UNIVERSITY OF CALIFORNIA

SANTA CRUZ

RUBBER-BAND BASED TOPOLOGICAL ROUTER

A dissertation submitted in partial satisfaction of
the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

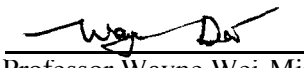
COMPUTER ENGINEERING

by

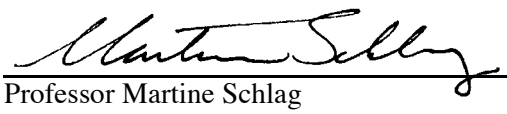
Tal Dayan

June 1997

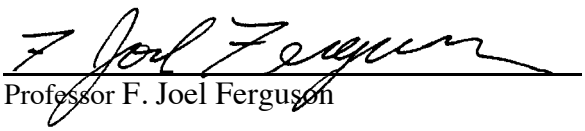
The dissertation of Tal Dayan is approved:



Professor Wayne Wei-Ming Dai, Chair



Professor Martine Schlag



Professor F. Joel Ferguson

Dean of Graduate Studies

ABSTRACT

A multi-layer, topological detailed-router is described. This is the first router ever reported that uses a rubber-band sketch (RBS) to represent the interconnect. The detailed-router is part of SURF, a routing system for multi-chip modules and VLSI that was designed to handle efficiently large multi-layer problems. The detailed-router supports various routing goals and can generate layouts for rectilinear, octilinear and any-angle wiring rules. It uses a novel approach of unconstrained layer-assignment that makes a better usage of the routing resources by considering a continuous metric of conflict between nets as opposed to the binary go/no-go approach. The layer-assignment is formulated as an optimization problem and various routing goals such as wire and via minimization or constrained-layers can be achieved by simple modifications to the cost function. The layer-assignment partitions the multi-layer problem into a set of single-layer sub-problems that are routed independently by a topological planar-router. The planar-router uses a new net-ordering algorithm that results in shorter wiring than the 'shortest first' approach. The nets are embedded sequentially using an optimal algorithm that finds a shortest planar path in the RBS. The generated RBS is then optimized by a simple re-route algorithm that takes advantage of the 'attachment' relation between branches and terminals in the RBS. A mathematical formulation of the concept of RBS is also presented and is used to prove the correctness of the shortest-path algorithm. This is the first exact analysis of RBS ever published. Empirical results are also shown and they demonstrate the merit of the router.

Dedicated to Nir

ACKNOWLEGMENT

I would like to thank the people that enabled and helped this research. First, to my advisor Professor Wayne Dai that his vision, guidance, and leadership was the driving force behind the SURF project. To all the students ('SURF'ers') that worked on various parts of SURF including (in alphabetical order): Joel Darnauer, Mark Fitzpatrick, Duane Foote, Maggie Zhiwei Kang, Kazuhiko Kobayashi, Raymond Kong, Michael Lanzetta, Yizhi Lu, Paul Morton, Paul Pham, David Prather, Darren Senn, David Staepelaere, Joseph Russack, Karen Wang, Jeffrey Su, Marco Man-Fai Yu, William Wu, and Qing Zhu.

Thanks to Professor Joel Ferguson for reviewing this dissertation. Thanks to Professor Martine Schlag for her thorough comments and suggestions regarding contents, organization, style, and spelling. Thanks for Caroline for here sincere notes. Special thanks to David Staepelaere for his enlightening comments on the layer-assignment and the net-ordering chapters, and for his overview of SURF that was the base for SURF overview included in this dissertation. Thanks to Jeffrey Su for his implementation of the 'region' module and for supporting the rubber-band 'engine' used by the local router. Thanks to George 'Joe 19' French for proof-reading the introduction, overview, and the net-embedding chapters. Thank the people at Frame Technology for making available Frame Maker, a great writing tool. And last but not least, I would like to thank Teiko for her support and her help in proof-reading this paper.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF GRAPHS

LIST OF TABLES

1 INTRODUCTION

1.1 General

Currently available routers are running out of steam when it comes to meeting the challenges presented by today's VLSI designs and packaging technologies. Multi-Chip Modules (MCM) [1] for example may consist of 60 or more layers and a large number of terminals and nets, and therefore the router must be able to handle large designs efficiently in both time and space. SURF is a routing system developed at University of California Santa Cruz that was designed to answer these challenges. The subject of this research is a rubber-band based topological local router that was developed as part of the SURF project. SURF is the first rubber-band based router ever reported and it has many advantages compared to existing geometrical routers. Its flexible representation of the interconnect enables it to achieve a high completion rate. It handles large designs efficiently and the rubber-band representation of the interconnect provides a powerful environment for manual editing.

1.2 Organization of This Thesis

In the rest of this chapter we present an overview of previous works related to our research and a discussion of our original contribution. In Chapter 2 we present an overview of topological representation of interconnect and an overview of SURF. Chapters 3, 4, 5 and 6 cover four aspects of our local router: layer-assignment, net ordering, topological net embedding, and the re- route optimizer respectively. Chapter 5 also includes a formulation of the rubber-band sketch. The formulation is based on mathematical terms from geometry and real analysis. The chapters of this thesis were designed to be independent of each other such that a reader can have a good understanding of each one without reading the previous ones. As a result, some degree of redundancy does exist, especially in the introduction of each chapter where the general terms and context are defined. Finally, Chapter 7 contains a conclusion and a discussion of future research.

1.3 Previous Works

A common method of handling large designs is to perform the routing in two phases, *global* and *local* routing. The global routing partitions the routing problem into a set of smaller sub-problems

which are then solved independently by the local router. Each of the sub-problems can be either a *channel* with the terminals on its boundary, or can be *channelless* and have terminals and features inside the routing area as well. Channel routing is more appropriate for Standard or Macro Cells where the routing is done in the spaces between the components, while channelless routing fits better MCM and modern VLSI designs. This dissertation describes a multi-layer, channelless, local router that uses rubber-band topological representation of the interconnect.

The known methods of routing can be classified into the two main categories of *geometric* and *topological* routing. Geometric routers determine the exact geometrical paths of the wires while topological routers use a more abstract representation of the interconnect and determine only the topology of the wires but not their exact paths. Geometric routing is by far the most common routing method used today. Some of the geometric routers mentioned in the literature are maze routers [51] [32] [19] that represent the wires as paths along a two or three dimensional grid, line-probe routers [21] [50] that use a gridless representation, and SLICE [29] which is an efficient plane-sweep router.

Since topological representation is more abstract than geometrical representation, the interconnect is easier to manipulate, and a topological router can avoid making too detailed decisions in early stages of the routing. When the topological routing is completed, it is checked to comply with the spacing requirements and is then transformed to exact geometrical layout.

Various aspects of topological routing have been discussed in the literature including routability check, topological representation, compaction and dynamic updating. The following is a discussion of previous works related to these areas.

The first problem related to topological routing mentioned in the literature is that of *routability test* defined in [54] [56]. This problem is also called *Detailed Routing given an Homotopy* (DRH) and is defined as follows: given a specification of the terminals and modules, and given a homotopy (a rough planar sketch of the wires), is there a geometrical layout that conforms to the homotopy, spacing requirements, and wiring rules? Cole and Siegal [5] proposed a polynomial-time algorithm that finds a complying geometric layout, and showed that such a layout exists if and

only if every *cut* is safe. Leiserson and Maley [35] [44] showed that it is sufficient to check a smaller set of *critical* cuts, and that the test can be done on the *Rubber-Band Equivalent* (RBE) in which wires are as short as possible and zero spacing is allowed (while maintaining the topology of the wires). Kong [31] used the idea of RBE and the concept of *spokes,* introduced by Leiserson and Maley [35], to perform a constructive incremental routability check that results in a complying geometric layout, if one exists. The works of Leiserson and Maley [35][44], and of Kong [31], assume no constraint on the wiring model. Maley, in a later work [46], showed that in the special case of polygonal grid-based wiring rules, the routability test can be performed in $O(n \log n)$ time.

The topological representations mentioned in the literature fall roughly into two categories: Triangulation Crossing Sketches (TCS) and Rubber-Band Sketches (RBS). TCS use a triangulation of the routing area and capture the crossing of triangulation edges by nets' wires. RBS on the other hand represents the interconnect as flexible rubber-bands that bend and stretch to form the shortest possible connection of a given topology. The RBS representation was originally proposed by Rivest (see acknowledgment in [35]). Leiserson and Maley formalized this concept and called it the *Rubber-Band Equivalent* (RBE) [35] [44] [45]. As far as we are aware of, the only rubber-band based router ever reported is SURF (Santa Cruz ULSI Routing Framework) [9] that is the framework of this research. SURF is based on ideas and results presented by Leiserson and Maley [35] [44] and it covers various aspects of rubber-band based topological routing including: global routing [63], multi-layer topological local routing (this research), representation and manipulation of RBS [8], optimization of an RBS (ROAR optimization, part of this research), testing for routability in an RBS [10] [31], and transformation of an RBS to geometric layout [62]. An overview of SURF is presented later.

The flexible nature of topological representation makes it easier to move features while maintaining the connectivity and the topology of the wires. Murata and Kajitani [52] [53], and Su [65] proposed algorithms for moving objects in TSC and RBS respectively. The flexibility of the topological sketches has also been used to solve the *homotopic compaction* problem. That is, to find the smallest (area wise) layout that conforms to given homotopy, wire rules, and spacing

requirements. Maley [43] proposed a one dimensional compaction algorithm. His work was later extended by [16] [45] [49] [66] [39] [13] and [25].

Several approaches for routing of multi-layer designs were discussed in the literature, including 3-dimensional (3D) search, *maximal-layer* routing, and *layer-assignment*. The 3D search is an extension of single-layer search algorithms such as maze-runner and line-probing that considers layer crossings. In the maximal-layer approach, which is used for example by SLICE [29], the layers are routed sequentially, trying to fit on each layer as much as possible of the missing interconnect. The layer-assignment approach (survey in [26]) considers simultaneously all nets and layers, and assigns nets or partial nets to the available layers.

Known methods for layer-assignment are usually classified as *constrained* or *unconstrained* (also called *topological*). In a constrained layer-assignment, the geometric paths of the wires have already been determined prior to the assignment and the assignment step only determines on what layers they will be routed. In an unconstrained layer-assignment on the other hand, the actual wire paths are not specified to the layer-assignment. In this research we adopted the unconstrained approach because it does not commit in early stages of the routing to specific wire routes, and therefore it is more likely to take advantage of the flexibility of the underlying topological representation.

The concept of unconstrained layer-assignment was introduced by Hsu [24] and since then has received a fair amount of attention in the literature. Hsu [24] and Marek-Sadowska [47] dealt with two-layer problems where all the terminals are on the boundary of the routing area and the via count is to be minimized, while guaranteeing planarity on each layer. This work was extended by Haruyzama, Wong, and Fussell [20] to support multi-terminal nets. Sarrafzadeh and Lee [59] showed that the via minimization problem is equivalent to the *maximum 2-independent set* problem in a circle graph which is NP-complete. Pinter [54] proposed an algorithm that supports terminals inside the routing area. It is based on a partitioning of the routing area into disjoint regions called *clusters* such that the wires in each cluster can be routed on two layers without crossing each other (i.e. each layer is planar). The algorithm then sub-optimally reduces the

problem into finding of a maximal cut in a planar graph, and this problem has a polynomial time solution. The advances in VLSI and MCM which may utilize a significantly larger number of layers resulted in published works that deal with multi-layers. Kiao, Lee, and Sarrafzadeh [36] presented a polynomial algorithm for finding a maximal-weighted planar subset of multi-terminal nets in a switchbox. This algorithm can be used to sequentially 'peel' maximal sets of nets for each layer, assuming that all the terminals are on the boundary. If terminals do exist within the routing area, the algorithm has exponential complexity. Other works that restricted terminals to exist on the boundary include [6], [64], and [25]. Cho at al [4] proposed a layer-assignment algorithm that handles terminals within the routing area and considers cross-talk, via count, and wire-length. This algorithm which was developed independently, uses a net pair-wise cost function similar to the one developed in our research (first published in [11]). His algorithm however, assigns complete nets to layers and does not consider insertion of vias as our algorithm does.

Our layer-assignment algorithm (called LAA) is to our best knowledge, the first successful attempt to perform topological layer-assignment with simultaneous minimization of via-count and wire-length. It scales well with an increasing number of layers, it supports multi-terminal nets, and it handles terminals inside the routing area. The LAA models the layer-assignment as an optimization problem with a cost function that considers via-count, and wire length. The cost function can be extended to consider other routing goals such as preferred layers, and cross-talk. The LAA supports rectilinear, octilinear, and any-angle wiring models, and it takes advantage of the flexibility of the underlying RBS.

The LAA decomposes the multi-layer routing problem into a set of independent single-layer sub-problems. Each single-layer problem specifies the terminals (which may include vias introduced by the layer-assignment) and nets or partial nets to be connected on that layer. The nets of each single-layer problem are guaranteed by the LAA to be planar (i.e. they can be topologically routed with no intersections). This kind of routing problem is often referred to as *single-layer* or *planar* routing. Known methods for single-layer routing fall roughly into two categories of *sequential* and *simultaneous* routing. In the sequential approach, nets are routed one-at-a-time, typically on a least-cost path, considering the previously routed nets. The least-cost path can be determined for

example by a maze-runner or a line-probing algorithm. A draw-back of sequential routing is its sensitivity to the order in which the nets are routed. In some cases for example, the optimal solution cannot be achieved by *any* order. Liao and Sarrafzadeh [38] addressed this problem with an algorithm that applies the concept of global routing. It performs the least-cost path search on a rough grid such that it makes decisions with a more global view. In simultaneous routing on the other hand, all nets are routed at the same time and therefore no net gets higher priority than others. This approach is used for example in SLICE [29] where it performs a single-layer routing in a single plan sweep. SLICE's approach is insensitive to the net order but is less likely to find more complex paths and to efficiently handle 'keep-out' areas due to its one directional sweep and short look-ahead.

