Compatible paths on point sets with two convex layers

Zantema, A.M.

Award date:
2015

Technische Universiteit Eindhoven University of Technology

Department of Mathematics and Computer Science
Applied Algorithms Research Group

Compatible paths on point sets with two convex layers

Master Thesis

A. M. Zantema

Supervisors:
Prof. Dr. B. Speckmann
Dr. K. A. B. Verbeek

Final version

Eindhoven, November 2015

Abstract

Consider a point set P in the plane, consisting of two convex layers. This means that all points that do not lie on the convex hull are in convex position. Two edges are said to cross each other if they share a point other than a common endpoint, and a Hamiltonian path is a path that visits all points. Two non-crossing Hamiltoninan paths on P are compatible if their union is non-crossing. We define the graph G of which the vertices represent the non-crossing Hamiltonian paths on P and two vertices share an edge if the corresponding paths are compatible. We prove that the graph G is connected.

Contents

Chapter 1

Introduction

1.1 Context

How can we transform one graph into an other, given a point set and a certain transformation operation? There are many variations to this problem. The graph class can vary: we can for example transform triangulations or spanning trees. We can also use different kinds of transformation operations, that range from local to global operations. An example of a local operation is removing one edge and replacing it by one other. The graph is only altered in one place, the rest of the graph remains the same. This operation is called an *edge flip* and is illustrated for a triangulation in Figure 1.1. Global operations can replace more edges at a time, not altering the graph in one place, but possibly everywhere. An example is compatibility: two planar graphs are compatible if their union is also planar. It is possible to construct two compatible graphs of a certain class, say, spanning trees, such that they have no edges in common.
If every possible graph on the specified point set and of the specified class is represented as a vertex and two vertices share an edge if the corresponding graphs are exactly one transformation operation away from each other, this defines the transformation graph G. For various point sets, graph classes and transformation operations, the main focus lies on whether G is connected, and if so, on an upper and a lower bound on the diameter of G.

1.2 Problem description

We will be considering the transformation graph G that depends on the following:

- The point set P consists of two convex layers, which means that all points that do not lie on the convex hull, lie in convex position, and the points in P are in general position, which means that no three points are on one line.
- The graph class we consider is non-crossing Hamiltonian paths. This means that no two edges of the path share a point except for common endpoints, and the path visits all points.
- Two points in G are adjacent if the corresponding paths are compatible. This means that, in the union of the two paths, no two edges share a point except for common endpoints.

We will investigate whether the transformation graph is connected.

1.3 Related work

To our knowledge, there is no other work discussing properties of this specific transformation graph. However, there has been research to transformation graphs depending on point sets that do not necessarily consist of two convex layers, other graph classes than Hamiltonian paths and other flip operations.

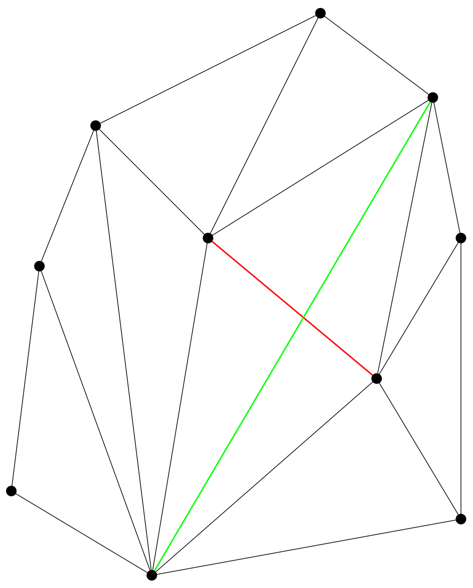
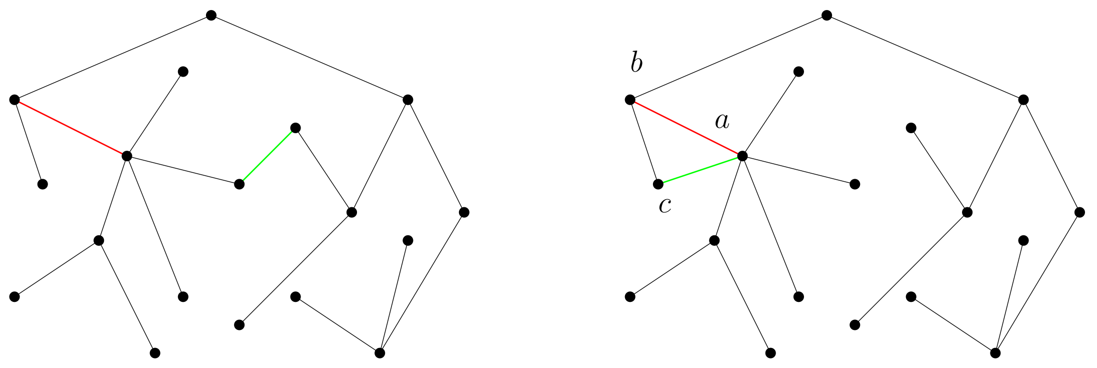


Figure 1.1: An example of an edge flip: remove the red edge and add the green edge.

Lawson [7] considers near-triangulations (that is, triangulations in which one face, called the *outer face*, is not a triangle), and their transformation operation is the *edge flip*, illustrated in Figure 1.1 and defined as removing one edge and adding an other edge such that the graph remains a near-triangulation. It appears that the transformation graph is connected, so every given near-triangulation can be turned into any other one using only flip operations, and $O(n^2)$ flips are sufficient. In [6], it is proven that this quadratic bound is tight.



(a) An example of an edge move: remove the red edge and add the green edge.

(b) An example of an edge slide: remove the red edge ab and add the green edge ac.

An other graph class that has been studied is the spanning tree. In [4], the transformation operation is defined similar to the edge flip above, namely removing an edge and adding an other one such that the graph remains a spanning tree; this is illustrated in Figure 1.2a. The resulting transformation graph is connected and any spanning tree can be transformed into any other one using a linear number of flips.

The authors of [1] also consider spanning trees, but use a different transformation operation: the *edge slide*, illustrated in Figure 1.2b. An edge ab can only be replaced by an edge ac if bc is an edge. The transformation graph is connected; in [3], it is shown that $O(n^2)$ edge slides are sufficient to transform any spanning tree into an other.

Apart from edge slides, [1] also discusses spanning trees under a more complex operation. There is an edge from tree T to tree T' in G if $T' = \min(T)$, that is, T' is the shortest-length spanning tree that is compatible to T. The operation is asymmetric, so G becomes a directed graph, but it is connected. More precisely, all edges are directed to the root, and the root represents the Euclidean minimum spanning tree T_{MST}, which is the only tree for which it holds that $T = \min(T)$. An upper bound on the diameter of G is $O(\log n)$.

A less complex operation is compatibility; two points in G share an edge if the corresponding trees do not cross each other. Since $T' = \min(T)$ implies that T and T' are compatible, G under the operation min is a subgraph of G under compatibility, so connectedness of G under min implies connectedness of G under compatibility, and $O(\log n)$ is an upper bound for the diameter for G under compatibility. The authors of [5] consider spanning trees under compatibility and prove a lower bound on the compatibility graph G of $\Omega(\log n/\log\log n)$.

The authors of [2] consider compatible perfect matchings. The notion of compatibility for matchings is similar to the notion of compatibility for spanning trees and paths: two graphs are compatible if their edges do not cross each other. As in the case for spanning trees, the compatibility graph G is connected and its diameter is bounded by $O(\log n)$, where $n = 2m$ is the number of points.

1.4 Thesis overview

We will prove that G is connected under compatibility for non-crossing Hamiltonian paths on point sets with two convex layers. To do so, we introduce some terminology and notation in chapter 2. Then, in chapter 3, we define a set of canonical paths and prove that the part C of G associated with these canonical paths is connected. Chapter 4 proves that any given arbitrary path is connected to C, by describing how we can construct a sequence of intermediate paths such that every path is compatible to the previous one; the first path of this sequence is compatible to the given path and the last path of this sequence is compatible to a canonical path. In chapter 5, we draw our conclusions and formulate open problems.

Chapter 2

Preliminaries

Let P be a point set with n points in general position (that is, no three points on a line), and let P have two convex layers. This means that the points on the convex hull form the *outer layer*, and all other points are in convex position and form the *inner layer*. Edges between adjacent points on the outer layer are called *outer edges* and edges between adjacent inner points are *inner edges*. Edges between two non-adjacent points on the same layer are called *inner* and *outer diagonals*, and edges between an outer point and an inner point are *connectors*. Outer diagonals and connectors may cross inner edges, these connectors and diagonals are said to be *cutting*. Note that inner diagonals can't cut inner edges, so a cutting diagonal is always an outer diagonal.

