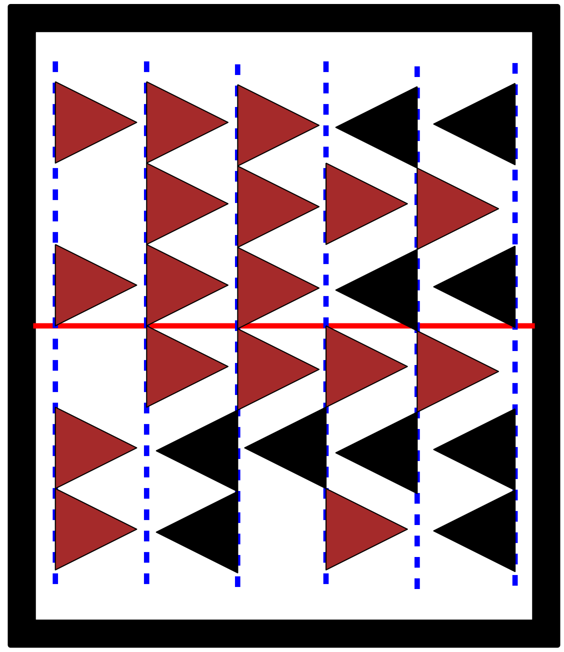
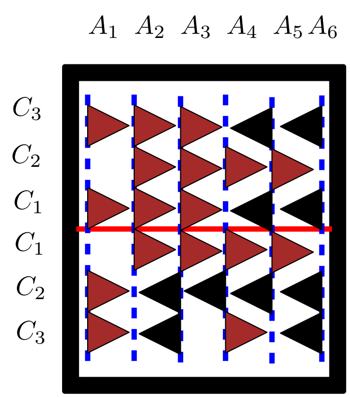


Figure 2.2: A logic engine that corresponds to a boolean formula in NAE3SAT form, Φ. The picture shows the outer rigid frame, the shaft, the armatures that correspond to the variables in Φ, with oriented flags.

2. If the literal $\bar{x}_j$ is found in clause C_k, then $\bar{l}_{j,k}$ is unflagged.

Each flag has two orientations with respect to armature it is attached to. Each flag has four potential positions, the flag can reflect left or right about the armature and the armature can reflect up or down about the shaft. The *flags* are equilateral triangles attached to the armatures. The placement of the flags is dependent



$$C_3 = (x_4 \cup \bar{x}_2 \cup \bar{x}_5)$$
$$C_2 = (x_1 \cup x_6 \cup \bar{x}_2)$$
$$C_1 = (\bar{x}_1 \cup \bar{x}_5 \cup x_4)$$

Figure 2.3: A logic engine constructed from the boolean formula $\Phi = C_1 \cap C_2 \cap C_3$.

on the instance of the NAE3SAT boolean formula. Each flag as two orientations.

For the NP hardness reduction in Theorem 7 we need to make sure that all parameters in the logic engine can be specified polynomially in terms of the size of the boolean formula. Given Φ, the corresponding logic engine is constructed as follows: all components will be specified with a quantity and coordinates defined as polynomials in m and n.

Component	Quantity	Set Definition
Rigid Frame	1	$\left\{ (x,y) \in \mathbb{R}^2 \,\middle|\, \text{The boundary of } \left[\frac{1}{2}, n+\frac{1}{2}\right] \times [-m,m] \right\}$
Shaft	1	$\left\{ (x,y) \in \mathbb{R}^2 \,\middle|\, x \in \left[\frac{1}{2}, n+\frac{1}{2}\right] \text{ and } y=0 \right\}$
Armatures	n	For the j^{th} armature we have $\left\{ (x,y) \in \mathbb{R}^2 \,\middle|\, x=j \text{ and } y \in [-m,m] \right\}$
Flags	$2mn-3m$	if $\ell_{j,k}$ is flagged then the attached flag is a regular triangle with side length 1.

Table 2.1: The quantity and coordinates of the logic engine components.

2.1.2 The mechanics of the logic engine

A *collision* of flags occur if either of the following occurs:

1. flags in the same row on adjacent armatures point toward each other.
2. a flag from the outermost armature A_n points towards the outer rigid frame.
3. a flag from the innermost armature A_1 points inwards of A_1.