In this research, we have focused on sequential routing for two reasons. First, the planar routing in SURF is done in a topological sketch and therefore is less sensitive to net ordering. Second, the planar routing is performed on relatively small regions called *bins* defined by the global routing and therefore, any 'bad' decision made while determining the path of a net will have only local and limited affect. Our router uses a net ordering algorithm that estimates the final wire-length for a given order of nets, and a rerouting post-processor that searches for an alternative topology with shorter wiring. The ordering and the planar routing algorithms considers the wire length (Euclidean, rectilinear or octilinear) and are guaranteed to find a planar solution.

Another known concept which is used by our single-layer router is *rip-up and reroute* (RUR). RUR is typically used to improve the completion rate and is performed by deleting some nets from the sketch, routing a residual net, and then rerouting the deleted nets. A variant of this approach, that can be used by routers that are able to represent violations such as cross-over, is to reduce the number of conflicts between already routed nets. The main difficulty in performing RUR is to determine which nets to delete and in what order to reroute them. Several works including [58] [61] [40] [7] [27] and [57] address this issue. The RUR approach is frequently used in sequential routers to compensate for the sensitivity to net ordering. This problem is less significant in topological routing. For example, the Puzzle Problem mentioned in [27] requires a RUR approach to be solved optimally by a geometrical router, but is easily solved optimally using a topological

router because the wires are flexible and are pushed away to create clearance for new wires (Figure 1). Our router uses a RUR optimizer called ROAR that accepts a planar RBS and is guaranteed to result in a planar RBS with the same or lower wire length. To determine which wires to remove and in what order to reroute them, the algorithm uses the *attachment* relation between branches and terminals in the RBS. The ROAR optimizer is described later in this thesis.

1.4 Contribution of this work

Our local router is the first ever published router that is based on rubber-band topological representation. By taking advantage of the flexibility of the RBS, it can efficiently achieve high quality layouts and to handle a wide range of routing problems. Our local router preforms the routing in four steps: layer-assignment, net-ordering, net-embedding, and optimization. Our contribution in each of these areas is outlined below.

Our unconstrained layer-assignment algorithm (LAA) uses a new approach in which the output of the layer-assignment is unconstrained as well, that is, it does not specify the actual paths of the wires but only what sub-nets are to be connected on what layers. This approach leaves more freedom to the single-layer router and enables it to make more informed decisions. The layer-assignment is formulated as an optimization problem with a cost function to be minimized. The cost is based on an estimation of the final wire-length and supports user specification of the desired balance between the conflicting goals of minimizing wire-length and via-count. The cost function uses a continuous metric of the conflict between nets as opposed to the go-no-go approach previously used. This enables the LAA to better utilize the routing resources by allowing nets with low conflict to be assigned to the same layer. The cost function can be extended to support other goals such as cross-talk minimization and constrained-layers. The cost function has a finite value if and only if the assignment is planar and this enables the LAA to guarantee that the nets or partial nets assigned to each layer can be routed on that layer without crossing each other. The LAA handles multi-terminal nets and can break a net into multiple layers if this results in a more desirable solution. The LAA handles terminals on the boundary and within the routing area and the terminals are not restricted to be on a grid. If terminals do exist on the boundary, the LAA can consider specification of layer-assignment done previously on the other side of the boundary. This

makes the LAA suitable for routing systems like SURF that perform global routing before the layer-assignment. The LAA is not restricted to a one-layer-one-direction routing style and supports Euclidean, rectilinear and octilinear wiring rules. It uses an algorithm (called 2NAA) that optimally assigns a single two-terminal net, given a sequence of candidate locations for vias and the assignments of previously assigned nets. We have compared our router to SLICE and the experimental results show that it achieves better layouts, even when it is restricted to rectilinear wiring rules.

Similar to the layer-assignment, our net-ordering algorithm is based on a formulation of net ordering as an optimization problem with cost function to be minimized. The cost function estimates the wire-length based on a pair-wise conflict between nets. The planarity of the order is handled by the Planarity Operator that, given a net ordering, finds an order of the same cost that is guaranteed to be routed with no net crossings. Experimental results show that our net ordering algorithm reduces the wire length by 5% and 30% in average compared to the 'shorter-net-first' and 'longer-net-first' approaches respectively.

Our net-embedding algorithm performs a shortest path search in the RBS and is the first rubber-band shortest path algorithm ever reported. The search algorithm is guaranteed to find a shortest planar path in $O((T^2 + S)\log(T + S))$ time where T and S are the number of terminals and number of wire segments respectively in the RBS. The algorithm can be modified to use a smaller search graph and to find a planar path that is likely to be short in $O((T + S)\log(T + S))$ time. The shortest-path algorithm is also used in our rip-up and reroute algorithm (called ROAR) that is used to compensate for dependency of the generated layout on the net order. The ROAR algorithm uses the 'attachment' relationship between wires and terminals that is unique to the RBS.

We also present a formulation that maps the concept of RBS into known mathematical terms from geometric and real analysis. This clarifies the definition of the RBS as being both planar and of zero spacing. The formulation is used to prove the correctness of the shortest planar path algorithm using exact mathematical tools. To our best knowledge, this is the first time an exact analysis of RBS is published.

2 OVERVIEW

In this chapter we present an overview of topological representation of the interconnect and an overview of the SURF routing system. This introduces the terms commonly used in topological routing and explains the context in which the proposed local-router is used.

2.1 Topological Representation and Routing

A topological sketch represents the embedding of nets in an abstract form. It contains the general relationship of the nets and terminals but ignores details such as the exact geometrical paths of the nets. This 'partial' representation avoids making too detailed decisions in early stages of the routing (Figure 1) and provides greater flexibility in manipulating the sketch. When the topological routing is completed, it is then transformed into a geometrical layout with exact location of the wires.

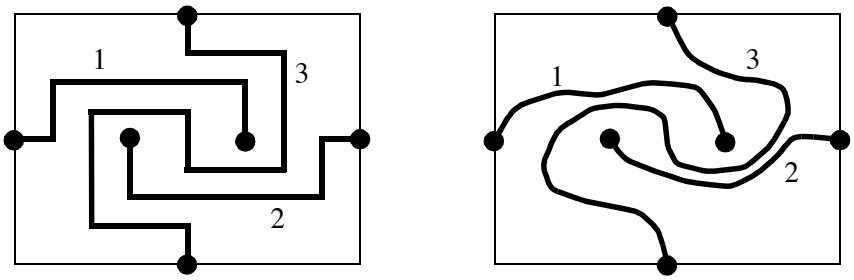


FIGURE 1 - The Puzzle problem. (a) shows an optimal solution to the Puzzle example proposed in [27]. The example has three nets that are routed in the order 1, 2, 3. A typical geometrical router that routes the nets sequentially on a shortest path will fail routing net 3 because there is not enough clearance between the previously routed nets 1, 2 and the two internal terminals. A topological router on the other hand (b) can easily insert net 3 since no commitment was done on the exact geometrical paths of nets 1, 2.

In the rest of this section we present an overview of topological representation of the interconnect. The discussion is limited to single-layer sketches. Multi-layer sketches can be constructed using a set of single-layer sketches and some specification of the layer crossings between terminals of

adjacent layers. For clarity, we deal here with a simplified model of the routing that includes only terminals and nets. Obstacle or 'keep-out' areas can be handled in a similar way.

2.1.1 Sketches and Topological Classes

A *geometric sketch* (*sketch* in short) represents the exact paths of the interconnect. It includes a set of *terminals*, each has a unique point inside or on the boundary of the routing area, and *branches,* each a connection between a pair of terminals. The terminals and the branches have zero width and they represent the centers of the pads and the center-lines of the wires respectively. A maximal set of terminals interconnected by branches is called a *net* or *component*. The branches and terminals of a component are assumed to have no cycles (i.e. the components are trees). A branch is represented by a path connecting its two end-terminals. A branch path is a parametric line $P(t) = (x_t, y_t)$, where $t \in [0..1]$, and (x_t, y_t) is a point on the plane, such that the two end-points of the path, $P(0)$, $P(1)$ are the locations of the end-terminals of the branch. The decision which of the two end-points of a branch $P(0)$ represents is arbitrary. A branch path is planar, it does not cross itself, terminals, or other paths, except for possibly at its end-points $P(0)$, $P(1)$, and is strictly contained inside the routing area. The function $P(t)$ is continuous and piece-wise differentiable.

Let A, B be two sketches with identical sets of terminals and same branch connectivity (but not necessarily the same paths), and let $<A_i(t)>$, $<B_i(t)>$ be the vectors of branch-paths of A, B respectively. We say that sketch A is *homotopic* [35] to sketch B, or that A 'has the same topology' as B, if there is a continuous transformation from the paths of A to the paths of B, $T(r) = <P_i(r, t)>$, $r \in [0..1]$, such that $P_i(0, t) = A_i(t)$, $P_i(1, t) = B_i(t)$, $P_i(r, 0) = A_i(0)$, $P_i(r, 1) = A_i(1)$, and all the intermediate sketches, when $0 < r < 1$, are planar. Intuitively this represents a continuous co-deformation of the branches of sketch A, to achieve those of sketch B, while maintaining their end-points and their relative locations (Figure 2).

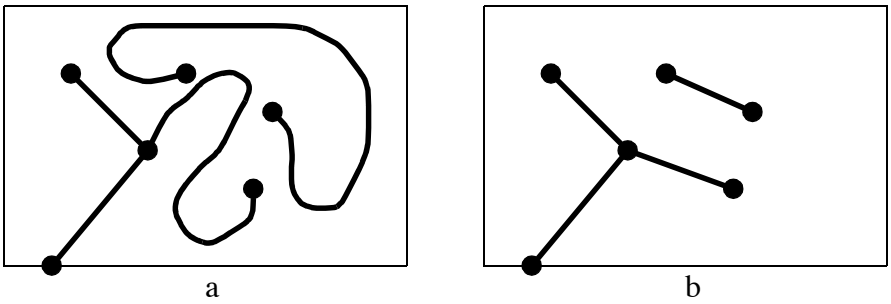


FIGURE 2 - Homotopic sketches. The two sketches are homotopic since they have the same sets of terminals, their branches represent the same connectivity, and one can be continuously transformed to the other while preserving the planarity and the end-points of the branches

The homotopic relation between sketches is reflexive, symmetrical, and transitive, and therefore it partitions the set of all possible sketches (of same terminal set and branch connectivity) into equivalence classes called *topology classes* (*classes* in short).

The wire length of a given sketch is defined as the sum the lengths of its branches. A topology class does not necessarily have a sketch of minimal length. That is because shorter wiring requires smaller spacing but when the spacing reaches zero, the sketch is non-planar and therefore it is not a member of the class, nor is it a valid sketch (Figure 3). Each class however has a upper lower bound (ULB) L on its wire length. That is, for every class, there is $L \geq 0$ such that the wire length of all the sketches in the class $\geq L$, and for every $\varepsilon > 0$, there is a sketch in the class whose wire length $< L + \varepsilon$. Informally, the neighborhood of sketches whose wire length are at the infinitesimally small neighborhood of L is called the *kernel* of the class. The kernel of a class is significant because it contains the more desirable solutions with shorter wires.

A *topological sketch* represents a subset of a topology class. It can represent a single sketch, the entire class, or a subset of it. It is possible to have the agreement that a topological sketch which represents *directly* a subset of a class, also represents *indirectly* the rest of the sketches in that class. In an extreme example, a geometrical sketch can be used to represent its entire class. This sketch represents directly only a single member of the class (itself) and represents indirectly the

rest of the class. In this discussion, unless explicitly specified otherwise, we will consider only direct representation.

a b

FIGURE 3 - Minimal length sketch. The topology class that contains the sketch in (a) does not have a sketch of minimal length. To further reduce the length of the sketch in (a) , the spacing between the branch and the terminal need to be reduced. However when the spacing reaches zero, the sketch looses its planarity (b) and therefore it is not a member of the class, nor it is a valid sketch.

2.1.2 Triangulation Crossing Sketch (TCS)

A *pure* topological sketch represents directly all the sketches of the class. We are not aware of any pure topological representations proposed in the literature, and it is even questionable if such representation would be useful for routing. An example of a topological representation, which is not pure, is the Triangulation Crossing Sketch (TCS) that represents directly only a subset of the topology class. The TCS uses an arbitrary triangulation of the terminals, and for each triangle, it specifies the connectivity of the nets crossing the triangle edges and of the terminals on its corners (Figure 4). The TCS does not include exact specification of the locations of the edge crossings and it indicates only their relative order along the triangulation edges. The TCS represents the subset of the topology class that includes sketches having the same pattern of edge crossing and the same connectivity in each triangle. A TCS represents its class with no ambiguity but is not unique. A topology class may have more than one TCS representing it, even for a given triangulation. A useful canonical version of the TCS is when the total number of crossings is minimized. This representation, called Triangulation Crossings Normalized Sketch (TCNS), is unique for a given topology class and triangulation. The TCNS is useful because it is more ‘focused’ on the kernel of

the topology class and the set of sketches it represents include the sketches whose wiring lengths arbitrarily close to the upper lower bound.

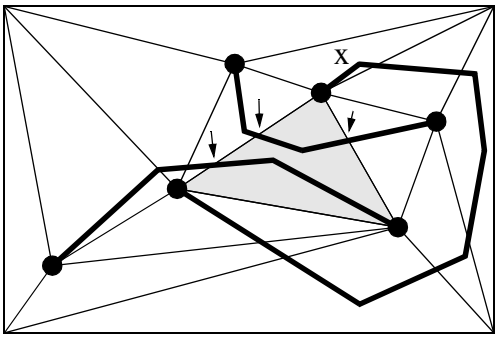


FIGURE 4 - The Triangulation Crossings Sketch (TCS). This topological representation captures the crossings of the triangulation edges by the branches (for a given triangulation of the terminals). The TCS represents the connectivity of the paths within each triangle without specifying exact locations of the edge crossings or the paths inside the triangle. The shaded area indicates a typical triangle and the arrows indicate the net crossings of the edges of this triangle. This sketch is not a TCNS because the number of crossings can be reduced (at the location marked by 'x') while preserving the topology.