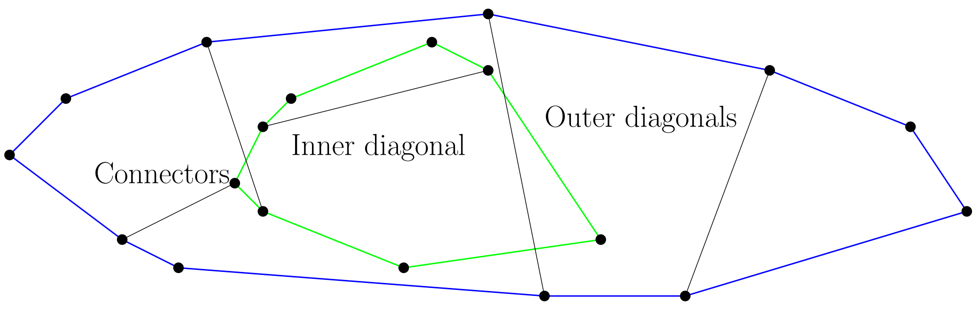
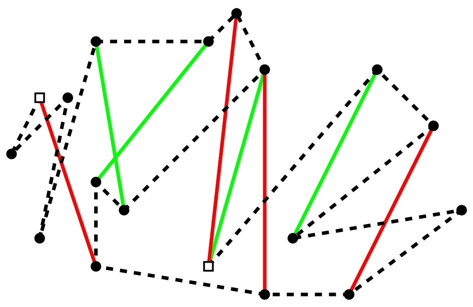
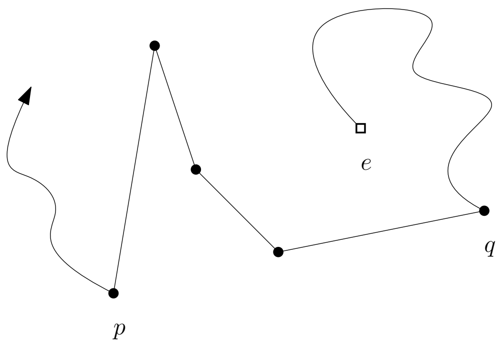


Figure 2.1: Several types of edges. The outer edges are indicated in blue and the inner edges in green. The black edges are, from left to right, a non-cutting and a cutting connector, an inner diagonal, and a cutting and a non-cutting outer diagonal.

Now consider non-crossing spanning paths, or Hamiltonian paths, on the point set P. Two edges are said to be *crossing* if they share a common point other than an endpoint, and a path is non-crossing if none of its edges crosses an other one. The edges of a non-crossing Hamiltonian path can be of any of the types of edges discussed above, and these edges are called *path edges*. All edges between two arbitrary points on P that do not cross path edges are called *feasible edges*. All path edges are feasible, and feasible edges may cross each other. Figure 2.2a shows a non-crossing Hamilton path, some feasible edges and some non-feasible edges. Endpoints of the path are indicated by an open square. We use solid circles and straight lines if we want to explicitly indicate certain points or edges, and use wiggly lines or wiggly arrows if the exact route of a path is not of interest. Subsets of a path are indicated by the name of the path and the first and last point. For example, if H is a path containing p and q, then $H(p, q)$ is the subpath starting at p and ending at q, including p and q. The notation for points and subpaths is illustrated in Figure 2.2b.



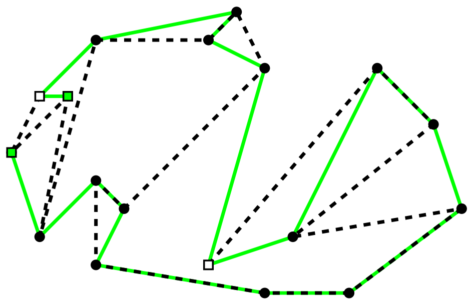
(a) A non-crossing Hamilton path (black, dashed), some feasible edges (green) and some non-feasible edges (red).



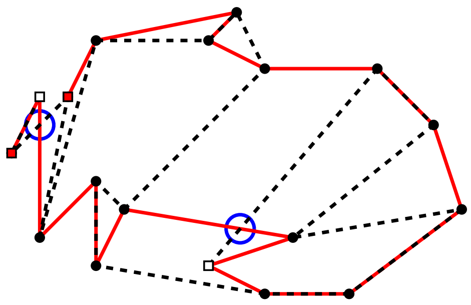
(b) A path H. All points in the subpath $H(p, q)$ are indicated, and the subpath $H(q, e)$ is indicated by a wiggly line. The rest of the path, including the other endpoint, is indicated by a wiggly arrow.

Figure 2.2: Terminology and notation for edges and points.

Two non-crossing Hamilton paths are *compatible* if their edges do not cross each other. Figures 2.3a and 2.3b show compatible and non-compatible paths.



(a) Two compatible paths.



(b) These paths are not compatible, as indicated by the blue circles.

Figure 2.3: Compatible and non-compatible paths.

Suppose path H is compatible with path I, and path I is compatible with path J, but H and J are not compatible. Then H and J are said to be *2-compatible*, because there exists a path that is compatible with both of them. More generally, two paths H_0 and H_k are called *k-compatible* if there exists a sequence of paths $H_0, H_1, \ldots, H_k$ such that every H_{i-1} is compatible with H_i for $1 \leq i \leq k$. We can also say that "the *distance of compatibility* between H_0 and H_k is k". In terms of the compatibility graph G, it means that there is a path of length k between H_0 and H_k. Note that, if two paths are k-compatible, they may also be $(k+i)$-compatible or $(k-j)$-compatible for some $i, j > 0$. If the exact distance of compatibility between two paths is not relevant, we use the term *eventually compatible* to refer to paths that are k-compatible for any $k > 0$. So, if two non-crossing Hamiltonian paths are eventually compatible, there exists a path in G between the points corresponding to those two paths. If we want to emphasize that two paths are compatible and not just eventually compatible, we say that they are *directly compatible*.

Given a point set P in general position in the plane and with two convex layers, and two arbitrary non-crossing Hamiltonian paths on P. Are these paths eventually compatible? In other words, is G connected?

Chapter 3

Canonical paths

The core element of a canonical path is a non-cutting connector, to connect the inner and outer layer. On the outer layer, the path follows the outer edges until the path contains all outer points. The same happens on the inner layer. In the end, the canonical path consists of a connector, all outer edges except for one and all inner edges except for one. This means that there exist four canonical paths per connector, which is illustrated in Figure 3.1.

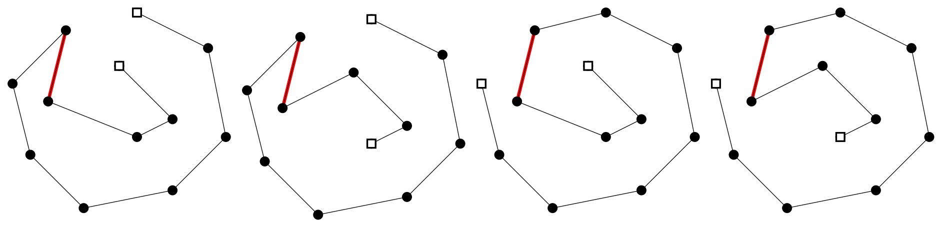


Figure 3.1: The four canonical paths with the same (red) connector.

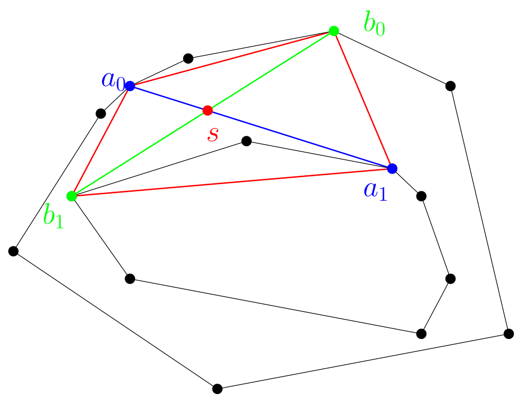
Before we can prove that all canonical paths are eventually compatible, we need the following lemma.