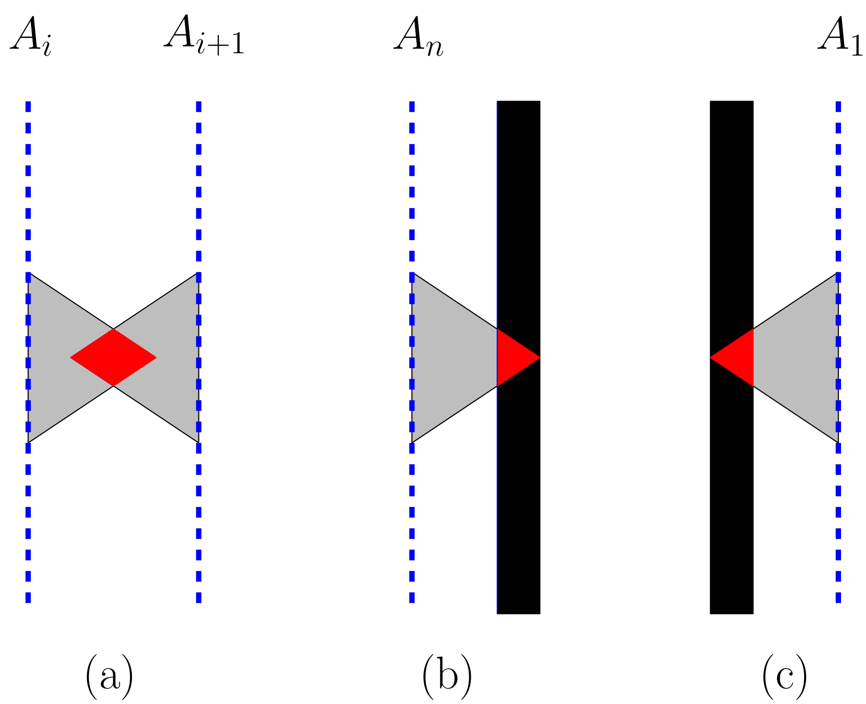


Figure 2.4: (a) Illustrates a adjacent flag collision at the same height, (b) and (c) illustrates a rigid frame collision.

The logic engine representation corresponding to Φ is to be configured such that no horizontally adjacent flags collide and flags do not collide with the rigid frame.

Lemma 1. *A row has a collision-free configuration if and only if it has at least one unglagged armature.*

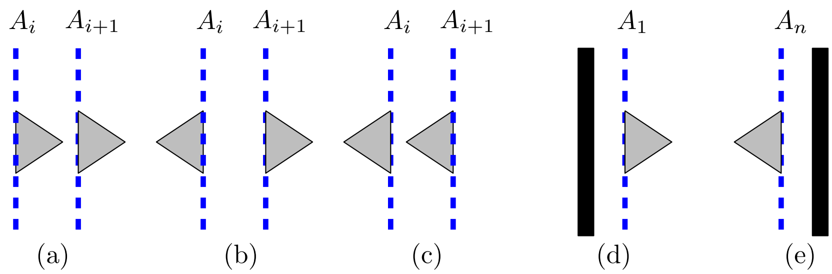


Figure 2.5: The following configuration of adjacent flags and flags that are adjacent to the rigid frame.

Proof. Suppose all armatures are flagged in a row. The flag on armature A_1 must point to the right otherwise we result in a rigid frame collision. A_2 must point to the right otherwise we result in a rigid frame collision. Without loss of generality, A_i and A_{i+1} must point to the right in order to prevent an adjacent flag collision. This implies that A_n must also point to the right which results into a rigid frame collision.

A same argument holds with the argument beginning with the flag on the armature A_n pointing to the left. Thus there is no collision-free configuration with all armatures flagged.

Suppose there is an unflagged armature in a row. Turn all flags towards the nearest unflagged armature. If there are flags on A_1 and A_n, point toward the interior thus they do not collide with the rigid frame. If there are flags on two consecutive armatures, they do not collide because the nearest unflagged armature cannot be between them. Therefore the row has a collision-free configuration. □

A logic engine is said to be *collision-free configurable* when every row has a collision-free configuration.