2.1.3 Rubber-Band Sketch (RBS)

An alternative representation which is even more 'focused' on the kernel then the TCNS is the *Rubber-Band Sketch* (RBS). Informally, RBS represents the shortest possible wiring of a topology class. That is, when wires are as short as possible, zero spacing is allowed, and the topology is preserved (Figure 5). If the class has a member of minimal length ~~than~~ then its RBS is exactly that member. Otherwise, the RBS represents the kernel of the class but not any specific member of it. Later in this thesis, we present a formal definition of the RBS and show how it can be used to prove properties of the RBS. Similar to the TCNS every class has a unique RBS representing it. The RBS is more specific than the TCNS in the sense that it contains more geometrical information. The RBS provides better estimation than the TCNS of the properties of the final layout such as wire length, signals delay, and cross-talk.

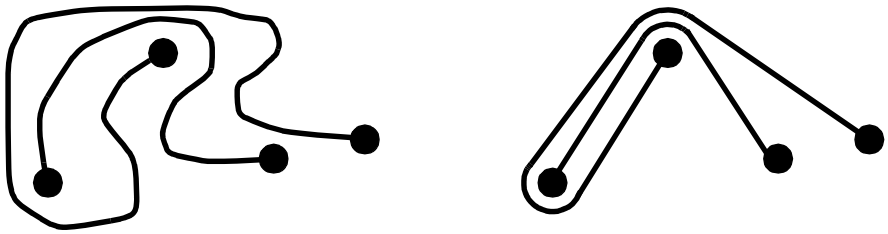


FIGURE 5 - Rubber-Band Sketch. (a) shows an arbitrary member of a topology class and (b) shows the RBS of that class.

2.1.4 Routability Tests

A topology class is said to be *routable* for given spacing requirements if it has a member sketch that satisfies those requirements. Leiserson and Maley [35] [44] proposed a necessary and sufficient condition for a topology class to be routable. The condition is based on the notion of *cuts* and their *capacity* and *flow*. A cut is a line segment connecting two terminals that does not cross any other terminal. The capacity of a cut is the distance between its end-terminals (using the metric appropriate for each design), and its flow is the sum of the spaces needed to satisfy the spacing requirements along the cut. This includes the widths of the branches crossing the cuts, the radius of the end-terminals, and the required spaces separating the branches and the end-terminals. A cut is said to be *overflowed* if its flow is greater than its capacity. Note that the set of cuts and their capacities is independent of the sketch or topology class while the flow is specific to a given sketch. By Leiserson and Maley [35] [44] a topology class is routable if and only if it has a member sketch that none of its cuts is overflowed. Note that the criteria holds for the topology class and not for a specific sketch, a sketch can has no overflowed cuts and yet not satisfies the spacing requirements. Leiserson and Maley also showed that a class is routable if and only if the sketches of its kernel (the term Rubber-Band Equivalent was used there) are routable, and this again shows the significance of the kernel.

An alternative approach for routability test that is used by SURF [31] is to extend the RBS using *spokes* [35]. Spokes (Figure 6) are artificial terminals which are added to the RBS to enforce the

desire spacing. The addition of spokes modifies the RBS such that the new RBS, which is called *extended rubber-band Sketch* (ERBS), has a super set of the terminals of the original RBS. The branch paths in the ERBS satisfy the spacing requirements if, and only if, the topological class represented by the RBS is routable. Furthermore, the paths of the branches in the ERBS, represent a minimal length member of the topological class of the RBS that satisfies the spacing requirements. This member represents, in the case of Euclidean metric routing, the final geometrical embedding. In case of rectilinear or octilinear routing, a conversion step called *geometrical transformation* [62] is required to find the minimal length member of the class that complies with the wiring rules. The advantage of testing for routability using spokes is that this method is constructive and it results in a sketch that satisfies the spacing requirements, while the flow/capacity criteria merely indicates if such a sketch exits.

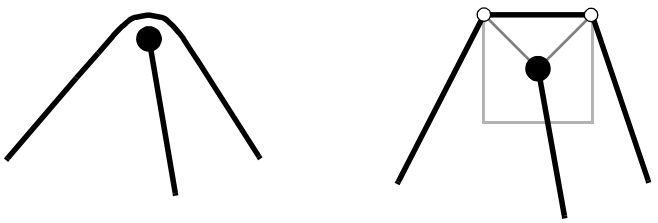
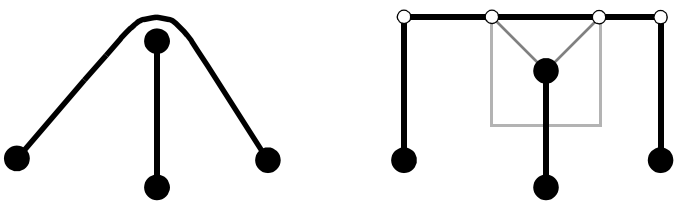


FIGURE 6 - Spokes and ERBS. The RBS at (a) has zero spacing between the branches. To achieve the required spacing, the RBS is extended into an ERBS (b) by the introduction of *spokes*. The paths of the branches in (b) are also the paths of a minimal-length sketch in the topology class of (a) that has the required spacing. The spokes in this example are for rectilinear metric.

The extended ERBS is an example how an RBS can also be used to represent arbitrary piece-wise linear geometrical sketches by the addition of artificial terminals along the branches. Sketches of this kind do not contain attachments of branches to terminals and they represent exactly a single member of their topology classes. This technique can be used for example to represent rectilinear geometrical sketches (Figure 7). This demonstrates the flexibility of the RBS and its support for step-wise refinement routing.



a b

FIGURE 7 - Rectilinear ERBS. This example shows how an RBS can be used to represent a geometrical piece-wise linear sketch. The RBS in (a) is extended to (b) by the addition of artificial terminals to have the require spacing and rectilinear wiring model. This representation of the geometrical sketch provides flexibility in manipulating the sketch.

2.1.5 Topological Routing

The problem of topological routing can be formulated as follows: given net-list and spacing requirements, find a topology class with the connectivity specified by the net-list, that has a member of minimal wiring length that satisfies the required spacing. A solution to the topological routing problem can be specified in various ways including the forms of topological sketches such as TCS, TCNS, or RBS, or by an arbitrary geometric sketch of that class. Since there are known efficient methods to convert between all those forms, any of the these representations is as good as the others.

Given a topology class that has a member with the required spacing, the final result of the embedding problem is a member of that class that satisfies the spacing requirements and has minimal wiring length. Such a sketch can be found mechanically, given its topology class, for example by extending the RBS with spokes. The conversion from a topological class to a member of it with shortest wiring length that satisfies the spacing requirements has optimal polynomial-time algorithms. Therefore, the main difficulty in routing a design is finding the topology class of the solution. This observation supports the approach of topological routing because it focuses on the core problem and ignores geometrical details that can be determined later mechanically.

A *topological* router is an algorithm that solves the topological routing problem. However, since every router (such as a maze router which is usually considered to be a *geometrical* router) solves the topological routing problem, we usually use this term only for routers that use topological representation and avoid making exact geometrical decisions at early stages of the routing. SURF for example is based on a topological router that uses RBS representation in the intermediate stages of the routing.

2.2 SURF Routing System

2.2.1 Introduction

SURF is a routing system developed at University of California, Santa Cruz that uses the topological local router presented in this thesis. SURF was designed to address some of the challenges of routing thin-film MCM substrates, including handling of large problems efficiently, conforming to a variety of non-traditional wiring geometries, supporting performance and production-cost constraints, and providing powerful manual editing in a gridless environment. In order to meet these needs, SURF adopts new routing strategies based on an efficient rubber-band representation. Although it is designed primarily for routing MCM, the same approach can easily be applied to other area-routing problems such as sea of gates, printed circuit boards, and VLSI.

Because existing routing systems generate precise geometry for nets one at a time, they lack the flexibility and global view required to solve the problems mentioned above. The main strategy of SURF is to transform the net-list to geometrical layout by a series of refinements. At each step, the solution becomes more precise—more information is acquired and it more closely resembles the final result. Because the earlier stages are not swamped with unnecessary information about precise routing geometries, they can address more global concerns. This also makes it easier to correct mistakes at earlier stages as more detailed information is discovered. Another aspect of the SURF strategy is to allow localized overflow regions or constraint violations during the refinement process. These are then used to direct the refinement of the layout until a final correct design is reached.

SURF models the interconnect using a *rubber-band sketch* (RBS) (Figure 8a) for each layer. The end-points of the rubber-bands may represent terminals, vias, or junction points. If an end-point is moved, the wires automatically stretch and move as would an elastic rubber-band. This rubber-band model also supports efficient and precise routability and design-rule checking.

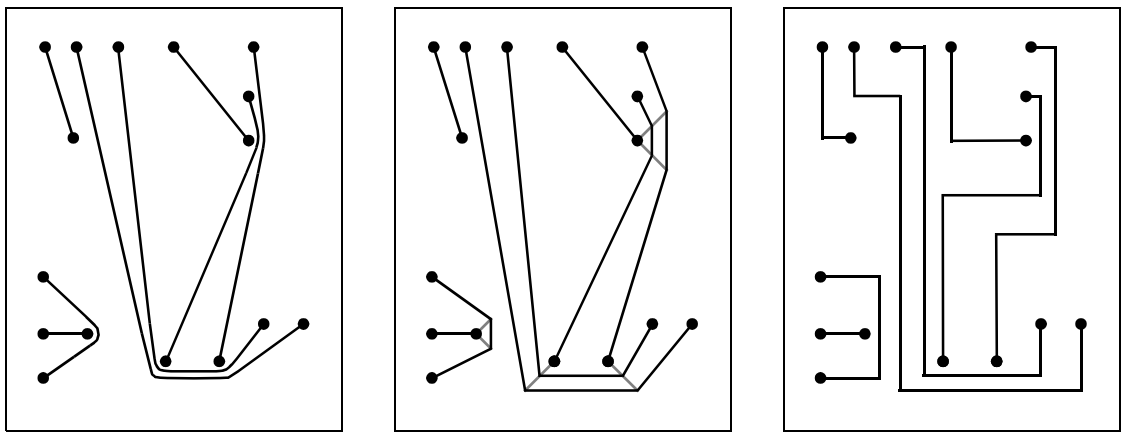


FIGURE 8 - Rubber-Band routing example. This figure shows various wiring patterns that are topologically equivalent. These patterns correspond to different stages of the transformation from rubber-band sketch to precise geometric layout. Figure (a) shows a rubber-band sketch, (b) shows the extended rubber-band sketch after spoke creation, and (c) shows the rectilinear wiring.

To improve the efficiency of rubber-band updating, each layer of the sketch is built on top of a triangular mesh. This mesh is maintained using an incremental constrained Delaunay triangulation algorithm. In addition to providing incremental triangulation modifications, it supports efficient geometrical queries such as point location, nearest neighbor, and range search [41][42]. Since the size of the triangulation is linear in the number of terminals in the sketch, this data structure is more space efficient than a traditional grid-graph.

2.2.2 Layout Refinement Strategy

The SURF routing approach is divided into two major steps (Figure 9a): (1) *topological routing* that transforms the net-list into a topological wire description in a form of a multi-layer RBS, and

(2) *geometrical transformation* that converts the RBS into precise geometrical layout [9][10][62][63].

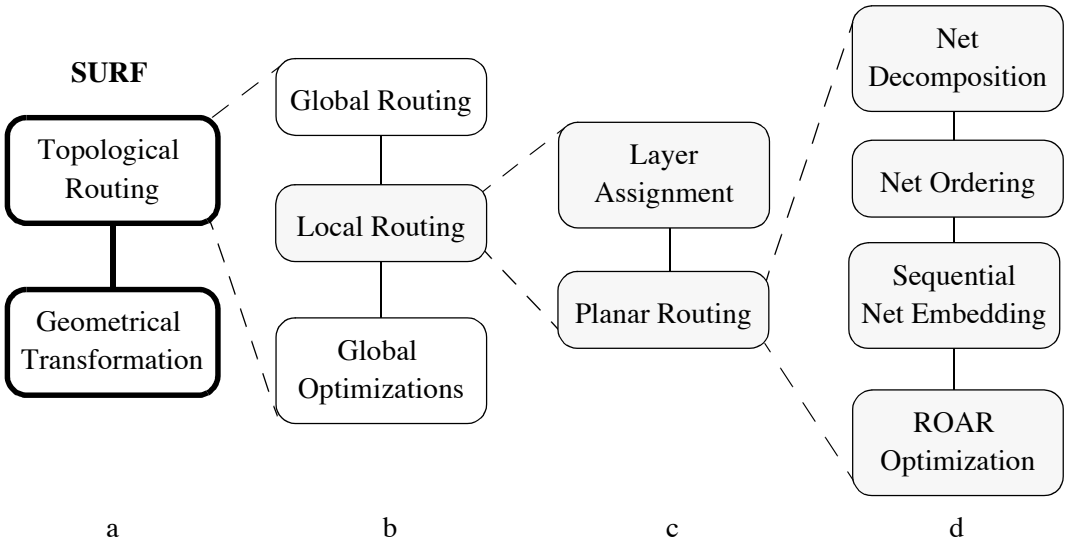


FIGURE 9 - SURF routing strategy. This figure shows a hierarchical decomposition of SURF routing stages with focus on the local-router presented in this thesis. The shaded blocks represent routing steps covered by this research. At the top level (a) SURF has two routing stages, the topological routing which generates a multi-layer RBS and the geometrical transformation that converts the RBS to a geometrical layout. The topological routing (b) is done in three steps of global-routing, local-routing, and global-optimizations. The local router (c) routes each bin independently. This is done in two steps of layer-assignment and then a planar routing of each bin-layer. The planar routing is done in four steps (d) of decomposing the nets into single-layer point-to-point connections (called *2-Nets* of *branches*), ordering the 2-Nets, sequential routing of the 2-Nets, and optimizing the sketch using the ROAR optimizer.

The RBS is generated in three steps (Figure 9b) of global routing, local routing, and global optimizations. The global router (Figure 10) uses hierarchical partitioning to divide the original problem into a set of smaller sub-problems [37][48][9]. The purpose of the global routing is to produce an initial rough routing that loosely specifies the route of each net. When processing a partition, the global router analyzes all nets simultaneously. As a result, it does not suffer from the ordering problems of sequential net routing. Also, because it uses a top-down divide-and-conquer approach, the global routing is produced relatively quickly. Once the global router has partitioned the original problem, the multi-layer topological local routing is performed on each of the sub-

problems and when the local routing of each sub-problem is done, solutions are merged to form a global RBS. The RBS is then optimized for various goals such as reducing wire-length, and improving the production yield (even wire distribution).