Lemma 1 *Let a_0a_1 and b_0b_1 be two non-cutting connectors that cross each other in s; a_0 and b_0 lie on the outer layer and a_1 and b_1 lie on the inner layer. Then a_0b_1 is a non-cutting connector.*

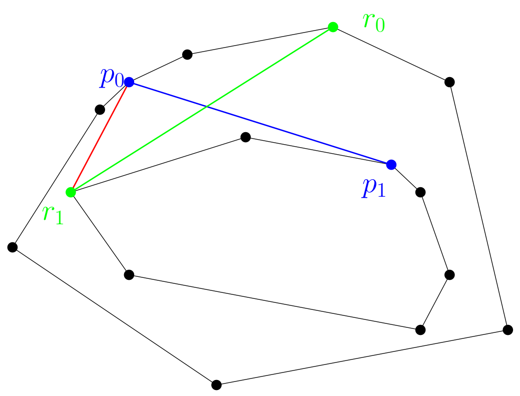
Proof. Consider the quadrilateral $a_0b_0a_1b_1$ in Figure 3.2a. Its diagonals cross, so it is convex and its diagonals divide it in four triangles. Only the triangle a_1b_1s can contain inner edges, since we know that the inner edges do not cross a_0a_1 or b_0b_1 because those connectors are not cutting. Since a_0b_1 does not lie on this triangle, we know it does not cross any inner edge and by definition it is not cutting. □

Lemma 2 *Two arbitrary canonical paths are either directly compatible or 2-compatible.*

Proof. All outer edges are feasible with respect to any canonical path, and all inner edges are too because the connector in a canonical path is never cutting. The only way for two canonical paths to be not compatible is when the connectors of those paths cross each other. This means that if the connectors of two canonical paths do not cross each other, the paths are compatible. Even if two canonical paths are not directly compatible, they are 2-compatible. Suppose two canonical paths H and I are not compatible. Let p_0 and p_1 denote the outer and inner point, respectively, of the connector of H, and let r_0 and r_1 be the outer and inner point, respectively, of the connector of I. Then the edge p_0r_1 does not cross p_0p_1 or r_0r_1, and, by Lemma 1, it is not cutting. This is illustrated in Figure 3.2b. Therefore, canonical paths with edge p_0r_1 as its



(a) The quadrilateral formed by the two connectors.



(b) Edge p_0r_1 is not cutting and does not cross p_0p_1 or r_0r_1.

Figure 3.2: Finding a connector that is feasible with respect to two incompatible canonical paths.

connector are compatible with both H and I, so canonical paths that are not directly compatible are 2-compatible. □

Chapter 4

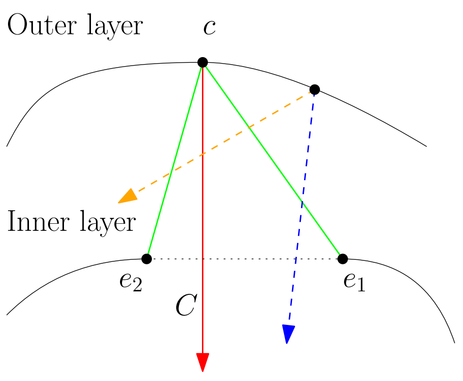
Eventual compatibility for arbitrary paths

We will start with the simplest case and assume that there are no cutting edges (cutting connectors or cutting diagonals).

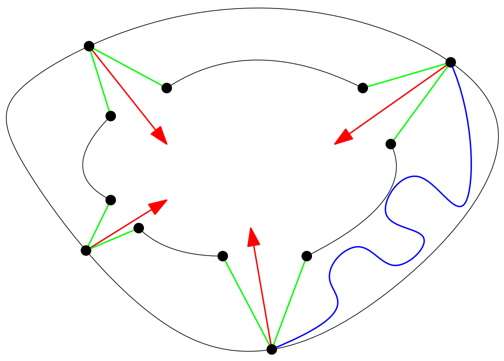
Lemma 3 *All paths without cutting edges are directly compatible to a canonical path.*

Proof. Paths without cutting edges must have at least one connector, otherwise the inner and outer layer would not be connected. Take a connector of the path and build a canonical path around it; the outer edges are feasible because outer edges are always feasible, and the inner edges are feasible because the path does not have any cutting connectors or cutting diagonals. This means that any path without cutting connectors or cutting diagonals is directly compatible to a canonical path. □

4.1 Dual chains



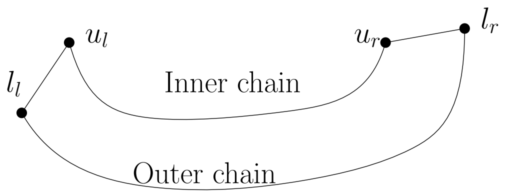
(a) A cutting connector (red) of a path cuts an inner edge (black, dotted), but the green edges are still feasible.



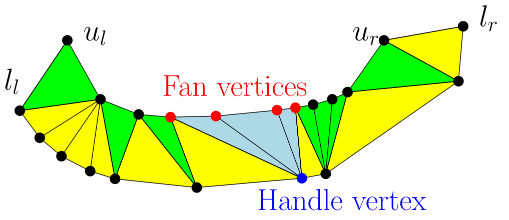
(b) Four cutting connectors (red) divide the point set in four dual chains.

Figure 4.1: Cutting connectors split the point set.

Now we take a look at paths with cutting connectors, but still without cutting diagonals. These paths are not directly compatible to canonical paths because the cutting connectors cut inner edges, so those inner edges are not feasible. Although we consider paths that may have many cutting connectors, we assume for a start that no two cutting connectors cut the same inner edge, and no two cutting connectors share the same outer point. These special cases are treated in Section 4.2. Let H be an arbitrary non-crossing path with cutting connectors (no two cutting the same inner edge and no two sharing the same outer point) and without cutting diagonals. We want to find a path without cutting connectors that is eventually compatible to H, to prove that paths with cutting connectors are eventually compatible to canonical paths. Suppose the cutting connector C in Figure 4.1a (drawn as a red arrow starting at the outer point c) is a part of H. It cuts inner edge e_1e_2, so that edge is not feasible, but the edge e_1c (green) is feasible. It can not be crossed by a non-cutting connector of H (orange), because such an edge would cross C and we know the path is non-crossing. The edge e_1c can also not be crossed by a cutting connector (blue), because such a connector would cut e_1e_2 and we were assuming that no two cutting connectors cut the same inner edge. This means that e_1c is not crossed by a path edge, so by definition it is feasible. The same feasibility property holds for e_2c (also green).



(a) A dual chain.

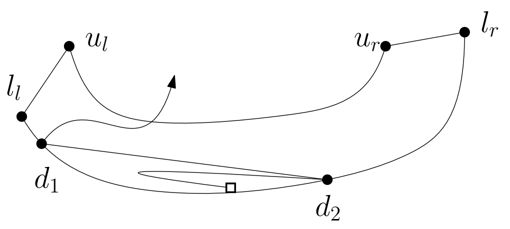


(b) A triangulation consists of fans, and fans consist of one handle vertex and two or more fan vertices.

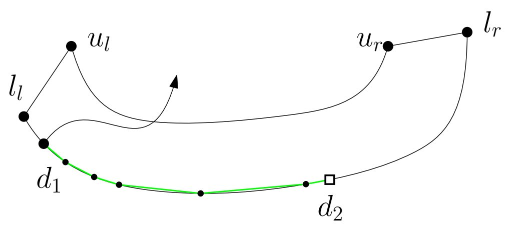
Figure 4.2: A dual chain can be divided into fans.

The path H can have any number of cutting connectors; for the sake of an example, suppose it has four. Figure 4.1b shows what happens if we consider all four cutting connectors. Intuitively, a path that zig-zags between the outer and the inner layer (of which a subpath is drawn in blue) and that picks up all points would be compatible to H and have no cutting connectors. It is such a path that we will construct. The four cutting connectors "split" the point set in four parts that we call *dual chains*. A dual chain consists of two convex chains, the inner chain running from u_l to u_r and the outer chain running from l_l to l_r. The inner chain contains only points of the inner layer and the outer chain contains only points of the outer layer. The outer chain must contain at least two points and the inner chain at least one, so it may be the case that $u_l = u_r$. All edges between consecutive points on the chains should be feasible, and the edges $u_l l_l$ and $u_r l_r$ should also be feasible. This is illustrated in Figure 4.2a. We will use dual chains to construct a path I we need (without cutting connectors and compatible to H). The new path I will consist of subpaths that each visit all points in one of the dual chains. To link the subpaths, they must each start at l_l, visit all points in the dual chain and end at l_r, using only feasible edges. If we create a subpath in all dual chains, we get a cycle and we remove one arbitrary edge to create I. To find feasible subpaths in the dual chains, we triangulate the dual chain such that all edges of the triangulation are feasible. The path edges are first added to the triangulation, then we triangulate the resulting simple polygons. In this way, the triangulation consists of feasible edges only. A triangulation on two chains consists of *fans*, as illustrated in Figure 4.2b. A fan is a polygon with one vertex, the *handle vertex*, on one chain and two or more *fan vertices* on the other chain. A *simple fan* is the smallest possible fan and has only two fan vertices. Note that handle vertices are also fan vertices for the adjacent fans, but fan vertices are not necessarily a handle vertex for an other fan. In other words, all vertices are fan vertices, but not all vertices are handle vertices so we can distinguish handle- and non-handle vertices. An other property of such a chain of fans is that the handle vertices of two adjacent fans lie on different chains.

The triangulation must consist of layer edges and (non-cutting) connectors. Outer (non-cutting)



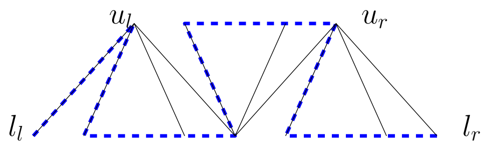
(a) A non-cutting outer diagonal.