2.1.2.1 The Relationship of the Logic Engine and NAE3SAT

We can show that given an boolean formula in 3-CNF form, Φ, and a truth assignement, τ, where the variables are given a truth assignment such that there is at least one true literal and one false literal in each clause of Φ, then the corresponding logic engine to Φ is collision-free configurable.

Theorem 6. *Given an instance of a NAE3SAT, it is a "yes" instance if and only if the corresponding logic engine is collision-free configurable.*

Proof. Suppose we have an instance of a *NAE3SAT* that is a "yes" instance. This implies that there is a truth assignment such that each clause contains a true and a false literal. Now consider the logic engine corresponding to this instance. We now show that it has a collision free configuration.

For variables that are true, configure the armatures such that the flags corresponding to the non-negated literals reside above the shaft and the flags that correspond to the negated literals reside below this shaft. For variables that are false, configure the armatures in the opposite orientation. Each clause corresponds to a pair of rows in the logic engine, one row for non-negated literals and one for negated literals. Because the *NAE3SAT* is a yes instance, every row contains at least one unflagged armature. By Lemma 1, every row has a collision-free configuration.

Suppose we have an instance of a *NAE3SAT* such that the corresponding logic engine has a collision-free configuration. By Lemma 1 every row at least one unflagged armature. The k^{th} clause is represented by the k^{th} rows above and below the shaft. If the literal x_j is found in clause C_k, then the armature is unflagged in that row. If the literal $\bar{x}_j$ is found in clause C_k, then $\bar{l}_{j,k}$ is unflagged. All flags corresponding to negated literals reside below the shaft and flags corresponding to non-negated literals reside above the shaft. All together we have that every clause has a true literal and a false literal. Thus, we have a 'yes' instance of the $NAE3SAT$. ☐

Theorem 7. *Deciding whether a logic engine is collision-free configurable is NP-Hard.*

Proof. In table **??**, we defined the components of the logic engine in terms of polynomials in m and n. If there were a polynomial time algorithm that decides whether a given logic engine is collision-free configurable, then by Theorem 6 we would have a polynomial time algorithm to decide whether an instance of the NAE3SAT is a 'yes' instance. Since NAE3SAT is NP-Hard [6], there is no such algorithm unless $P = NP$. ☐

2.2 Logic Engines Represented as Polygonal Linkages

We can modify the mechanical structure of the logic engine to form a polygonal linkage.

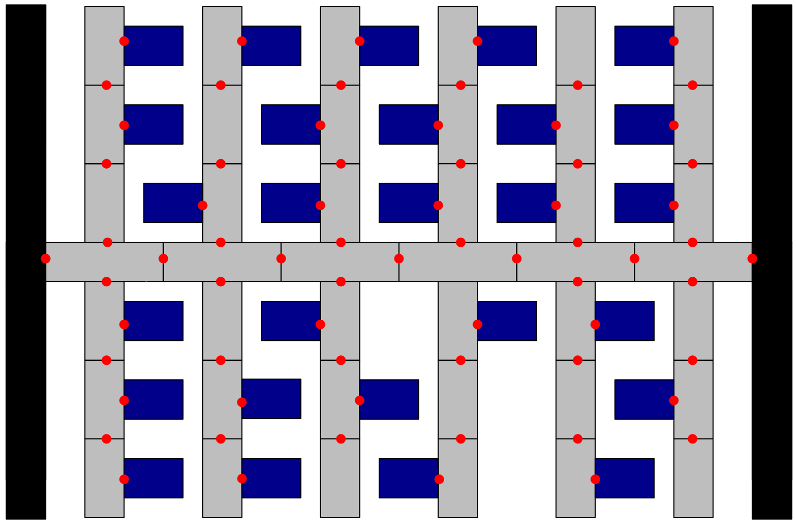


Figure 2.7: A logic engine realized as a polygonal linkage.