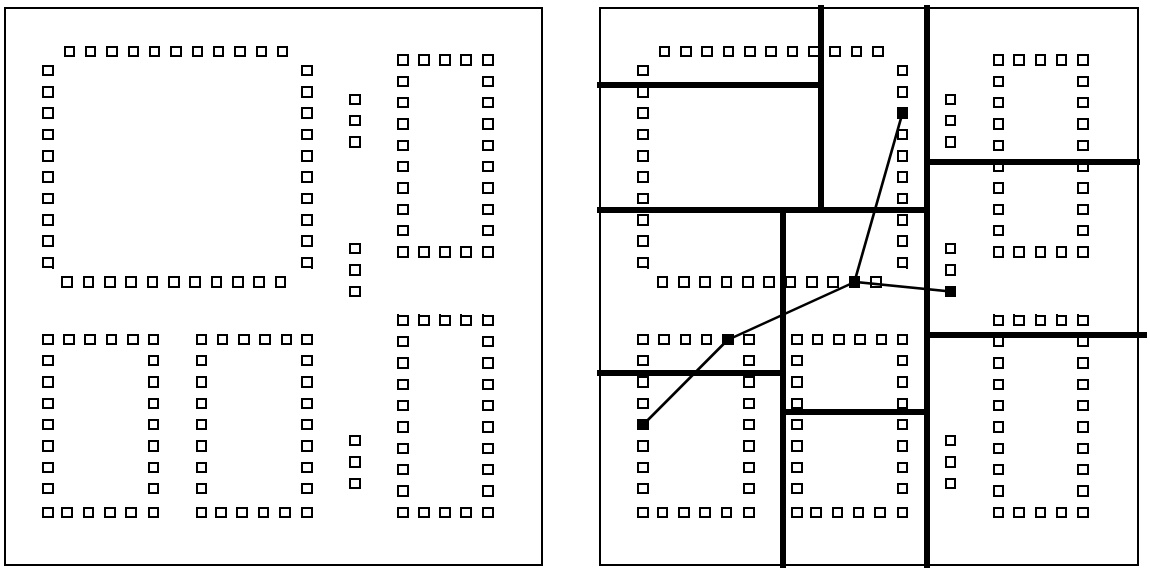


FIGURE 10 - Global routing. The global routing step partitions the routing problem into *bins* and finds rough routes for each net. A typical net is shown. Note that the global router determines the location at which the nets cross the bin boundaries but not their paths or topologies within the individual bins.

The topological multi-layer local router (which is the subject of this research) routes the bins generated by the global assignment. Each bin represents a multi-layer routing problem of a relatively small area of the design. The local routing is done in two steps (Figure 9b) of layer-assignment and then a planar-routing of each bin-layer.

The layer-assignment phase takes the net-list for a single multi-layer sub-problem (a *bin*) and, by adding vias, partitions it into a set of single-layer routing problems called *bin-layers*, one for each layer. This layer-assignment is *unconstrained* since it occurs before any detail or local routing is done. This allows for much greater flexibility. In constrained layer-assignment, when two wires cross each other, they are treated as 'conflicting' and must be put on different layers. Since SURF

performs layer-assignment prior to the creation of actual rubber-band paths, it can treat conflict as a continuous value instead of a boolean one (a 'go/no-go' approach). The amount of conflict between two nets is an estimate of the detour length required to route those two nets on the same layer. SURF uses a cost function that balances detour against via cost to perform an iterative layer-assignment process. This process starts with an initial assignment and then uses steepest descent to reach a final solution. At each step, the connection that allows the largest cost reduction is reassigned.

When layer-assignment of a bin is completed, a planar routing of each of the bin layers is performed. This is done (Figure 9d) by decomposing the nets of each bin layer into single-layer point-to-point connections (called *2-Nets* of *branches*), ordering the 2-Nets, and then routing them sequentially within the bin layer. At this stage, the difficulties of sequential routing are significantly reduced. There are two reasons for this. First, by determining a rough routing and breaking the original problem into bins of relatively few nets, the global router has reduced the magnitude of the problem. Second, because rubber-bands are flexible, a later branch can easily be inserted between two previous branches without the need for explicit shoving or rerouting. When the sequential routing is done, each bin-layer is optimized by the ROAR optimizer to further reduce the total wire length.

After the rubber-band sketch is created by the topological router, it is transformed to a topologically equivalent form that maintains the proper spacings and obeys the proper geometry. An efficient process called *spoke creation* checks the routability of a sketch and satisfies the width and spacing constraints [35][31]. This process pushes wires away from fixed objects with open-ended line segments called *spokes* (Figure 8b). The number of spokes required at each point is related to the final geometry (Figure 11). Rectilinear wiring uses up to four spokes and octilinear uses up to eight. If no spacing violations are detected, the result of the spoke creation phase is a legal arbitrary-angle routing called an *extended* rubber-band sketch (ERBS).

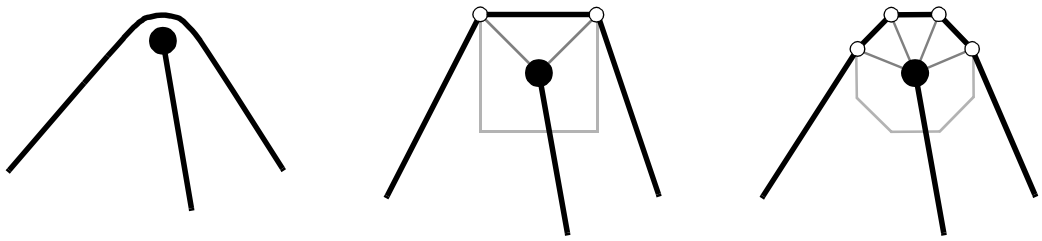


FIGURE 11 - Spokes. The spokes are used to enforce the required spacing. (a) shows a terminal in the RBS and (b) and (c) show the same terminal with rectilinear and octilinear spokes respectively. In case of any-angle routing, SURF uses the octilinear spokes as an approximation to the arc required for pure Euclidean routing. This enables to produce the layout with standard manufacturing equipment.

The extended rubber-band sketch has a shorter wire length and fewer jogs than the corresponding rectilinear or octilinear routing. However, if other CAD tools or the fabrication process requires a more restricted wiring pattern, an extended rubber-band sketch can easily be converted to either rectilinear (Figure 8c) or octilinear wiring. This conversion can be done with two plane sweeps. The first sweep transforms the sketch to the proper geometry and the second removes unnecessary jogs [62].

2.2.3 Routing Examples

In this section we present a few SURF generated layouts that demonstrate its capabilities. Because of space limitations, only relatively small examples are shown. SURF, however, has been used to route much larger designs with tens of thousands of nets.

Figure 12 shows an octilinear routing of a two-layer mixed-signal MCM. Figures 13, 14 and 15 show three layouts of the same MCM layer using rectilinear, octilinear, and Euclidean wiring rules respectively. These three examples were intentionally routed to have the same layer-assignment and wire topology and to differ only in the wiring rules. In the general case however, changing the wiring rules affects the layer-assignment and wire topology because it affects the cost function of

the assignment and the distance metric used by the shortest-path algorithm. Finally, Figure 16 shows a layer of a larger MCM with octilinear wires.

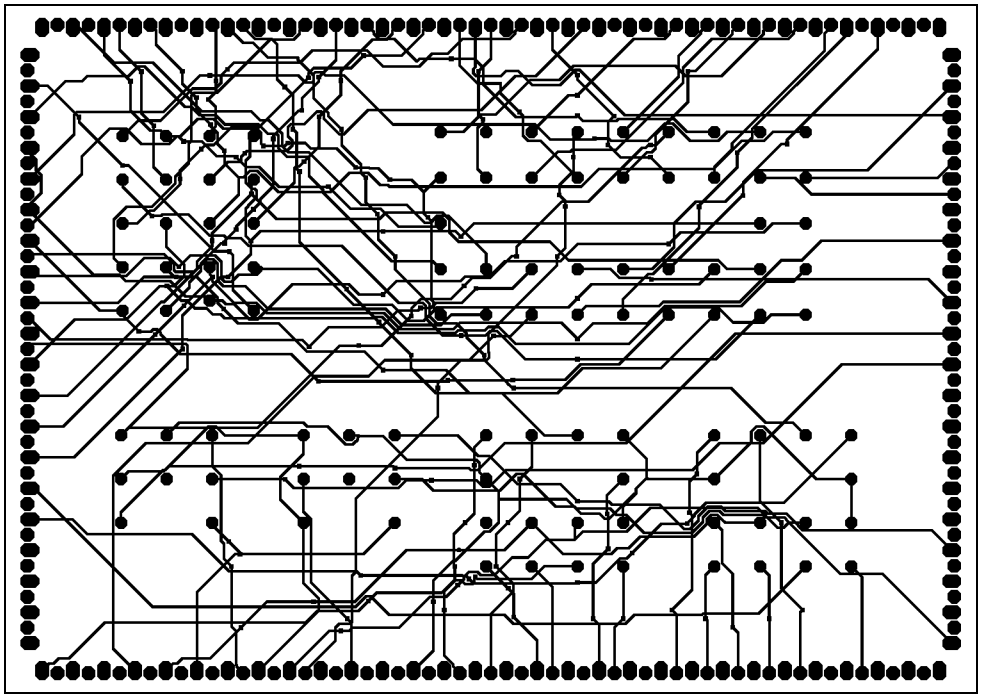


FIGURE 12 - Mixed Analog/Digital. This figure shows an octilinear layout of a two-layer mixed analog/digital consumer product.

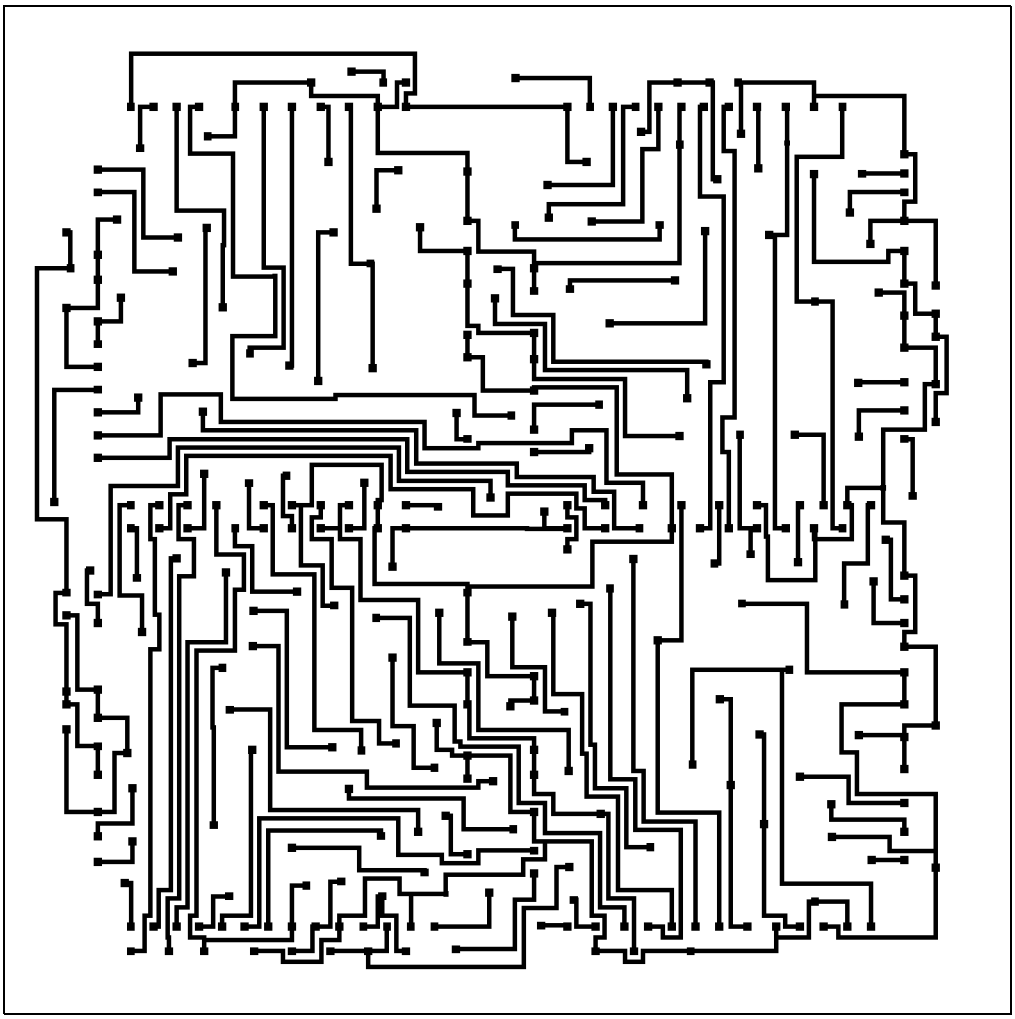


FIGURE 13 - Rectilinear Routing. Rectilinear (Manhattan) layout of an MCM layer. The same layer is shown in Figure 14 and Figure 15 with octilinear and any-angle wires respectively.