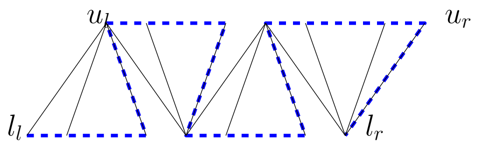
(b) Rerouting the diagonal.

Figure 4.3: Removing a non-cutting outer diagonal.

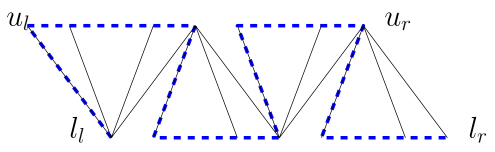
diagonals in the path cause some non-cutting connectors we need to be non-feasible. To avoid this, we create a path H' that is the same as H but without the non-cutting outer diagonal. One endpoint of the path has to be below the diagonal, the other endpoint has to be above the diagonal. This means the path can easily be rerouted. Let d_1d_2 be the diagonal we want to remove, and without loss of generality, assume that d_1 shares an edge with a point above the diagonal and d_2 shares an edge with a point below the diagonal; this is illustrated in Figure 4.3a. The new path, illustrated in Figure 4.3b, will start at d_2, follow the outer layer until d_1 and continue along the original path H. This new path H' is compatible to H because the only edges of H' that are not in H are outer edges, and outer edges are always feasible.



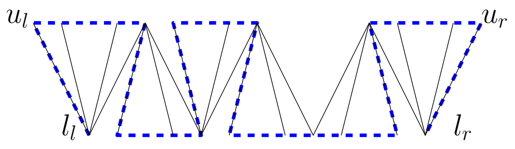
(a) Case nn. The path goes handle-to-fan.



(b) Case nh. The path goes fan-to-handle.



(c) Case hn. The path goes handle-to-fan.



(d) Case hh with a simple fan in the middle. The path goes handle-to-fan before the simple fan and fan-to-handle after the simple fan.

Figure 4.4: Four cases of triangulation in a dual chain.

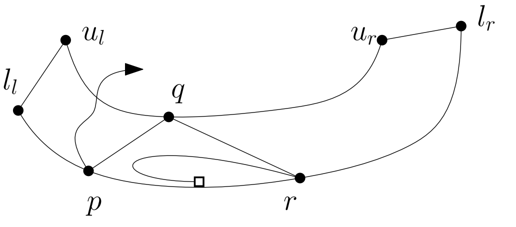
After triangulating the dual chain, we want to create a subpath I' of I that starts in l_l, visits all points of the dual chain and ends in l_r, and uses only the edges of the triangulation. We distinguish four cases: l_l and l_r can both be a handle or non-handle vertex, independently.

- Case nn in Figure 4.4a: if l_l and l_r are non-handle vertices, I' goes handle-to-fan. This means that when the path visits a fan, it starts at the handle vertex and then visits the fan vertices. An exception is the leftmost fan vertex of every fan. This vertex is visited before the handle of this fan, either because this is l_l and it is the starting point of the path, or because that fan vertex is the handle vertex of the previous fan and it has already been visited. After visiting the handle of a fan, the path skips the leftmost fan vertex and goes to the second-leftmost fan vertex. Then it continues along the remaining fan vertices, if any. In this way, the path will visit all points in the dual chain and automatically end at l_r.
- Case nh in Figure 4.4b: if l_l is a non-handle vertex and l_r is a handle vertex, I' goes fan-to-handle. This means that when the path visits a fan, it starts at the fan vertices and then

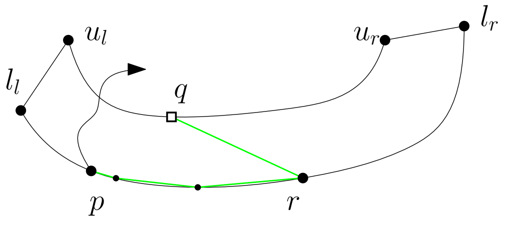
visits the handle vertex. An exception is the rightmost fan vertex of every fan. This vertex is also the handle vertex of the next fan and it must be visited in the future. Therefore, the path skips the rightmost fan vertex and visits the handle vertex directly after the second-rightmost fan vertex. Then it continues along the fan vertices of the next fan. Because there is no fan after the last one, the path does not skip the rightmost vertex of the last fan.

- Case hn in Figure 4.4c: if l_l is a handle vertex and l_r is a non-handle vertex, I' goes handle-to-fan. As in case nn, the path first visits the handle of a fan and then the fan vertices, skipping the leftmost fan vertex because it has already been visited in the previous fan. Because there is no fan before the first one, the path does not skip the leftmost fan vertex in the first fan.
- Case hh in Figure 4.4d: if l_l and l_r are handle vertices, we need a simple fan s "in the middle", that is, not as the first or last fan. Until s, the path goes handle-to-fan as in case hn, and after s, the path goes fan-to-handle as in case nh. This is possible because the leftmost fan vertex of s is skipped (because it has already been visited in the handle-to-fan regime) and the rightmost fan vertex is also skipped (because it will be visited in the fan-to-handle regime). Since s has only two fan vertices, there are no more fan vertices left to visit and the path I' can go directly from the previous fan to the next fan. Recall that the handle vertex of s is also a fan vertex of both the previous and the next fan.

  If there is no simple fan in the middle, we can construct one or alter the triangulation such that it becomes case nh or hn; this is explained in the following paragraphs.



(a) A corner forces an endpoint.

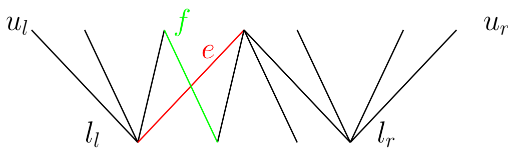


(b) Rerouting the corner.

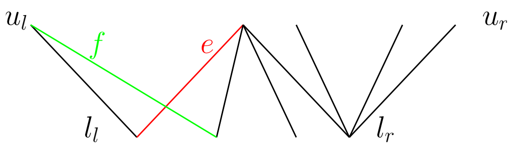
Figure 4.5: Removing a corner.

Problems can occur when there is a *corner* in the dual chain, see Figure 4.5a. If the original path H visits some point p on the outer chain, then continues to some point q on the inner chain and then continues to some point r on the outer chain such that p and r are not adjacent, the triangle pqr is called a corner. Such a configuration splits the point set in two; there must be one endpoint inside the corner and one endpoint outside. If there were two endpoints outside the corner, there is no way for the path to visit the points inside the corner. By a similar argument, there can not be two endpoints inside the corner. So there must be one endpoint outside the corner and one inside. This means that the corner and everything inside it can be rerouted in order to remove the corner. Without loss of generality, we can assume that the path before p is outside the corner and the path after r is inside. The new path H', illustrated in Figure 4.5b, will start at q, go to r and follow the points on the outer chain until p, and then continue along the original path H. The paths H and H' are compatible. They use the same edges outside the corner, so those edges are feasible, and the same holds for qr. The only edges in H' that may not be in H are the outer edges between p and r, but outer edges are always feasible, so H and H' are compatible.

If l_l and l_r are both handle vertices and there is no simple fan "in the middle" (not first or last), we can alter the triangulation such that it either contains a simple fan in the middle or l_l or l_r becomes a non-handle vertex. There are at least three fans in this configuration: l_l and l_r are distinct points, so if they are both handle vertices, they belong to different fans. And they can not be adjacent because their handle vertices lie on the same chain, so there must be at least one fan between them. That makes at least three fans. Now consider the leftmost edge e of the



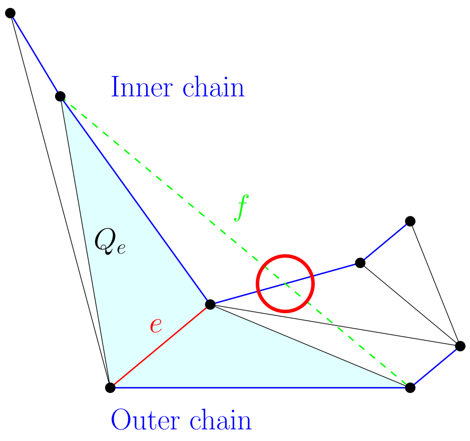
(a) Flipping the leftmost edge of the second fan, replacing e by f, thus creating two simple fans.



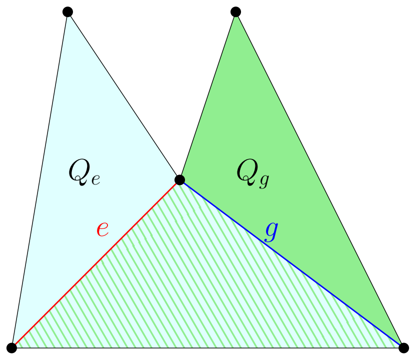
(b) Flipping the leftmost edge of the second fan, replacing e by f, thus turning l_l into a non-handle vertex.