2.2.1 Construction of the Polygonal Linkage Logic Engine

Suppose we are given an boolean formula with m clauses and n variables in 3-CNF form, Φ, we construct the polygonal linkage similarly to the logic engine. The corresponding polygonal linkage $P_\ell = (\mathsf{PP}, \mathscr{H})$ has the following polygon components:

The large frame subcomponents are hinged on the left most and right most shaft subcomponents. Each adjacent shaft subcomponents are hinged and each shaft subcomponent has two orientations, a reflecion up and a reflection down about the shaft hinge points. On each shaft subcomponent there are two armature shaft subcomponents, one above the shaft subcomponent and one below the shaft subcomponent. By the two orientations of the shaft subcomponent, each armature subcomponent has to possible positions. Each armature comprises of m armature subcomponents that are hinged together; in total there are $2n$ armatures. Each armature subcomponent has two orientations, a reflection left and a reflection right about the armature hinge points. Label the armature subcomponents on the j^{th} armature starting from the shaft by $\ell_{j,1}, \dots, \ell_{j,n}$ on one side and $\bar{\ell}_{j,1}, \dots, \bar{\ell}_{j,n}$ on the other side of the shaft. Attach a rectangular flag specified in Table **??**, to some of these segments. Each segment is either flagged one or zero flags.

Component	Height	Width	Quantity
Shaft Subcomponent	1	3	n
Armature Subcomponent	2	1	$2 \cdot m$
Flag	1	1.5	$2mn - 3m$
Large Frame Subcomponent	$2 \cdot m$	1	4

Table 2.2: The components of PP specified polynomially in terms of the size of the boolean formula Φ.

1. If the literal x_j is found in clause C_k, then $\ell_{j,k}$ is unflagged.
2. If the literal $\bar{x}_j$ is found in clause C_k, then $\bar{l}_{j,k}$ is unflagged.

Each flag has two orientations with respect to armature it is attached to. Each flag has four potential positions, the flag can reflect left or right about the armature and the armature can reflect up or down about the shaft.

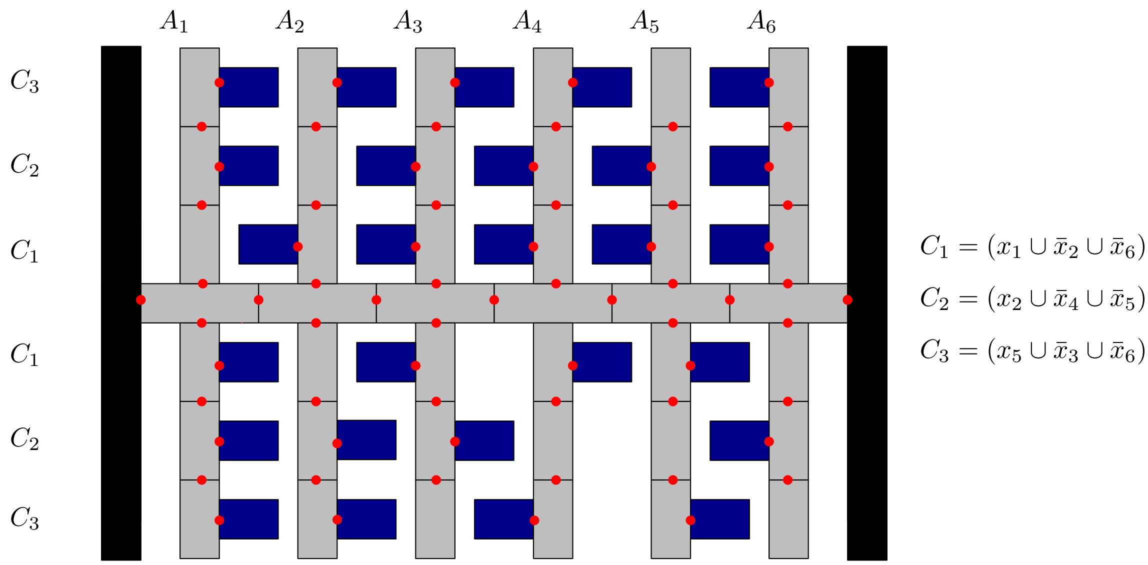


Figure 2.8: A polygonal linkage logic engine that corresponds to the boolean formula $\Phi = C_1 \cap C_2 \cap C_3$.

Theorem 8. *Given an instance of a NAE3SAT, it is a "yes" instance if and only if the corresponding polygonal linkage logic engine has a collision-free configuration.*

Proof. Suppose we have an instance of a *NAE3SAT* that is a "yes" instance. This implies that there is a truth assignment such that each clause contains a true and a false literal. Now consider the polygonal linkage logic engine corresponding to this instance. We now show that it has a collision free configuration.