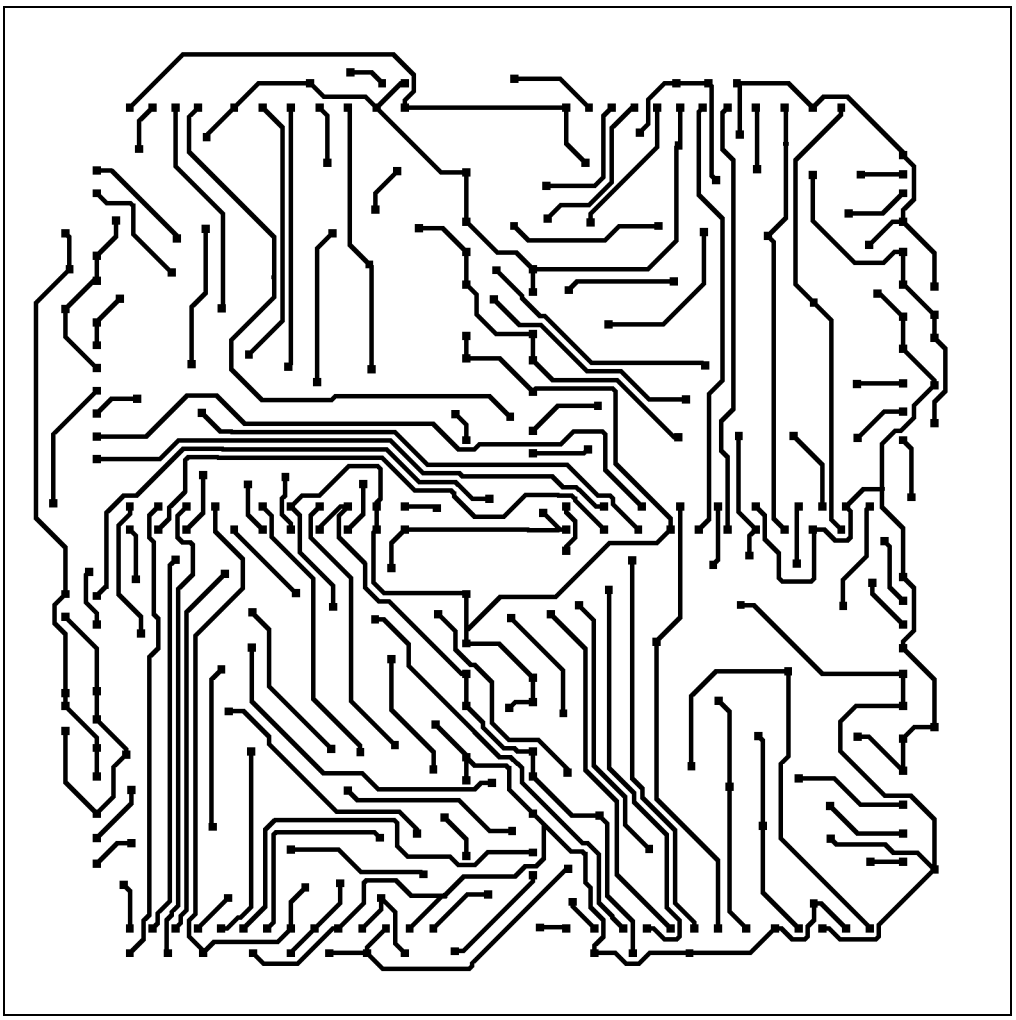


FIGURE 14 - Octilinear Routing. This is the same layer shown in Figure 13 but with octilinear wires.

FIGURE 15 - Euclidean routing. This example shows the same layer in Figure 13 but with euclidean (any-angle) routing.

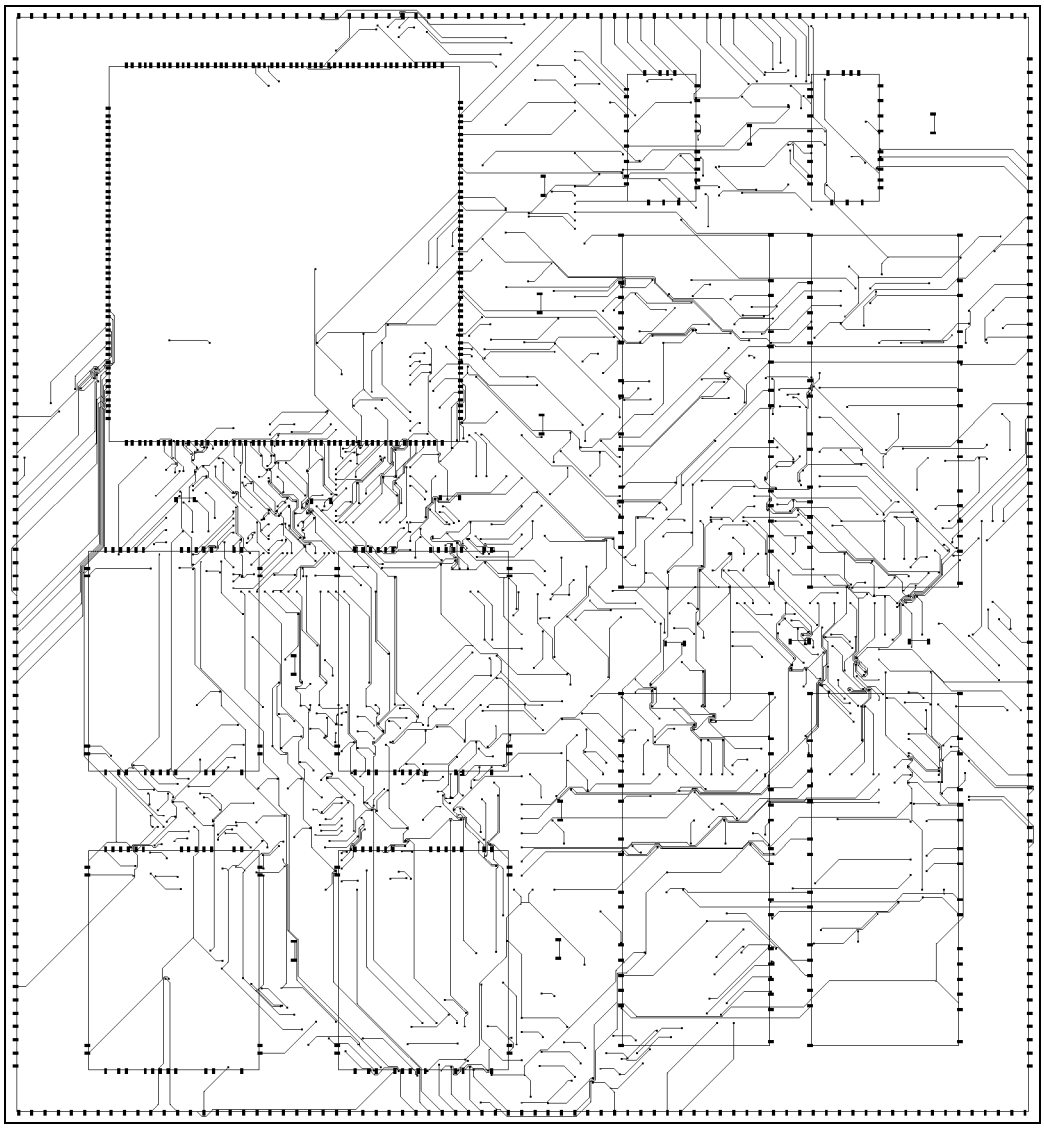


FIGURE 16 - Larger routing example. This figure shows a layer of a larger MCM routed with octilinear wiring rules.

3 TOPOLOGICAL LAYER-ASSIGNMENT

3.1 Introduction

The first routing step our topological local router performs is the layer-assignment. The layer-assignment accepts a bin routing problem, which includes the placement of the terminals, and the net-list inside the bin, and generates a set of single layer sub-problems, one for each layer. These sub-problems are then solved independently by later steps of the local router and merged back to form the final solution. Figure 17 shows the relationship between the design, the bins, and the single layer sub-problems.

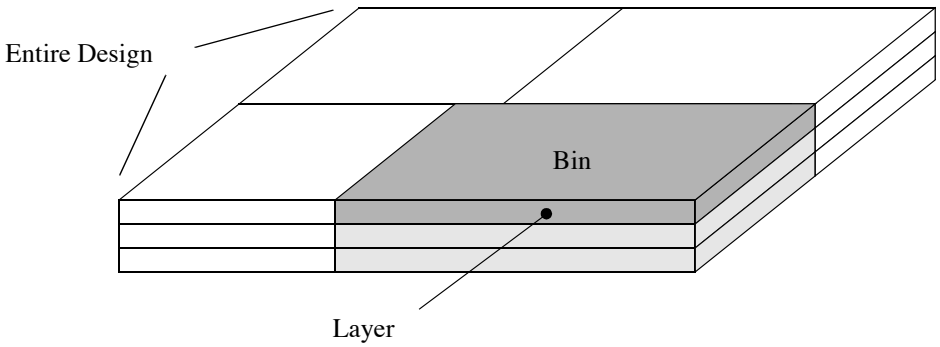


FIGURE 17 - The two level decomposition of a routing problem. First, the global router partitions the routing area into bins and then the layer-assignment decomposes each bin into a set of single-layer routing problems.

Figure 18 shows an example of a layer-assignment and routing of a bin. The set of input terminals the layer-assignment accepts includes both terminals specified in the design input and *cross-points* defined by the global router on the bin border. Each cross-point specifies the place where a net crosses from one bin into another. The set of cross-points on a boundary between two bins represents the *interface* between the two bins. The layer-assignment partitions the input nets into single-layer sub-nets called *components* which are the input nets of the single-layer routing problems. When doing so, the layer-assignment may introduce layer crossings called *vias,* which define the interfaces between single layer sub-problems on adjacent layers.

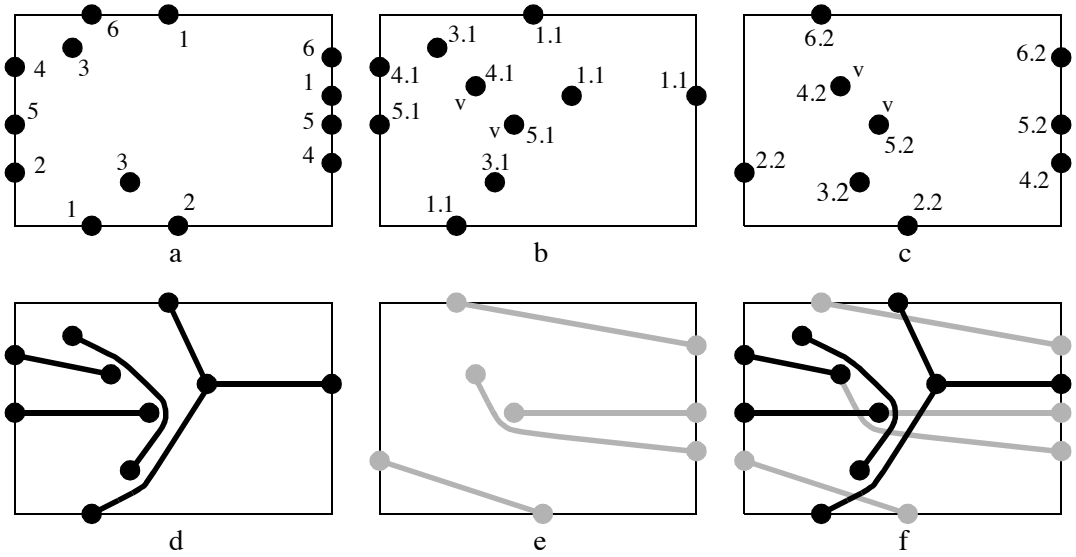


FIGURE 18 - An example of a bin layer-assignment and routing. The net list in (a) is decomposed by the layer-assignment into two single-layer routing problems (b) and (c) respectively. These sub-problems are then routed as shown in (d) and (e) respectively and then merged together to form the final two layer routing (f). The six nets in (a) are decomposed by the layer-assignment into eight components, four on each layer. The assignment defines two vias ('V') and a single Steiner point ('S').

The layer-assignment step does not determine the exact geometrical locations or even the topologies of the traces—it leaves these decisions to later stages of the routing. However, it must guarantee that each layer of the assignment it produces can be routed in a planar fashion. In addition, it should choose an assignment that allows a final routing that has few vias, and short wiring length[1].

The rest of this chapter describes the details of the layer-assignment. First the Layer-Assignment Problem is formulated as an optimization problem, then an algorithm is presented which solves the problem, and finally experimental results of automatic routing are presented.

1. The layer-assignment can be extended to consider other goals such as the 'one and a half layer' routing presented later in this thesis.

3.2 The Layer-Assignment Problem

The Layer-Assignment Problem (LAP) is formulated as a minimization problem with a simple and useful objective function, which can be computed efficiently. A minimization problem is defined by three components, an *input domain* of *problem instances*, a possibly empty *output domain* of *feasible solutions*, and a *cost function* to be minimized. The next three sections defines these three components of the LAP.

3.2.1 The Input Domain

Definition 1 (LAP Instance) An instance P of LAP is specified by the 5-tuple $\langle B, n, \dot{T}, \dot{E}, \dot{N} \rangle$, where:

- B is a rectangle on the routing plane representing the bin routing area.
- $n \geq 2$, is the number of available layers which defines a set of layers $L = \{l_1 .. l_n\}$. The layers are oriented such that layer l_i is said to be *above* layer l_j if $i > j$.
- $\dot{T} = \{\dot{t}_i\}$ is a set of input terminals, where each terminal $\dot{t}_i = \langle x, y \rangle$ is specified by its location in the routing area B. $\dot{T}$ includes the actual terminals of the design as well as the cross-points defined by the global router over the bin boundary.
- $\dot{E}:(\dot{t}_i) \rightarrow 2^L$ is a function which maps each input terminal $\dot{t}_i$ to a possibly empty set of layers on which the terminal is said to *exist* prior to the assignment. For terminals, it is the set of layers on which the terminal is *required* to have pads. For cross-points it is the set of layers to which the cross-point has already been assigned in previously routed bins. $\dot{E}(\dot{t}_i)$ is contiguous, that is, if it includes layers l_i and l_j, $i < j$, then it includes every layer l_k, $i < k < j$ in between the two. This information of pre-existing layers is used by the layer-assignment to reduce the number of vias.
- $\dot{N} = \{\dot{n}_i\}$ is the input *net-list*. It is a partition of the input terminals $\dot{T}$ into subsets called *nets* of terminals to be interconnected.

The input of an LAP does not include any specification of wire widths, spacing requirements, or dimensions of the terminals. The layer-assignment considers only planarity and leaves the handling of congestion and routability to later steps of the routing process. This can lead to

situations where a routed layer has design rules violations. This case is analogous to a less than 100% completion-rate with a geometric router. In both cases, the designer is required to complete or rectify the routing manually.