Figure 4.6: Flipping an edge in order to solve case hh without a simple fan in the middle.

second fan; the handle vertex of this fan lies on the inner chain. Removing this edge creates a quadrilateral Q_e, and e is one of its diagonals. Replacing e by the other diagonal f gives an other triangulation that has either two additional simple fans (Figure 4.6a), or, if e was shared with a simple fan at the left end of the dual chain, l_l is no longer a handle vertex (Figure 4.6b). This procedure is called *flipping* edge e. Two problems can occur: e can be a path edge, so f is not feasible, or Q_e is not convex, so f is a cutting connector; this second problem is illustrated in Figure 4.7a. We will say that e is *non-flippable*, more precisely, e is *P-non-flippable* if it is a path edge and *Q-non-flippable* if Q_e is not convex.



(a) If Q_e is not convex, f becomes a cutting connector.

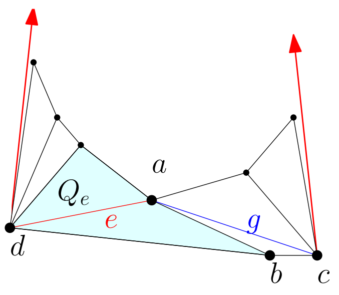


(b) If both the left- and rightmost edge of a fan yield a quadrilateral that is not convex, this fan has to be a simple fan.

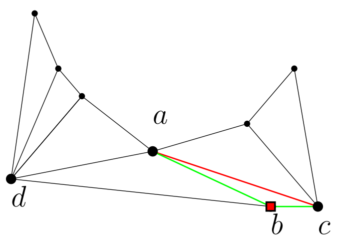
Figure 4.7

If e is non-flippable, we do not consider the leftmost edge e of the second fan, but try to flip the rightmost edge g. It is not possible that e and g are both path edges, that would be a corner. Nor is it possible that both Q_e and Q_g are not convex. In such a configuration, illustrated in Figure 4.7b, the fan can only have two fan vertices so we already have the simple fan "in the middle" we are looking for. Any one edge in the fan between e and g would be an edge of Q_e and Q_g, and turn Q_e or Q_g (or both) into a convex quadrilateral.

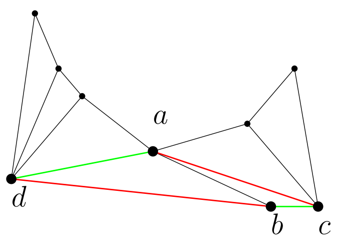
However, the problem remains if one of the edges e and g yields a quadrilateral that is not convex and the other is a path edge; such a fan is called a *non-flippable fan.* We will first consider the minimal case when the dual chain consists of only three fans, and without loss of generality, we assume that e yields a quadrilateral Q_e that is not convex and g is a path edge. This configuration is illustrated in Figure 4.8a. The point a is the handle of the second fan, c is the rightmost point



(a) A non-flippable fan.



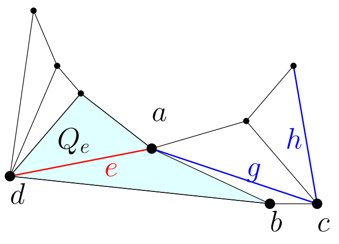
(b) Rerouting the path in case b is an endpoint, by removing the red edge and adding the green edge.



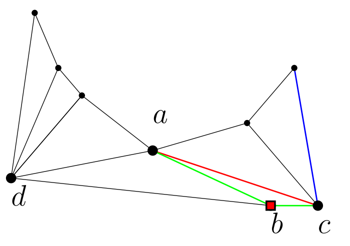
(c) Rerouting the path in case b is no endpoint, by removing the red edges and adding the green edges.

Figure 4.8: Rerouting a P-non-flippable edge.

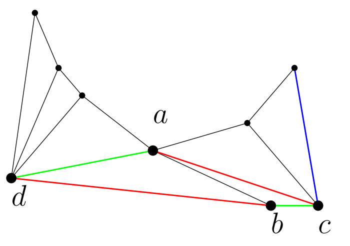
in this fan, and b and d are the second- and third-rightmost point of the fan. If there are more than three fan vertices, d is not the leftmost point in the fan. The edge g is equivalent to ac and is indicated in blue, and the cutting connectors that define this dual chain are red. The point b is either an endpoint or no endpoint, and we will consider these cases separately. If it is an endpoint (Figure 4.8b), we can create a new compatible path that does not contain ac. This is done by removing the edge connected to b and adding the edges cb and ab; these edges are feasible because these edges are part of the triangulation. For the rest, the new path is the same as the original path. If b is no endpoint (Figure 4.8c), there must be two path edges connected to this point. A close inspection shows that this must be the edges ab and bd. Recall that d is the point to the left of b on the outer chain, but not necessarily the leftmost. All edges of the triangulation are feasible, so we know that they are not crossed by path edges. This means that only ab and bd are possible path edges; bc can not be a path edge because c would then have a degree of 3, which is of course not possible in a path. We can construct a new compatible path by removing ac and bd and adding bc and ad; these edges are feasible because they are part of the triangulation.
Whether b was an endpoint or not, we have constructed a compatible path that does not contain ac, so we can now flip this edge as described above to create a simple fan or turn l_l or l_r in a non-handle vertex.



(a) Two P-non-flippable edges sharing the outer point c.



(b) Rerouting the path in case b is an endpoint, by removing the red edge and adding the green edges.



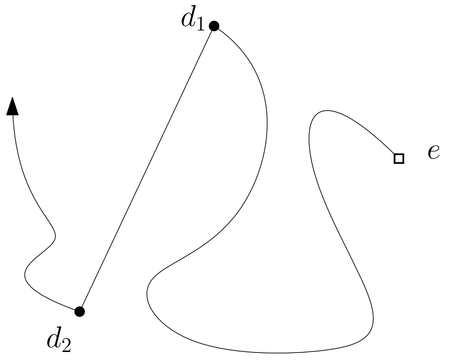
(c) Rerouting the path in case b is no endpoint, by removing the red edges and adding the green edges.

Figure 4.9: Rerouting a P-non-flippable edge.

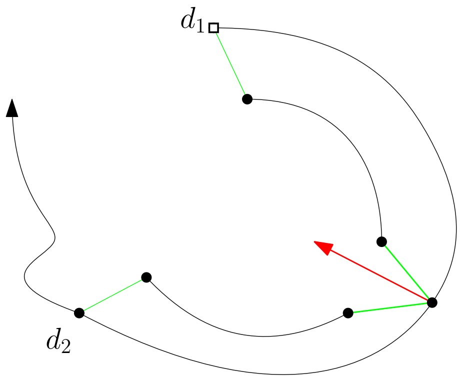
The above procedure only works for a minimal dual chain; we will now consider larger dual chains, where l_l and l_r are still handle vertices. A larger dual chain consists of multiple fans with their handle on the inner chain. If the leftmost of these fans is a non-flippable fan, we simply choose an other fan with its handle on the inner chain and try to flip the leftmost- or rightmost edge. If all fans with the handle on the inner chain are non-flippable, there are two possibilities. Either there is a P-non-flippable edge sharing an outer point with one of the cutting connectors that define this dual chain, in which case we can apply the same procedure as in the case of a minimal dual chain. Or there are two P-non-flippable edges that share a point on the outer layer. This is illustrated in Figure 4.9a; the blue edges g and h share the point c. In that case, it turns out that we can also use the same procedure to remove a P-non-flippable edge. If b is an endpoint (Figure 4.9b), we remove the edge that was connected to b and edge ac and add the edges cb and ab. If b is no endpoint (Figure 4.9c), it can only be connected by ab and bd. We remove ac and bd and add cb and ad.

We now have a means of constructing a path I that does not contain cutting connectors and that is eventually compatible to H, a path with cutting connectors but without cutting diagonals, as long as the cutting connector(s) of H split(s) the point set in valid dual chains with at least one point on the inner chain and at least two points on the outer chain.

Observation 1 *An outer diagonal d_1d_2 splits the point set in two. Let e_1 and e_2 be the endpoints of path H. Then all points on $H(d_1e_1)$ lie on one side of the diagonal, and all points on $H(d_2e_2)$ lie on the other side.*



(a) A tail.



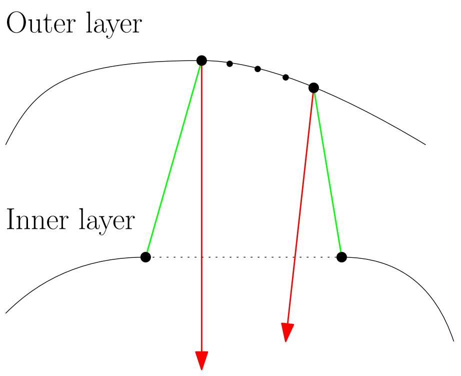
(b) One cutting connector (red) divides the tail in two dual chains.

Figure 4.10: Dividing a tail in dual chains.