For variables that are true, configure the armatures such that the flags corresponding to the non-negated literals reside above the shaft and the flags that correspond to the negated literals reside below this shaft. For variables that are false, configure the armatures in the opposite orientation. Each clause corresponds to a pair of rows in the polygonal linkage logic engine, one row for non-negated literals and one for negated literals. Because the *NAE3SAT* is a yes instance, every row contains at least one unflagged armature. By Lemma 1, every row has a collision-free configuration.

Suppose we have an instance of a *NAE3SAT* such that the corresponding polygonal linkage logic engine has a collision-free configuration. By Lemma 1 every row at least one unflagged armature. The k^{th} clause is represented by the k^{th} rows above and below the shaft. If the literal x_j is found in clause C_k, then the armature is unflagged in that row. If the literal $\bar{x}_j$ is found in clause C_k, then $\bar{l}_{j,k}$ is unflagged. All flags corresponding to negated literals reside below the shaft and flags corresponding to non-negated literals reside above the shaft. All together we have that every clause has a true literal and a false literal. Thus, we have a 'yes' instance of the *NAE3SAT*. $\square$

2.2.2 Properties for Weighted Trees and Polygonal Linkages

In order to perform our analysis for weighted trees and polygonal linkages, we'll want to use a suitable metric. The usual Euclidian distance will not suffice for this analysis and so we turn to the Hausdorff distance.

Hausdorff Distance Let A and B be sets in the plane. The *directed Hausdorff distance* is

$$d(A,B) = \sup_{a \in A} \inf_{b \in B} ||a-b|| \tag{2.1}$$

$h(A,B)$ finds the furthest point $ainA$ from any point in B. *Hausdorff distance* is

$$D(A,B) = \max\{d(A,B), d(B,A)\} \tag{2.2}$$

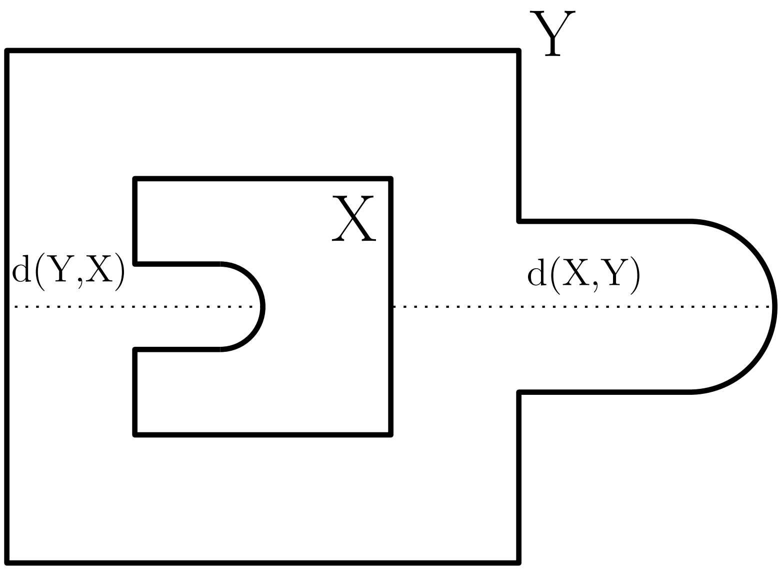


Figure 2.9: An illustrative example of $d(X,Y)$ and $d(Y,X)$ where X is the inner curve, and Y is the outer curve.

ε-approximation The weighted graph, G, is an *ε-approximation* of a polygon P if the Hausdorff distance between every realization such realization of G as a contact graph of disks and a congruent copy of P is at most epsilon. A weighted graph G is said to be a *$O(f(x))$-approximation* of a polygon P if there is a positive constant M such that for all sufficiently large values of x the Hausdorff distance between every realization such realization of G as a contact graph of disks and a congruent copy of P is at $M \cdot |f(x)|$. A weighted graph G is said to be a *stable* if it has the property that for every two such realizations of G, the distance between the centers of the corresponding disks is at most ε after a suitable rigid transformation.