3.2.2 The Solution Domain

Definition 2 (Feasible Solution) A feasible solution S to P is specified by $S = \langle \hat{T}, \hat{E}, \hat{C} \rangle$, where:

- $\hat{T} = \{\hat{t}_i\}$ is a set of output terminals where each output terminal $\hat{t}_i = \langle x, y \rangle$ is specified by a unique location in the routing area B. $\hat{T}$ is a super set of $\dot{T}$, it includes all the input terminals as well as any vias introduced by the layer-assignment.
- $\hat{E}:\hat{T} \rightarrow 2^L$ is a function[1] which maps each output terminal $\hat{t}$ to a set of layers on which the terminal exists after the assignment. An output terminal exists on a layer if it is connected on that layer, if it crosses that layer or if it represents an input terminal that already exists on that layer. Similar to $\dot{E}$, the set $\hat{E}(\hat{t})$ is contiguous.
- $\hat{C} = \{\hat{c}_i\}$ is a set of output *net-components* (*components* in short), each of which is a set of terminals to be interconnected on a specific layer. A component $\hat{c}_i = \langle t, l \rangle$ is specified by $t \subseteq \hat{T}$, the set of terminals to be connected and l, the layer on which to connect the terminals. The output components defines the net-lists for the individual single layer sub-problems.

To be considered *feasible*, a solution S needs to satisfy the following conditions:

- If an output terminal $\hat{T}_i$ represents an input terminal $\dot{T}_j$ then it exists on all the layers on which the input terminal exists $\dot{E}(\dot{T}_j) \subseteq \hat{E}(\hat{T}_i)$.
- Any pair of output terminals which represent a pair of input terminals is *to be connected* (defined below) if and only if both input terminals belong to the same input net. Informally, a pair of output terminals is said *to be connected* if the solution S specifies that they should be connected directly or indirectly by a sequence of vias and traces. Formally we define it recursively. A pair

1. This function can be implicit as it can be computed from $\hat{T}$, and $\hat{C}$.

of output terminals is considered *to be directly connected* if the two output terminals belong to the same component, or if they are vias at the same location (on different layers). A pair $\{a, b\}$ of output terminals is said *to be connected* if it is directly connected or if there exists a third output terminal c, such that both pairs $\{a, c\}$ and $\{b, c\}$ are connected.

- If a component connects an output terminal t on layer l then t should exists on layer l, $l \in \hat{E}(t)$.
- Finally, the output components can be topologically routed inside the bin area without crossing each other. This is the *planarity* property (discussed below).

A solution is planar if all the components on each layer can be routed entirely on that layer, inside the bin area, without crossing any two. Note that violations of this requirement can only occur if a component connects two or more points on the boundary of the routing area thereby partitioning the routing area into two or more disconnected regions (Figure 19). The layer-assignment is not required to find a planar routing but just to guarantee that the such a routing exists for its solution.

A solution to a LAP specifies the terminals and the components on each layer. It does not specify however the exact geometry or topology of the connections. These decisions are determined by later stages of the routing.

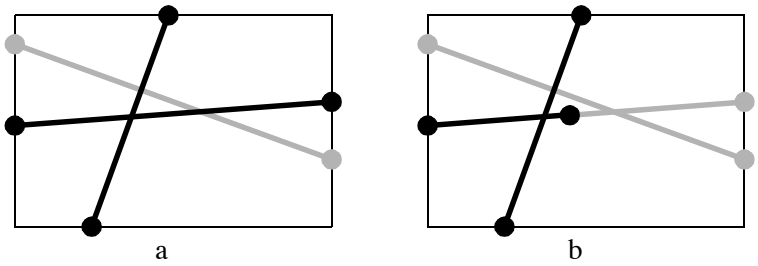


FIGURE 19 - An example of non-planar layer-assignment. (a) and (b) represent two solutions for the same layer-assignment problem. The black and gray lines represents a pair of points to be interconnected on different layers. Solution (a) is not feasible as it is not planar. The components specified by it can not be routed inside the bin without crossing each other. Solution (b) which uses an extra via is planar and thus feasible.

3.2.3 The Cost Function

The cost function C maps a feasible solution of P to a non-negative real number that indicates the relative cost of the solution such that more desirable solutions have lower cost.

Ideally, C should directly reflect the properties we prefer to have in the final layout. These properties can vary widely, depending on the design requirements. In this thesis we focus on two goals: minimizing the number of vias and minimizing the total wire length[1]. By making simple modifications to the cost function we can address other goals such as preferred layers or "one and a half" routing as presented later in this thesis.

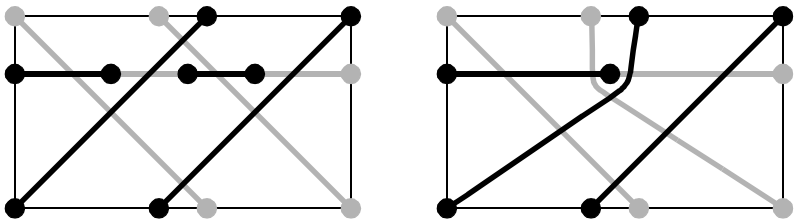


FIGURE 20 - Minimizing the via count vs. minimizing wire length. Both (a) and (b) are solutions of the same routing problem. Solution (a) minimizes the Euclidean wire length but has three vias. Solution (b) minimizes the number of vias but results in longer wiring.

The two goals are inherently conflicting as reducing the number of vias may require longer wires to detour around other nets (Figure 20). So the cost function is defined as a weighted balance between the two. If S is a feasible solution of P, the cost $C(S)$ is formulated as follows:

$$C(S) = \alpha V(S) + (1 - \alpha) W(S) \tag{1}$$

where:

- $V(S)$ is the total number of vias in S

1. Unless specified otherwise, we assume a Euclidean metric for wire length. Modifications for rectilinear and octilinear wiring rules are discussed later in this thesis.

- $W(S)$ is an estimate of the total wire length of S (explained below).
- $0 \leq \alpha \leq 1$ is a user controlled parameter which indicates the relative importance of the two goals for the routed design. The setting of this parameter depends on the requirements of the specific routed design and is done based on the experience of the user.

The value $V(S)$ is well defined and can be calculated precisely from S. $W(S)$ on the other hand requires a more precise definition since the actual wire length depends on the router and S does not imply any specific routing. Logical solutions for $W(S)$ might seem to be either the wire length obtained by routing S with either a perfect router (that guarantees an optimal solution) or with the actual router used in the subsequent single-layer routing phase. However, if C needs to be computed efficiently during the layer-assignment process, such a choice would be impractical. To overcome this problem, we define $W(S)$ as an *estimation* function which can be computed efficiently and is still closely related to the actual length. Our choice of estimation function is validated by experimental results presented later in this thesis.

Let c be a component in assignment S and $\bar{c}$ its embedding in some routing $\bar{S}$ of S. We define three values related to the length of c, the *basic length*, the *actual length* and the *detour length* (Figure 21):

Definition 3 (Basic Length) The basic length $W_0(c)$ is the length of the minimum Steiner tree of the terminals of c. This is minimum wire length among all the possible interconnects of the c.

Definition 4 (Actual Length) The actual length $\overline{W}(c)$ is the length of $\bar{c}$.

Definition 5 (Detour Length) The detour length $\overline{D}(c) = \overline{W}(c) - W_0(c)$, is the extra length of the traces of $\bar{c}$ beyond the basic length of c.

Note that the basic length can be computed solely from S, while the actual and the detour lengths depend on the choice of the actual routing $\bar{S}$. These three length metrics are also applied to individual input nets and to the entire net list.

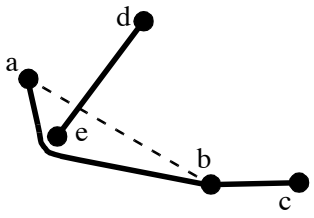


FIGURE 21 - The basic, actual, and detour lengths of routed components. The basic length of the component $\{a, b, c\}$ is $|(a, b)| + |(b, c)|$. Its actual length is the distance the length of its traces and its detour is the difference between its actual and basic lengths which is the difference between the length of the trace $\{a, b\}$ and the distance $|(a, b)|$.

Using these metrics, the actual wire length of $\bar{S}$ can be represented as:

$$\overline{W}(\bar{S}) = W_0(S) + \overline{D}(\bar{S}) \tag{2}$$

where $W_0(S)$ and $\overline{D}(\bar{S})$ are the sums of the *basic* and the *detour* lengths respectively of the components of S.

This form suggests a similar representation for the estimation of the wire length of S when S is to be routed with an unspecified router:

$$W(S) = W_0(S) + D(S) \tag{3}$$

Where $W_0(S)$ is the precise value of the basic length of the components in S and $D(S)$ is some estimation of the detour.

The detour estimation $D(S)$ we use is based on the concept of *conflict* between pairs of components assigned to the same layer. Figure 22 shows three examples of pairs of components. In the first case we say that they have no conflict because they can be co-routed such that the lengths of their wires equal their basic lengths. In the second and third examples, the components have low and high conflicts respectively. The concept of conflict provides a *continuous* metric which allows components with low conflict to be placed on the same layer.

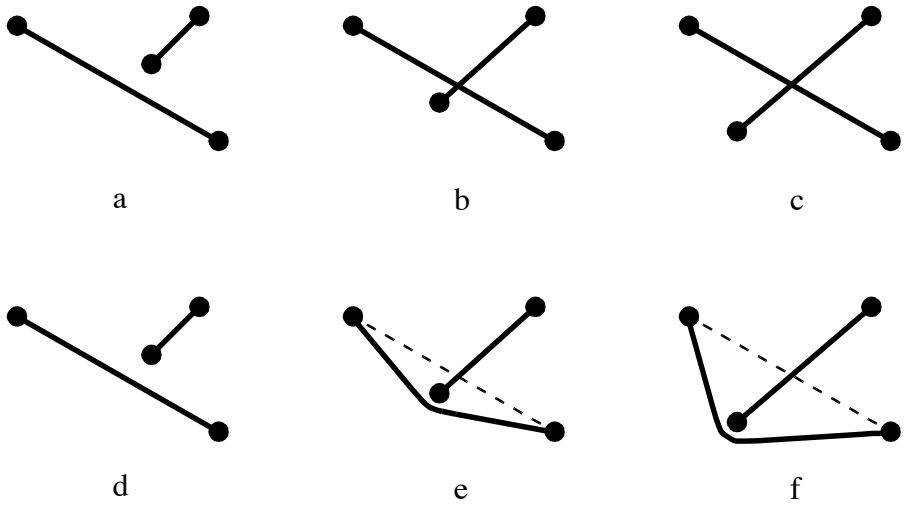


FIGURE 22 - Conflict between pairs of components. (a), (b), and (c) are three pairs of components assigned to the same layer and (d), (e), and (f) are their optimal embeddings respectively, ignoring any other component. The pair in (a) has zero conflict while the (b) and (c) have low and high conflict respectively.

Formally, the conflict between a pair of components is defined as follows:

Definition 6 (Conflict) Let c_i, c_j be components. Let W^* be the length of an optimal routing of c_i and c_j inside the bin, independent of other components[1]. The conflict $H(c_i, c_j)$ is the minimum pair-wise detour, that is: $H(c_i, c_j) = W^* - (W_0(c_i) + W_0(c_j))$. Note that $H(c_i, c_j)$ is zero if the components are assigned to different layers. If c_i, c_j cannot be routed inside the bin without crossing each other, the conflict is ∞.

H represents precisely the detour in the case of two components. To estimate the detour of the entire assignment, we use a sum of the pair-wise conflicts of the components:

$$D(S) = \sum_{c_i, c_j \in \hat{C}} H(c_i, c_j) \tag{4}$$

1. A routing of a component is assumed to be embedded solely on the layer to which it was assigned.

By combining the detour estimation function from (*4*) and (*1*) and (*3*) we end up with the final cost-function:

$$C(S) = (1-\alpha)W_0(S) + \alpha V(S) + (1-\alpha) \sum_{c_i, c_j \in \hat{C}} H(c_i, c_j) \quad (5)$$

3.2.4 Properties of the cost function

The cost function is a linear combination of two functions of S, $V(S)$ and $D(S)$, which represent the exact number of layer crossings in S and the estimation of wire detour length in S respectively, plus a value $(1-\alpha)W_0(S)$ which is a constant for a given problem and thus does not have to be considered when minimizing the cost. $D(S)$ is a non-negative function. It is zero when the components in the solution do not intersect at all and tends to give higher values when the actual detour is higher. Note that $D(S)$ is not a lower or upper bound of the actual detour nor can its error be bounded by a constant. This is because $D(S)$ considers only the conflicts between pairs of components and ignores the dependency between different pairs. Figure 23 shows two examples, one with an underestimated detour and another with an overestimated detour. Figure 24 shows an example where the estimated detour can be arbitrarily high while the actual detour is bounded. A large number of experiments however show a close correlation between the estimated and the actual detour. The results of some of those experiments are included later in this chapter.

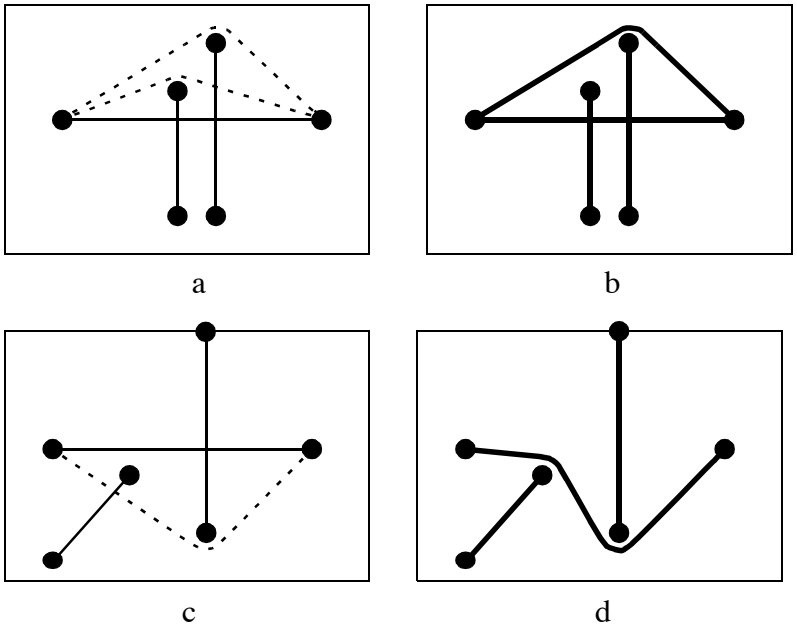


FIGURE 23 - Pair-wise conflict vs. the actual detour. The pair-wise conflict is neither an upper bound nor a lower bound on the actual optimal detour. In (a) the horizontal component conflicts with the two vertical components, resulting in overestimation of the detour compared to the actual routing in (b). In (c) the horizontal component conflicts only with one component but in the actual routing (d) it conflicts also with the diagonal component, which results in underestimation of the detour.