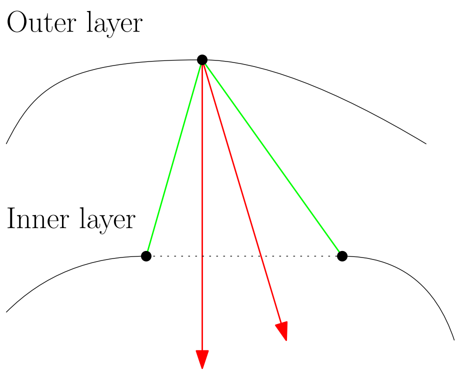
Using this observation, we can define a *tail* of a path K_k with k cutting diagonals. Start at an endpoint and follow K_k until the first cutting diagonal d_1d_2; the visited part of the path, including the diagonal, is a tail. A path has two endpoints so, counter-intuitive f a path has only one cutting diagonal, that diagonal is part of both tails. By Observation 1, all points in the tail lie on or on one side of d_1d_2 and all other points lie on the other side, which is illustrated in Figure 4.10a. Instead of splitting the whole point set in dual chains, we only split the tail as illustrated in Figure 4.10b. If there are no cutting connectors in the tail, the cutting diagonal splits the tail in one dual chain. The new path we will construct, starts at d_1 and moves along the dual chains until d_2, from where it will follow the original path K_k. In this way, we have constructed a new path K_{k-1} that is eventually compatible to K_k and has one cutting diagonal less. By applying this procedure multiple times, we can construct a sequence of paths $K_k, K_{k-1}, \ldots, K_0$ such that every K_i is eventually compatible to K_{i-1} and has i cutting diagonals. This means that paths with cutting diagonals are eventually compatible to a path without cutting diagonals, and such a path is in turn eventually compatible to a canonical path. This means that all paths are eventually compatible to a canonical path – as long as all dual chains are valid. In the following section, we will discuss special cases when the cutting connectors yield no valid dual chains.

4.2 Special cases

A dual chain needs to have at least one point on the inner chain and two points on the outer chain. If two cutting connectors have different outer points and cut the same inner edge (Figure 4.11a), the resulting structure is no valid dual chain because its inner chain is empty. However, if we are constructing a new (sub)path that is to visit all points and moves along the dual chains, we let the new path move along the outer layer from the previous dual chain to the next. In this way, we only add feasible edges to the path (outer edges are always feasible) and the path visits all points. If two cutting connectors cut the same edge and share the same outer point (Figure 4.11b), this yields the same valid dual chains as when there had been only one cutting connector.



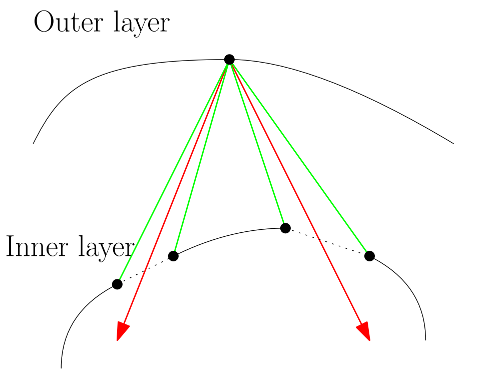
(a) Two cutting connectors with different outer points that cut the same edge yield a non-valid dual chain with an empty inner chain.



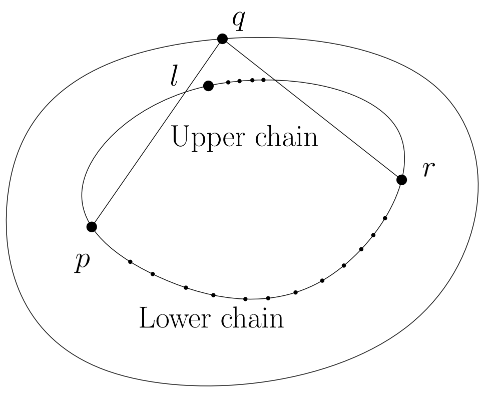
(b) If the two cutting connectors that cut the same edge have the same outer point, the resulting dual chains are the same as when there had been only one cutting connector.

Figure 4.11: Two special cases when two cutting connectors cut the same inner edge.

The only big problem occurs when the outer chain of a dual chain consists of only one point, see Figures 4.12a and 4.12b. This happens in a special case when two cutting connectors pq and qr share a point q on the outer layer, and there is at least one point in the triangle pqr. This configuration is called non-empty cutting connector triangle. The points in the triangle are called the *upper chain*, the leftmost of these points is called l and these points are part of the inner layer. The points on the inner layer between p and r form the *lower chain*.



(a) A non-valid dual chain with only one point on the outer chain.



(b) A non-empty cutting connector triangle.

Figure 4.12: The origin and definition of a non-empty cutting connector.

A path L_m with m non-empty cutting connector triangles can not be split in valid dual chains. Therefore, we want to find an intermediate path that is eventually compatible to L_m without the non-empty cutting connector triangles so it can be split in valid dual chains. If a path L_m contains m non-empty cutting connector triangles, we create a sequence of intermediate paths $L_{m-1}, \dots, L_0$, each L_i compatible to L_{i-1} and each containing *at most* i non-empty cutting connector triangles. We will describe a procedure, called *stealing*, that is intended to remove one non-empty cutting connector triangle at a time, but in rare occasions, it "accidentally" removes more than one. In those cases, L_{i-1} contains $j < i - 1$ non-empty cutting connector triangles, so we let $L_{i-1} = L_{i-2} = \cdots = L_j$.
The procedure of stealing works as follows. If we *steal* l, the edge pq is removed and the *thief edges* pl and lq are added; thief edges have to be feasible. The resulting graph is no path (except for a very special case), so we have to fix this. It becomes a case distinction that is explained below. In general, we give an edge a red colour if this edge belongs to L_i but we do not want it to belong to L_{i-1}, so we "remove" that edge, and we give an edge a green colour if this edge does not belong to L_i but we want it to belong to L_{i-1}, so we "add" this edge. Edges that belong to both L_i and L_{i-1} remain black. Blue indicates recursion; the blue point has to be stolen by the blue thief edges. Eventually, possibly after some recursion steps, all edges are either black, green or red. The union of the black and red edges is L_i, and if all green edges are placed correctly, the union of the black and the green edges is the non-crossing Hamiltonian path L_{i-1}. Not only do the green edges have to be feasible with respect to L_i, but they also should not cross each other, otherwise L_{i-1} is not a non-crossing path. Whenever we add a green edge, we make sure it is feasible. By construction, green edges that are added in recursive steps are always added in an area where previously no green edges were added. In cases where recursion is needed, the area where new green edges can be added is indicated in gold. In the end, when no more recursive steps are needed, we can remove the red edges and add the green edges, thereby replacing the "original" path L_i by the "new" path L_{i-1}. The case distinction consists of five possibilities that will be handled in this order:

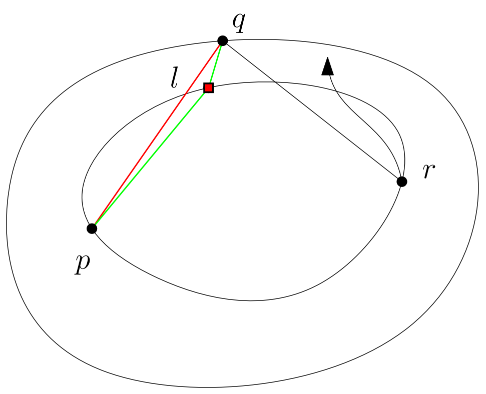


Figure 4.13: Case 1.

1. Point l is an endpoint of L_i. Remove pq and the path edge that was connected to l, and add the thief edges pl and lq. See Figure 4.13.