2.3 Realizability Problems for Weighted Trees

Recall problems (7) and (9): given a positive weighted tree, T, is T the (ordered) contact graph of some disk arrangement where the radii are equal to the vertex weights. For now, we'll focus on a particular family of this problem space where the weighted trees can be realized as a *snowflake*. For $i \in \mathbb{N}$, the construction of the snowflake tree, S_i, is as follows:

- Let v_0 be a vertex that has six paths attached to it: p_1, p_2, ..., p_6. Each path has i vertices.
- For every other path p_1, p_3, and p_5:
  - Each vertex on that path has two paths attached, one path on each side of p_k.
  - The number of vertices that lie on the path attached to the j^{th} vertex of p_k is $i-j$.

A *perfectly weighted snowflake tree* is a snowflake tree with all vertices having weight $\frac{1}{2}$. A *perturbed snowflake tree* is a snowflake tree with all vertices having weight of 1 with the exception of v_0; in a perturbed snowflake tree, v_0 will have a radius of $\frac{1}{2}+\varepsilon$. For our analysis, all realizations of any snowflake, perfect or perturbed, shall have v_0 fixed at origin. This is said to be the canonical position under Hausdorff distance of the snowflake tree.

Consider the graph of the triangular lattice with unit distant edges:

$$\begin{aligned} V &= \left\{a\cdot(1,0)+b\cdot\left(\frac{1}{2},\frac{\sqrt{3}}{2}\right) : a,b\in\mathbb{Z}\right\} \\ E &= \{\{u,v\} : ||u-v|| = 1 \text{ and } u,v\in V\} \end{aligned}$$

The following graph, $G=(V,E)$ is said to be the *unit distance graph* of the triangular lattice. We can show that no two distinct edges of this graph are non-crossing. First suppose that there were two distinct edges that crossed, $\{u_1,v_1\}$ and $\{u_2,v_2\}$. With respect to u_1, there are 6 possible edges corresponding to it, with each edge $\frac{\pi}{3}$ radians away from the next. Neither edge crosses another; and so we have a contradiction that there are no edge crossings with $\{u_1,v_1\}$.

The perfectly weighted snowflake tree that is a subgraph over the *unit distance graph*, $G=(V,E)$, of the triangular lattice. To show this, for any S_i, fix $v_0 = 0\cdot\cdot(1,0)+0\cdot\left(\frac{1}{2},\frac{\sqrt{3}}{2}\right) = (0,0)\in V$ at origin. Next consider the six paths attached from origin. Fix each consecutive path $\frac{\pi}{3}$ radians away from the next such that the following points like on the corresponding paths: $(1,0)\in p_1$, $\left(\frac{1}{2},\frac{\sqrt{2}}{3}\right)\in p_2$, $\left(-\frac{1}{2}\mathsf{p}4,\frac{\sqrt{3}}{2}\right)\in p_3$, $(-1,0)\in p4$, $\left(-\frac{1}{2},-\frac{\sqrt{3}}{2}\right)\in p_5$, $\left(\frac{1}{2},-\frac{\sqrt{3}}{2}\right)\in p_6$. For S_i, there are i vertices on each path.

We define the six paths from origin as follows:

$$\begin{aligned} p_1 &= \left\{a\cdot(1,0)=\vec{v}\,\middle|\,a\in\mathbb{R}^+\right\} \\ p_2 &= \left\{a\cdot\left(\frac{1}{2},\frac{\sqrt{3}}{2}\right)=\vec{v}\,\middle|\,a\in\mathbb{R}^+\right\} \\ p_3 &= \left\{-a\cdot(1,0)+a\cdot\left(\frac{1}{2},\frac{\sqrt{3}}{2}\right)=a\left(-\frac{1}{2},\frac{\sqrt{3}}{2}\right)=\vec{v}\,\middle|\,a\in\mathbb{R}^+\right\} \\ p_4 &= \left\{a\cdot(-1,0)=\vec{v}\,\middle|\,a\in\mathbb{R}^+\right\} \\ p_5 &= \left\{a\cdot\left(-\frac{1}{2},-\frac{\sqrt{3}}{2}\right)=\vec{v}\,\middle|\,a\in\mathbb{R}^+\right\} \\ p_6 &= \left\{a\cdot(1,0)-a\cdot\left(\frac{1}{2},\frac{\sqrt{3}}{2}\right)=a\cdot\left(\frac{1}{2},-\frac{\sqrt{3}}{2}\right)\,\middle|\,a\in\mathbb{R}^+\right\} \end{aligned}$$