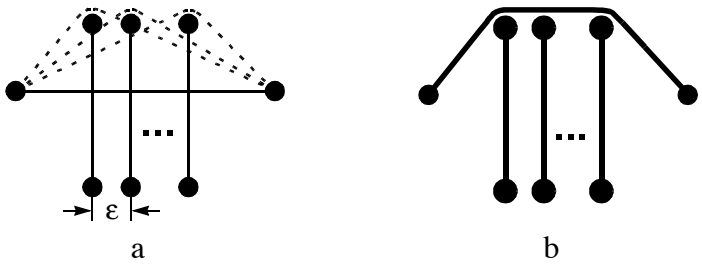


FIGURE 24 - The error of the estimated detour function has no upper bound. In (a), a single horizontal component intersects with arbitrarily large number of vertical nets which are $\varepsilon > 0$ spaced. The sum of the pair-wise detours goes to infinity when the number of vertical nets goes to infinity while the actual detour length is bounded (b).

The cost function $C(S)$ captures the planarity of the assignment. It is finite if and only if the solution S is planar (proven below). This property guarantees that a solution that satisfies the other requirements for feasibility is feasible if and only if its cost is finite

Theorem 1 An assignment S is planar if and only if its cost $C(S)$ is finite.

Proof From the definition of the cost function (*5*), the cost $C(S)$ is finite if and only if the pair-wise conflict of all the component pairs assigned by S to the same layer is finite, and by *Definition 6*, the conflict of a component pair is finite if and only if the pair is planar. Therefore, it is sufficient to show that S is planar if and only if every component pair assigned to the same layer is planar. Further more, since the planarity of the individual layers are independent of each other, it is sufficient to show that this holds for an arbitrary layer.

One direction of the proof is simple, if a set of components assigned by S to a layer is planar then, by definition, the layer has a planar embedding and this embedding includes a planar embedding of each component-pair on that layer which implies that all the component pairs are planar.

To prove the other direction[1] we use induction on the number of components assigned to a layer. Let $c_1 .. c_{n+1}$ be a set of components assigned to a layer such that every pair of them is planar. From the induction assumption, the set $c_1 .. c_n$ is planar and therefore has a planar embedding E. The embedding E contains zero or more paths between external terminals (note that paths of the same component may overlap). These paths, called *external paths,* divide the bin into non-connected areas called *regions* (Figure 25). For any given pair of points in the bin, a planar path between the points can be added to E if and only if the points are inside or on the boundary of the same region. The set of regions divides the terminals of c_{n+1} into a complementary and disjoint

1. An alternative approach for the proof could be to use a known criteria for planarity of non-separable graphs (see Lemma 7.2 in [15]). The criteria is based on a pair-wise property of *bridges* in a non-separable graph in respect to a simple cycle. This proof can be made by considering the components as bridges and the bounding box as the cycle. This however requires some preparation steps to match the conditions of *Theorem 1* to the conditions of the criteria. Some of the differences are: (1) The graph in the theorem may be separable and even non-connected. (2) The locations of terminals in the theorem are given while in the criteria they are free. (3) In the criteria, the tree for each component is given while in the theorem they are free and adding junction points is allowed. We have chosen to present a proof which is somewhat closer to the concept of rubber-band sketches.

subsets of terminals $g_1 .. g_k$ (Figure 26-a) each of which is a set of terminals in a different region[1]. We modify E by adding to it a planar embedding of each of the groups (Figure 26-b). If E contains only one group of c_{n+1} terminals then the resulting E is a planar embedding of $c_1 .. c_{n+1}$. Otherwise, let g_1 and g_2 be two arbitrary groups of c_{n+1}. g_1 and g_2 can be connected in E by a path P (Figure 26-c) which does not intersect with any terminal and is planar except for a finite number of crossings of external paths (at least one) of other components. If both g_1 and g_2 have an external terminal then *every* path between these external two terminals has to cross an external path of another component which separates between the terminals of g_1 an g_2 but since c_{n+1} is pair-wise planar with each of the components $c_1 .. c_n$ this is not the case and therefore at least one of g_1 or g_2 contains no external terminals. Let assume, without lost of generality, that g_1 has no external terminals. The path P crosses, in direction from g_1 to g_2 a finite series $P_1 .. P_q$, $q \geq 1$, (Figure 26-c) of external paths of components other than[2] c_{n+1}. P_1, the first crossed path in the sequence can be transformed such that the terminals of g_1 and their interconnect are moved to the region on the other side of P_1 while the rest of the terminals and connections in E stay in the same regions (Figure 26-d). This step can be repeated for the paths $P_2 .. P_q$ (in that order) until g_1 end up in the same region of g_2 and the terminals of $g_1 \cup g_2$ are connected within this region with no crossings. This reduced by one the number of non-connected groups in c_{n+1}, and if repeated will end with a planar embedding in which all the terminals of c_{n+1} are in a single region. The resulting embedding is a planar embedding of $C_1 .. C_{n+1}$ and therefore the set of $C_1 .. C_{n+1}$ is planar. Q.E.D.

---

1. If any of the terminal of C_{n+1} happens to be on a path in E we can move this path such that the terminal is in a close neighborhood of the path but does not intersect with it. This is true since the number of terminals in a bin is finite. Therefore, we assume, without loss of generality, that the terminal of C_{n+1} does not intersect with the paths of E.

2. Note that the planar embedding of each of the groups $g_1 .. g_q$ can include external paths. This happens if a group contains two or more external terminals.

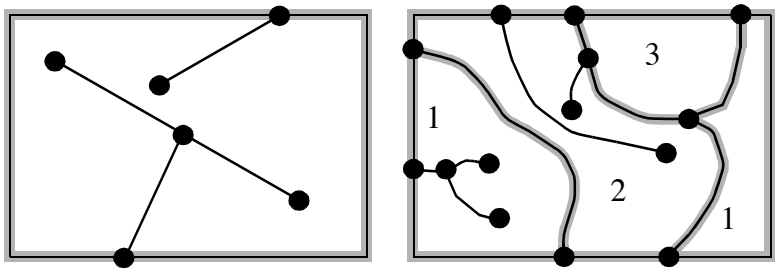


FIGURE 25 - External paths and regions. An external path connects two terminals on the bin boundary. The set of (possibly overlapping) external paths in a bin layer divides the bin area into disconnected regions. The gray shadow mark the boundaries of the regions. Example (a) has a single region that includes the entire bin while (b) has 4 regions (marked 1 to 4).

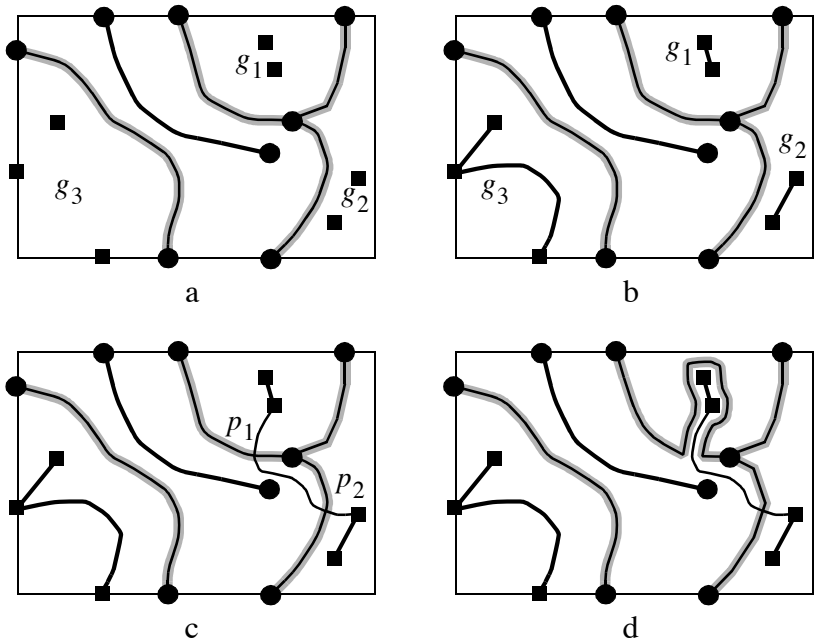


FIGURE 26 - Constructing a planar embedding. The regions in this example (a) divides the terminals of component C_{n+1} into three groups $g_1..g_3$, each of which is a set of terminals in a separate region. The terminals of each group can be interconnected with no intersections as shown in (b). Every pair of groups can then be connected by a path which is planar except for a finite number of crossings of external path of other components. (c) shows such a path between g_1 and g_2 with two crossings of external paths, P_1 and P_2. By modifying the path P_1 as shown in (d) we can reduce the number of paths crossings by one. Repeating this step for P_2 will end up with the groups g_1 and g_2 in the same region, merged into a single group. Repeating this group merging step will result with a planar embedding in which all the terminals of C_{n+1} are in a single region.

3.2.5 User control of α

The cost function (*1*) (page 33) is controlled by the parameter α which indicates the desired balance between low number of vias and short wiring. A practical setting of α depends on the size of the design and requires the user to perform scaling of α values for designs of different sizes. To ease the selection of this value, SURF provides a more abstract way of controlling it and performs the scaling automatically. This is done using a normalized parameter β in the range 0 to 100, which indicates the relative importance of having a low number of vias compared to having short wiring. Higher values of β prefer short wiring and lower values prefer fewer vias. SURF maps the value of β to α such that α is zero when β is zero and α is increased when β is increased. The mapping is done as follows. First a value α_1, which represents the cost of a single via in length units is computed (d is the length of the diagonal of the design):

$$\alpha_1 = 0.12d\frac{\beta}{100} \tag{6}$$

Then, α can be computed using the formula:

$$\alpha = \frac{\alpha_1}{1+\alpha_1} \tag{7}$$

The constant 0.12 in (*6*) sets a range of α which was found useful by users though other values may do as well.

3.3 The Layer-Assignment Algorithm

3.3.1 Introduction

The proposed Layer-Assignment Algorithm (LAA) solves the LAP. It accepts P, an instance of the LAP, and outputs a feasible solution S, while trying to minimize the cost $C(S)$. The definition of the LAP does not imply any specific optimization technique, and many optimization algorithms can be used to solve it. Our choice, which is a greedy approach, is efficient, simple, and yet generates practically good solutions as supported by the experimental results included in this thesis.

The LAA performs the assignment in three steps (Figure 27):

STEP I - Generating 2-Nets. Each multi-terminal net is decomposed into a set of two-terminal nets (*2-Nets*). This is done by generating a tree which spans the net terminals. The tree can include Steiner points (junctions) to reduce the length of the tree. The decomposition into 2-Nets is done independently for each net. Note that the 2-Nets defined by this step are used internally by the LAA and do not necessarily correspond to the 2-Nets in the final design (which are determined by the single-layer router in use).

STEP II - Building the *assignment graph* for each 2-Net. An *assignment graph* represents all the assignments that the algorithm is going to consider for a given 2-Net. This includes all the possible layer crossings and the point-to-point connections between the vias and terminals of this 2-Net. The graph for each 2-Net is built independently of the other 2-Nets

STEP III - Assignment. In this step, the 2-Nets are assigned to layers while trying to minimize the overall cost of the solution. Each 2-Net can be broken into sections which are assigned to different layers. The proposed algorithm is guaranteed to generate a planar and feasible solution.

The details of the three steps are presented in the following sections.

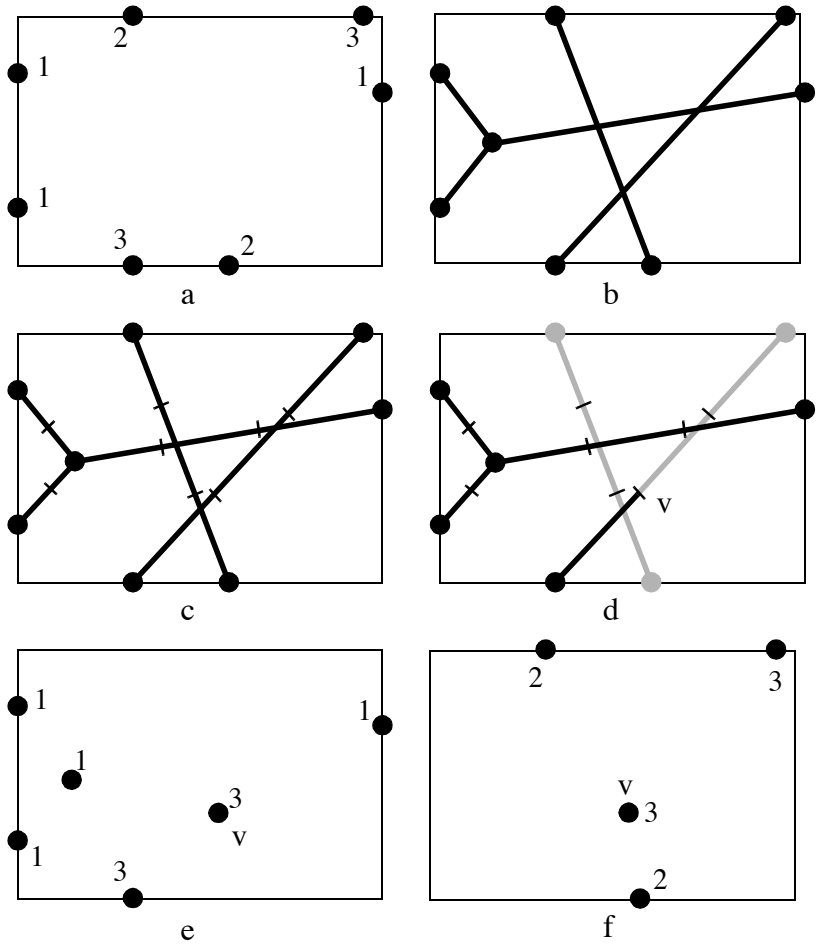


FIGURE 27 - The three steps of the assignment algorithm. The LAP in (a) has three nets and should be routed on two layers. Step I breaks the nets into 2-terminals nets (b) by generating a Steiner tree for each net. Step II generates for each 2-Net an assignment graph which defines a subset of its possible assignments the algorithm will consider (to keep the example simple, we don't include here the actual assignment graphs of the 2-Nets). In this example we are considering only assignments with possible vias at the candidate points marked with a short segment (c). Step III then assigns the 2-Nets to the layers by selecting for each 2-Net one of its candidate assignments. In this example (d), nets 1 and 2 are assigned entirely to the black and the gray layers respectively while net 3 crosses layers at point v. The output of the LAA, a single-layer routing problem for each of the layers is shown in (e) and (f).