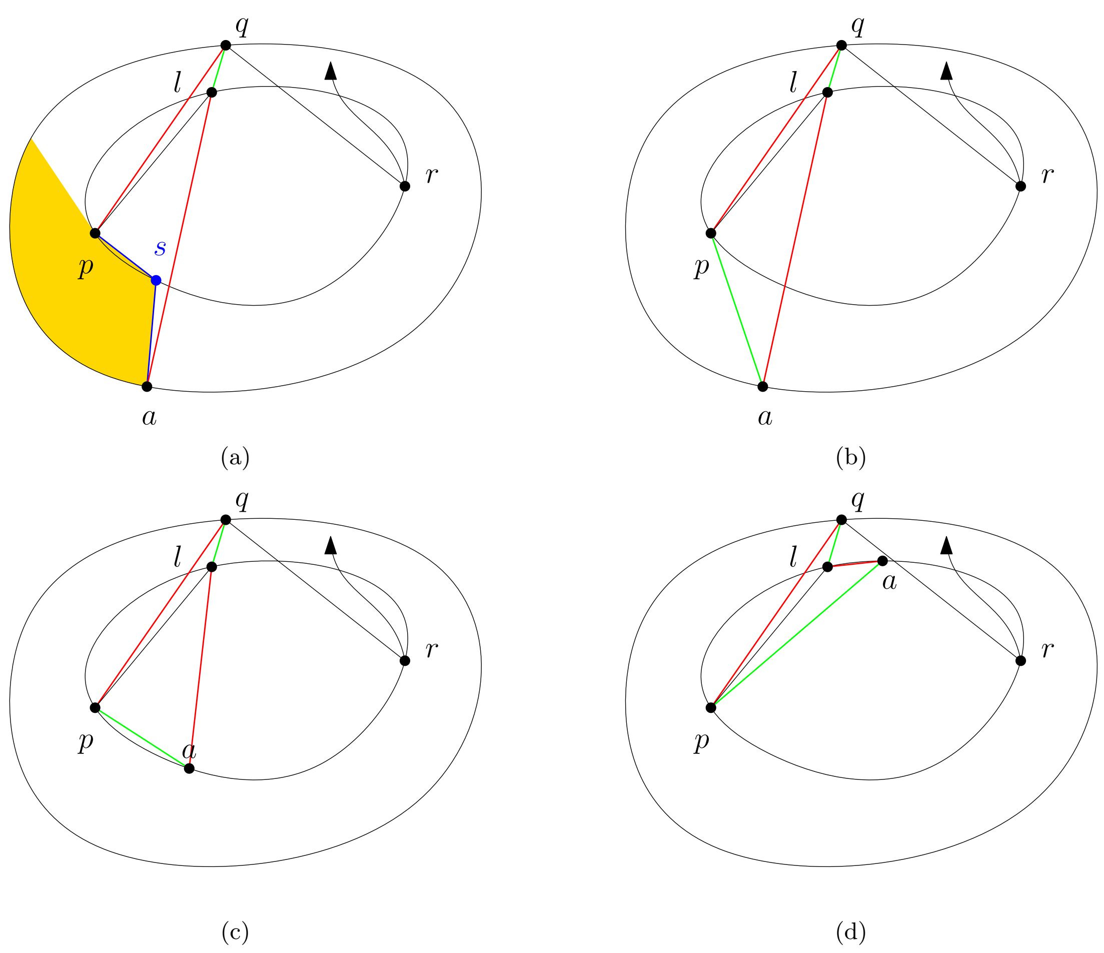


Figure 4.14: Case 2.

2. The thief edge pl is a path edge of L_i, and point a shares an edge with l. The edge ap may or may not be feasible. If it is not feasible (Figure 4.14a), we know that a has to be on the outer layer. If it had been on the inner layer, any edge crossing ap would have no point to go to because there are no points in the triangle alp. To reroute, we need the geodesic inside the triangle alp. Because everything inside that triangle is a convex chain, the geodesic consists of one point: the rightmost point inside alp. This point is called s. We remove pq and al, add lq, and we recurse by stealing the rightmost point s with the thief edges ps and as. The area of recursion is bounded by a, s, p, and the furthest point p can connect to without cutting. Recursion can only happen on points on the lower chain, so the recursion always ends.
   If the edge ap is feasible, a can be on the outer layer (Figure 4.14b), on the lower chain (Figure 4.14c) and on the upper chain (Figure 4.14d). To reroute, we remove the edges pq and al and add the edges lq and ap.

3. Points a and b share a path edge with l and edge ab is feasible. Remove edges pq, bl and la and add ab and the thief edges pl and lq. There are six possible configurations. Both points a and b can be on the upper chain (Figure 4.15a); both on the lower chain (Figure 4.15b); both on the outer layer (Figure 4.15c); one on the upper chain and one on the lower chain (Figure 4.15d); one on the upper chain and one on the outer layer (Figure 4.15e); one on the lower chain and one on the outer layer (Figure 4.15f).

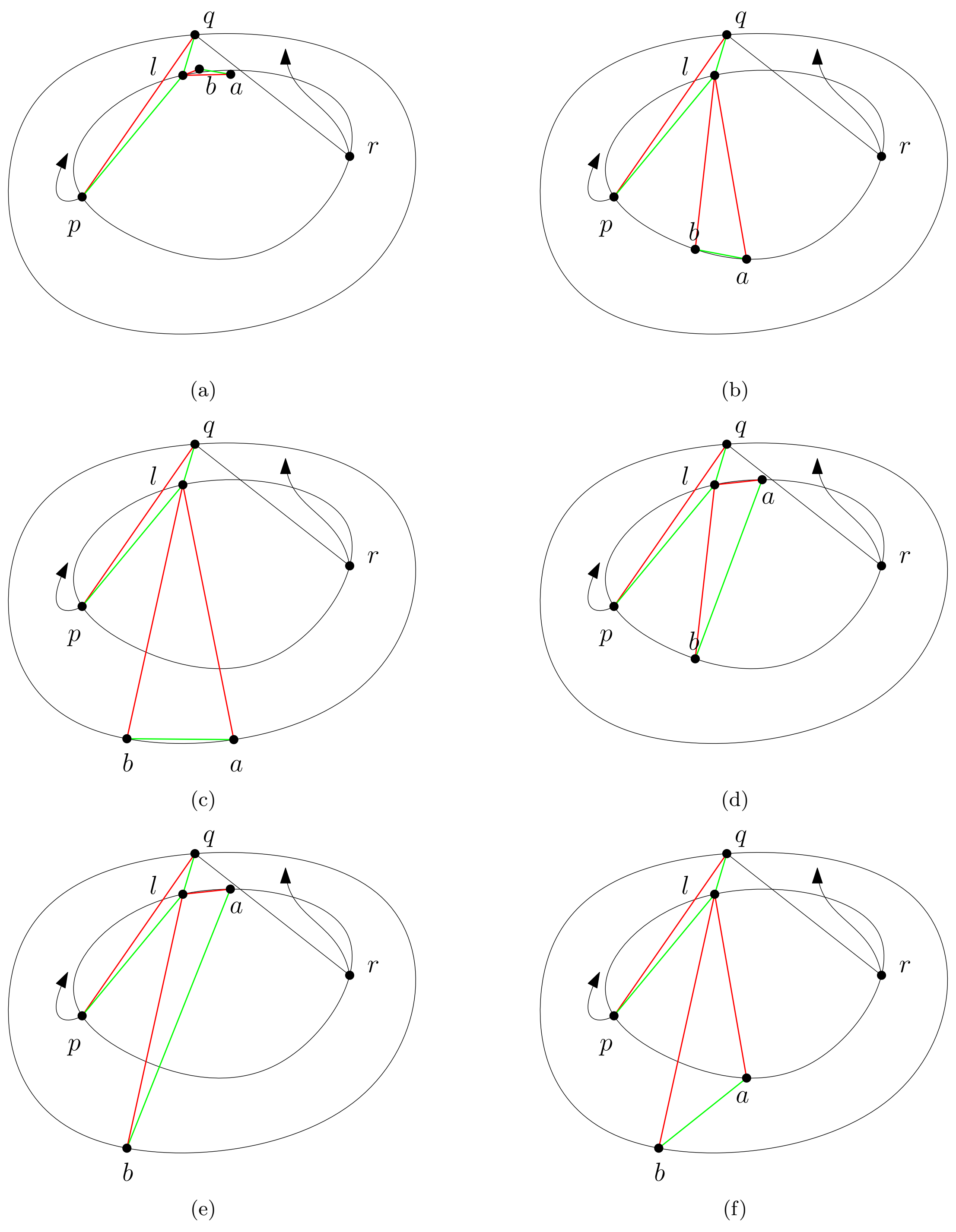
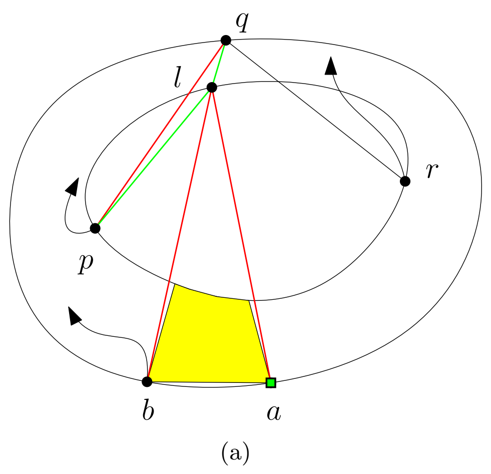


Figure 4.15: Case 3.