For S_i there exists i vertices on each path. We shall denote the i^{th} vertex on the j^{th} path as $v_{j,i}$. For each path defined above, the paths are defined as a set of vectors, $\vec{v}=a\cdot\vec{p}$ for some $a\in\mathbb{R}^+$ and $\vec{p}\in\mathbb{R}^2$. By setting $a=1,2,\ldots,i$, we obtain points that are contained in V. For $j=1,3,5$ and $l=1,\ldots,i$, there exists two paths attached to each vertex $v_{j,l}$. For S_i, each path attached to the k^{th} vertex of p_j, there are $i-k$ vertices. We will need to show that each of the $i-k$ vertices on each corresponding path are also in V.

The triangular lattice is symmetice under rotation about v_0 by $\frac{\pi}{3}$ radians. For each vertex $v_{1,l}$ for $l=1,2,\ldots,i-k$, we place two paths from it; the first path $\frac{\pi}{3}$ above p_1 at $v_{1,l}$ and $\frac{-\pi}{3}$ below p_1 at $v_{1,l}$ and call these paths $p^+_{1,l}$ and $p^-_{1,l}$ respectively. With respect to $v_{1,l}$, one unit along $p^+_{1,l}$ is a point on the triangular lattice and similarly so on $p^-_{1,l}$. Continuing the walk along these paths, unit distance-by-unit distance, we obtain the next point corresponding point on the the triangular lattice up to $i-k$ distance away from $v_{1,l}$. This shows that each of the $i-k$ vertices on $p^-_{1,l}$ and $p^+_{1,l}$ are in V. By rotating all of the paths along p_1 by $\frac{2\pi}{3}$ and $\frac{4\pi}{3}$, we obtain the the paths along p_3 and p_5 respectively, completing the proof.

[Display a perfect snowflake graph over the triangular lattice]

2.3.1 On the Decidability of Problem (7)

Recall that problem (7) states: given a positive weighted tree, T, is T the contact graph of some disk arrangement where the radii are equal to the vertex weights?

Proof. Suppose we are given a positive weighted tree, $T = (V_1, E_1)$. By the Disk Packing Theorem, there is a disk arrangement in the plane, D, whose contact graph, $G = (V_2, E_2)$ is isomorphic to T. We need to so that $G = T$ and the radii of the disks in D are equal to the vertex weights of T.

To show that $G = T$, we need to show that $V_1 = V_2$ and $E_1 = E_2$.

To show that the radii of the disks in D are equal to the vertex weights of T, we first consider .... □

Bibliography

[1] S. N. Bhatt and S. S. Cosmadakis. The complexity of minimizing wire lengths in VLSI layouts. *Inform. Process. Lett 25*, 4:263–267, 1987.

[2] H. Breu and D. G. Kirkpatrick. Unit disk graph recognition is NP-hard. *Comput. Geom*, 9:3–24, 1998.

[3] R. Connelly, E. D. Demaine, M. L. Demaine, S. P. Fekete, S. Langerman, J. S. B. Mitchell, A. Ribó, and G. Rote. Locked and unlocked chains of planar shapes. *Discrete Comput. Geom*, 44:439–462, 2010.

[4] R. Connelly, E. D. Demaine, and G. Rote. Straightening polygonal arcs and convexifying polygonal cycles. *Discrete Comput. Geom*, 30:205–239, 2003.

[5] P. Eades and N. C. Wormald. Fixed edge-length graph drawing is NP-hard. *Discrete Applied Mathematics*, 28:111–134, 1990.

[6] Michael R Garey and David S Johnson. *Computers and Intractibility*. WH Freeman and company New York, 1979.

[7] J. Kleinberg and E. Tardos. *Algorithm Design*. Pearson Education, 2006.

[8] Casimir Kuratowski. Sur le probleme des courbes gauches en topologie. *Fundamenta mathematicae*, 1(15):271–283, 1930.

[9] S.S. Skiena. *The Algorithm Design Manual*. Springer, 2009.

[10] I. Streinu. Pseudo-triangulations, rigidity and motion planning. *Discrete Comput. Geom*, 34:587–635, 2005.