4. Points a and b share a path edge with l, edge ab is not feasible and a and b lie on the outer layer. There is at least one dual chain in this configuration, and there may be more if there are cutting connectors inside the angle alb. If there is at least one non-empty cutting connector triangle among these dual chains, that one should be resolved before the one we are now resolving. The subpath alb forces an endpoint, by a similar argument as the endpoint



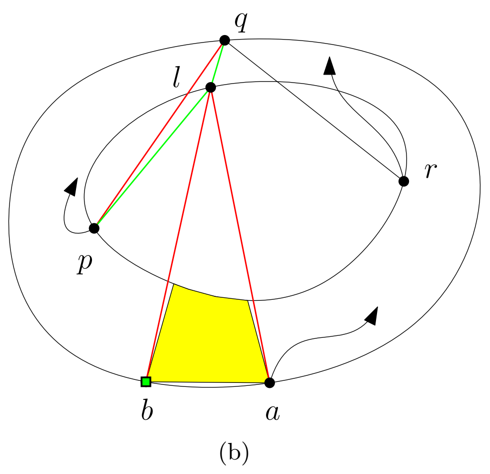
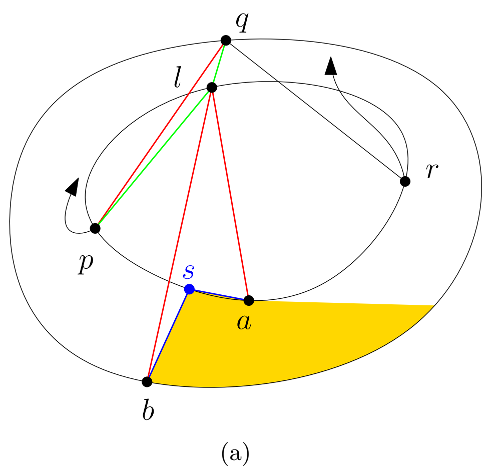


Figure 4.16: Case 4.

in a corner. There are two possibilities. The first one is when point b shares an edge with a point outside the angle alb (Figure 4.16a). Then a becomes the new endpoint and the new path will move along the dual chain(s) until b, from where it will follow the original path. The second possibility is when a shares an edge with a point outside the angle alb (Figure 4.16b). Then b will be the new endpoint, the new path will move along the dual chain(s) until a, from where it will follow the original path.



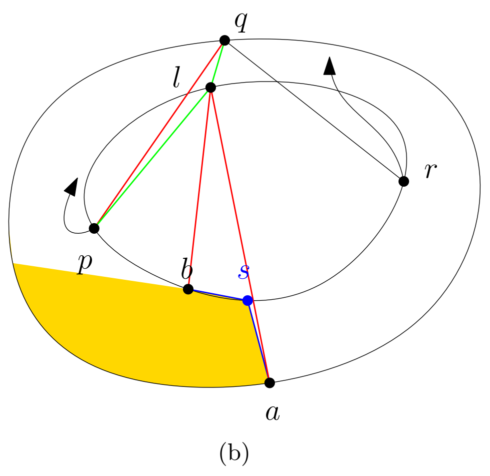


Figure 4.17: Two of the three instances of case 5.

5. Points a and b share a path edge with l, edge ab is not feasible and a and b do not lie both on the outer layer. There are three possible configurations: one of the points a and b has to be on the outer layer, but the other point can be on the lower chain (Figures 4.17a and 4.17b) or on the upper chain (Figure 4.18). It is not possible that both points lie on the inner layer, because the triangle abl would then be empty and any edge crossing ab would have no point to go to. This means that no edge crosses ab so it is feasible, but we know that ab is not feasible. Therefore, we know that there must be at least one point inside abl, which is only possible if a or b lies on the outer layer. As in case two, we need the geodesic inside abl, and because the points inside abl form a convex chain, the geodesic is either the left- or the rightmost point inside abl; this depends on the configuration. That point is called s. To reroute, we remove pq, al and bl, add lp and lq, and we recurse by stealing s with the thief

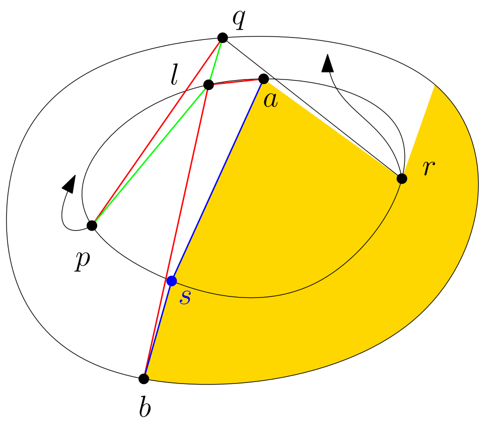


Figure 4.18: One of the three instances of case 5.

edges as and bs. The area of recursion is bounded, in case a or b lies on the lower chain, by a, s, b, and the furthest point the point on the lower chain can connect to without cutting. In case a lies on the upper chain, the area of recursion is bounded by b, s, a, all points to the right of a on the upper chain, r, and the furthest point r can connect to without cutting. Recursion can only happen on points in the lower chain, so the recursion always ends.

In recursion, we consider the different cases once more. Some cases are the same for both the initial situation and recursive steps, some are different.
Case 1 remains the same: if s is an endpoint, we remove the edge connected to s and add the thief edges.
Case 2, one of the thief edges is a path edge, is somewhat different. In the initial situation, only pl can be a path edge, because if the thief edge ql had been a path edge, this would give q a degree of 3 in the original path. In recursive steps however, either of both thief edges can be a path edge, but not both, since that would have been a cycle in the original path. This is the main difference between the initial and recursive situation; for the rest, it is the same. Without loss of generality, we assume that the thief edge as appears to be a path edge; a mirrored version of this argument holds when the other thief edge is a path edge. Let c be the point other than a that shares an edge with s. If ca is not feasible, we take the geodesic of the points inside the triangle csa, which is only one point s', and we recurse on that point with as' and cs' as thief edges. If ca is feasible, we add ca and the other thief edge and remove cs.
Case 3 remains the same: if c and d share an edge with s and cd is feasible, we remove cs and ds and we add cd and the thief edges as and bs.
Case 4 is not possible in recursion. Since s lies on the lower chain, two points c and d on the outer chain that share an edge with s would yield an empty triangle cds, so any path edge crossing cd would have no point to go to, which means that cd is feasible so this turns out to be case three, not four.
Finally, case 5 remains similar. If c and d share an edge with s and cd is not feasible, we take the geodesic of the points inside the triangle cds, which is only one point s', and recurse on that point with cs' and ds' as thief edges.

We now are able to construct a sequence of compatible paths between an arbitrary path and a canonical path. This means that all Hamiltonian paths on point sets of two convex layers are eventually compatible. With a local step, we can compatibly reroute paths with non-empty cutting connector triangles in such a way that they contain one less, eventually constructing a path without non-empty cutting connector triangles. If there are cutting diagonals in the path, the tail of such a path can be compatibly rerouted in a global step using dual chains, which defines a path with one cutting diagonal less, so eventually we can define a path without cutting diagonals. With an other global step using dual chains, we can compatibly construct a path without cutting connectors. Such a path is directly compatible to a canonical path. By providing this mechanism that compatibly turns any arbitrary path into any other, we have proven that all paths are eventually compatible to each other.

Chapter 5

Conclusions

Given a point set P on two convex layers, we define a graph G as having the set of all non-crossing Hamiltonian paths on P as its vertex set. Two points in G share an edge if the corresponding Hamiltonian paths are compatible, that is, their edges do not cross each other. We have proven that G is connected, by providing a mechanism to turn an arbitrary path into any other via compatible intermediate paths.

It remains an open problem whether G is connected if P consists of more than two convex layers. We think it is, and we think that our mechanism can be adapted to work for multiple convex layers. Convex layers are defined recursively, the first convex layer is defined as the convex hull of P, and the ith convex layer is defined as the convex hull of P after removal of the points in layers 1 to $i - 1$. Our mechanism might be adapted to work recursively, first constructing some sort of canonical subpath on layer 1, then on layer 1 and 2, and so on, until a canonical path on the entire point set is reached.

Bibliography

[1] O. Aichholzer, F. Aurenhammer, and F. Hurtado. Sequences of spanning trees and a fixed tree theorem. *Computational Geometry: Theory and Applications*, 21:3–20, 2002. 3

[2] O. Aichholzer, S. Bereg, A. Dumitrescu, A. García, C. Huemer, F. Hurtado, M. Kano, A. Márquez, D. Rappaport, S. Smorodinsky, D. Souvaine, J. Urrutia, and D. R. Wood. Compatible geometric matchings. *Computational Geometry: Theory and Applications*, 42:617–626, 2009. 3

[3] O. Aichholzer and K. Reinhardt. A quadratic distance bound on sliding between crossing-free spanning trees. *Computational Geometry: Theory and Applications*, 37:155–161, 2007. 3

[4] D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65:21–46, 1996. 2

[5] K. Buchin, A. Razen, T. Uno, and U. Wagner. Transforming spanning trees: A lower bound. *Computational Geometry: Theory and Applications*, 42:724–730, 2009. 3

[6] F Hurtado, M Noy, and J. Urrutia. Flipping edges in triangulations. *Discrete and Computational Geometry*, 22:333–346, 1999. 2

[7] C. Lawson. Transforming triangulations. *Discrete Mathematics*, 3:365–372, 1972. 2