3.3.2 Step I - Breaking the Nets Into 2-Nets

This step decomposes each multi-terminal input net into 2-Nets that represent edges of a planar tree. The nodes of each tree include the terminals of the input net as well as any Steiner points

introduced by the algorithm. The decomposition is done independently for each input net and ignores any layer consideration. The decomposition can be achieved in several ways which differ in the count and total length of the tree edges (and 2-Nets) they generate. It is desirable on one hand to reduce the total length of the tree as it affects the length of the wiring, and on the other hand to reduce the number of edges (and thus the number of 2-Nets) so that the database is smaller and the assignment is faster. Figure 28 shows three methods of generating the tree. The simplest one is to generate a minimum spanning tree (MST) of the net terminals. An MST can be generated efficiently and results in a tree with minimal number of edges. Its total length however can be up to $\frac{\sqrt{3}}{2}$ times the total length of the shortest possible tree [18] [14]. The second approach is to generate a minimum Steiner tree (MSTT) of the terminals[1]. This is the shortest tree among all the possible trees, but its edge count can be almost double that of the minimum spanning tree. The third approach is to use a minimal Steiner tree with collapsed edges (MSTTC). This is a MSTT in which Steiner points that don't significantly reduce the total tree length are removed. This is done by collapsing edges which are incident to Steiner points if the increase in the total tree length is below a predefined threshold. In general, the MSTTC is shorter than the MST and it has less nodes and edges then the MSTT. The experimental results presented later in this thesis include comparisons of routing using the three tree kinds.

1. Our implementation of the LAA includes an algorithm [11] which uses heuristics to generate a Steiner tree of a given set of points using Euclidean, Rectilinear, or Octilinear metrics. The description of this algorithm is outside the scope of this thesis.

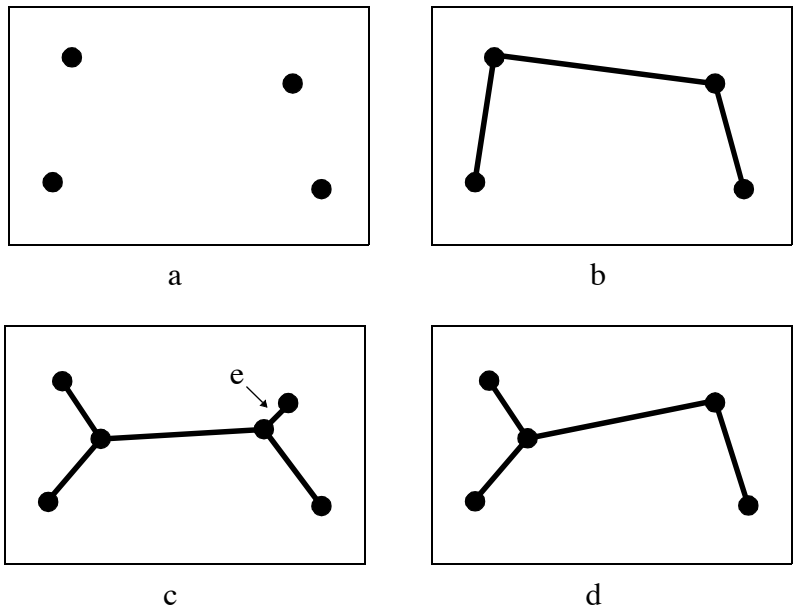


FIGURE 28 - Three methods of decomposing a net (a) into 2-Nets. The minimum spanning tree (b) results in a small tree while the minimum Steiner tree (c) has lower wire length. The minimum Steiner tree with collapsed short edges (d) provide a balance between the two (*e* denotes the collapsed edge).

3.3.3 Step II - Generating The 2-Net Assignment Graphs

An assignment of a 2-Net decomposes the 2-Net into vias and single layer, point-to-point connections called *branches*. The vias are defined by their location in the routing area and the layers on which they exist. The branches are used to connect the terminals and the vias of the 2-Net. Since there is an infinite number of possible assignments for a single 2-Net, (even if we consider only assignments with a single via), we reduce the search space by considering a smaller set of assignments called *candidate assignments*.

A special graph called an *assignment graph* is used to capture the set of candidate assignments considered by the LAA for a single 2-Net. In this directed graph, nodes correspond to branch end-points, which include input terminals, Steiner points, and candidate via locations on specific

layers. The arcs represent branches and layer crossings. This graph also contains special *source* and *sink* nodes which represent the end-points of the 2-Net[1]. The assignment graph is constructed so that each possible candidate assignment for the 2-Net corresponds to a source to sink path through the graph. Also, the arcs are weighted[2] so that a shortest path represents an optimal assignment of the 2-Net.

A simplified version of the assignment graph is presented in Figure 29. For this 2-Net there are 3 wiring layers and a single via candidate location is considered. Note that there are 9 candidate assignments in this case, 3 with no via, 4 with a single layer crossing, and 2 with two crossings, each of which corresponds to a path through the graph.

1. The decision of which end-point will be the source and which will be the sink is arbitrary and does not affect the result of the routing.

2. Because of the non-linearity of the cost function, the weights are actually assigned to a modified version of the assignment graph. This is described later, as part of Step III of the LAA.

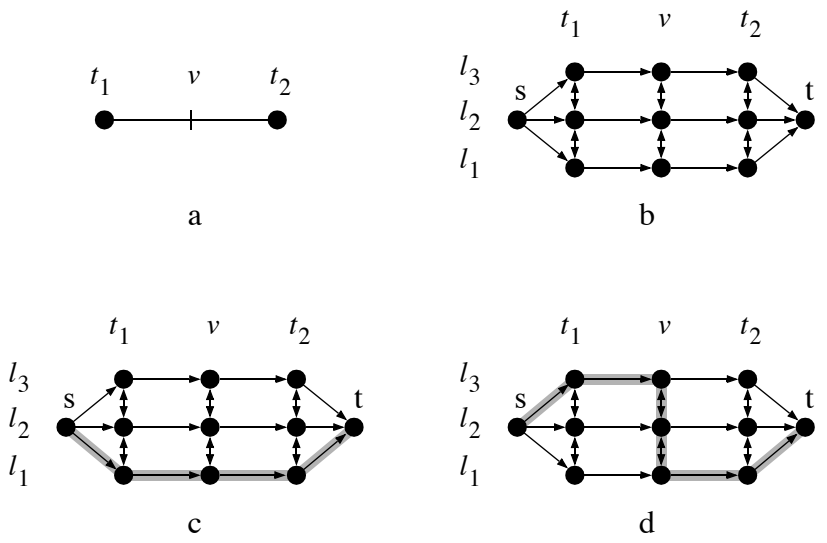


FIGURE 29 - An example of a simple 2-Net assignment graph. The 2-Net in (a) is to connect two terminals t_1, t_2 with three layers available for routing. A third point v is defined as a position of a candidate via. The assignment graph for this 2-Net is shown in (b). Each of the three points t_1, t_2, and v defines three graph nodes, one on each of the three layers l_1, l_2, l_3. The nodes s and t represent the source and sink nodes respectively. The vertical edges represent layer crossings between adjacent nodes of the same location while the horizontal edges represent wires between nodes on the same layer. (c) and (d) show two paths on the graph between s, and t, each representing a possible assignment of the 2-Net. The path in (c) represents a direct connection of t_1, t_2 on layer l_1 while (f) represents a connection with a crossing from layer l_3 to l_1 at point v

The complexity of the assignment graph used by the LAA controls the number of candidate assignments considered by the algorithm. A more complex graph considers a larger set of candidate assignments and potentially yields a higher quality solution at the expense of a larger search space and a longer running time (Figure 30). To achieve a reasonable running time, the proposed LAA considers relatively few via candidates (between 1 and 5)[1], evenly spaced along the straight line between the end-points of the 2-Net. Note that for a graph with m layers and n possible candidate points, there are m^{n+1} possible assignments for the 2-Net.

1. To guarantee that the LAA will find a feasible solution, each 2-Net must have at least one candidate via. This is used in the proof of correctness of the LAA presented later.

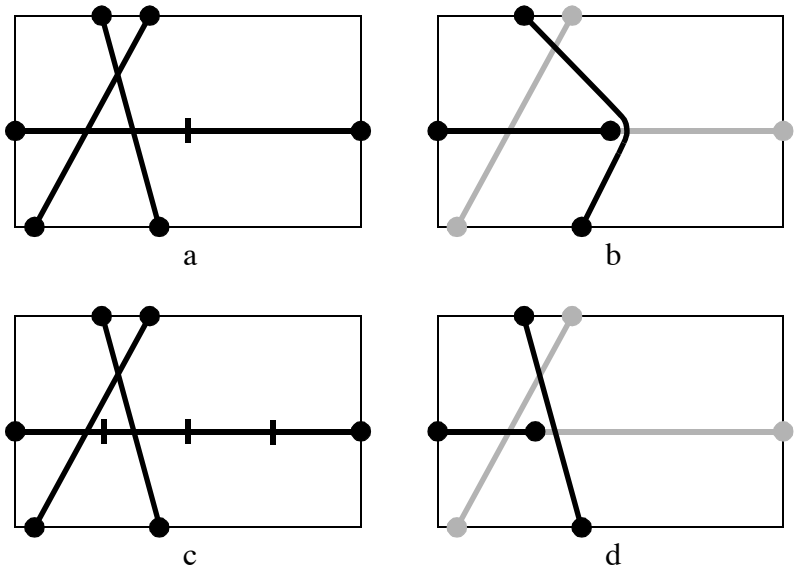


FIGURE 30 - Two choices for the number of candidate points per 2-Net. The horizontal 2-Net in (a) and (c) are assigned using one and three candidate points respectively which results in assignments (b) and (d) respectively. Assignment (d) has a shorter wire length as the candidate points of (c) have a better resolution.

A post-processing step (Figure 31) [11] that locally repositions vias is used to compensate for the limited granularity of the candidate points and for the fact that via positions are restricted to the straight line between the end-points (the post-processing is not part of this research). Our experiments show that when using this post-processor, considering more than five candidate points per 2-Net yields no significant improvement in the final routing quality

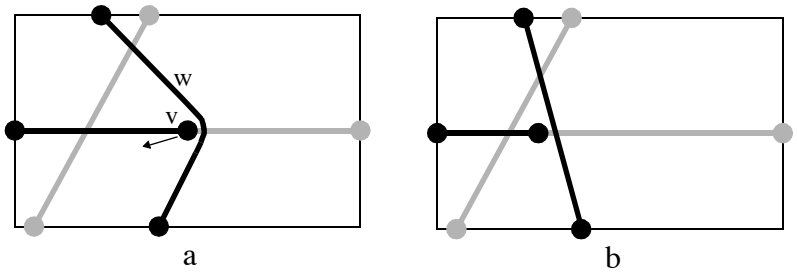


FIGURE 31 - The local-spacing post-processor. The sketch of the first example of Figure 30 is shown in (a). The post-processor detects the 'tension' applied to via *v* by the wire *w* and moves it in the direction of the arrow while treating the wires as rubber bands. This is done iteratively and eventually converges to the sketch in (b) with shortest wire length. In the general case, the post-processor finds a sketch in which any topology preserving relocation of a single via or a Steiner point does not reduce the wire length. The use of the post-processor compensates for the limited number of via candidate points as well as the restriction of vias to be on the straight line between the end points. The local-spacing post-processor is not part of this research

3.3.4 Step III - Solving the Layer-Assignment Problem

Step III of the LAA (the layer-assignment step) generates a solution for the LAP by selecting for each 2-Net one of its candidate assignments such that the overall assignment is feasible and has minimal cost. In this section we describe how the LAA performs this step.

3.3.4.1 Configurations and Assignments

The operation of the assignment step can be viewed as a search in the domain of *configurations*.

Definition 7 (configuration) A configuration is a mapping of some 2-Nets to their candidate assignments. That is, each 2-Net of the design is either mapped to one of its candidate assignments defined by its assignment graph, or it is not mapped at all. For a given configuration, the mapped and unmapped 2-Nets are said to be *assigned* and *free* respectively in that configuration.

Intuitively, a configuration represents an assignment of part of the interconnect of the bin, and is said to be *complete* when it assigns all the 2-Nets of the bin. A configuration is said to be *planar* if

the branches defined by its assignments of the 2-Nets can be routed on their corresponding layers without crossing each other or the bin boundary.

A solution to the LAP can be constructed directly from a complete configuration. The terminals of the solution are the original terminals and any vias introduced by the assignment of the 2-Nets in the configuration. A component in the solution is a maximal set of terminals and vias on a single layer that are connected by branches in the complete configuration (Figure 32). Note that single-layer Steiner points and explicit point-to-point connection topologies for multi-terminal components do not appear in the solution. A component merely defines a set of points that must be connected on a single layer leaving the single-layer router free to introduce Steiner points and choose component topologies. The cost of a complete configuration is defined to be the cost of its corresponding solution.

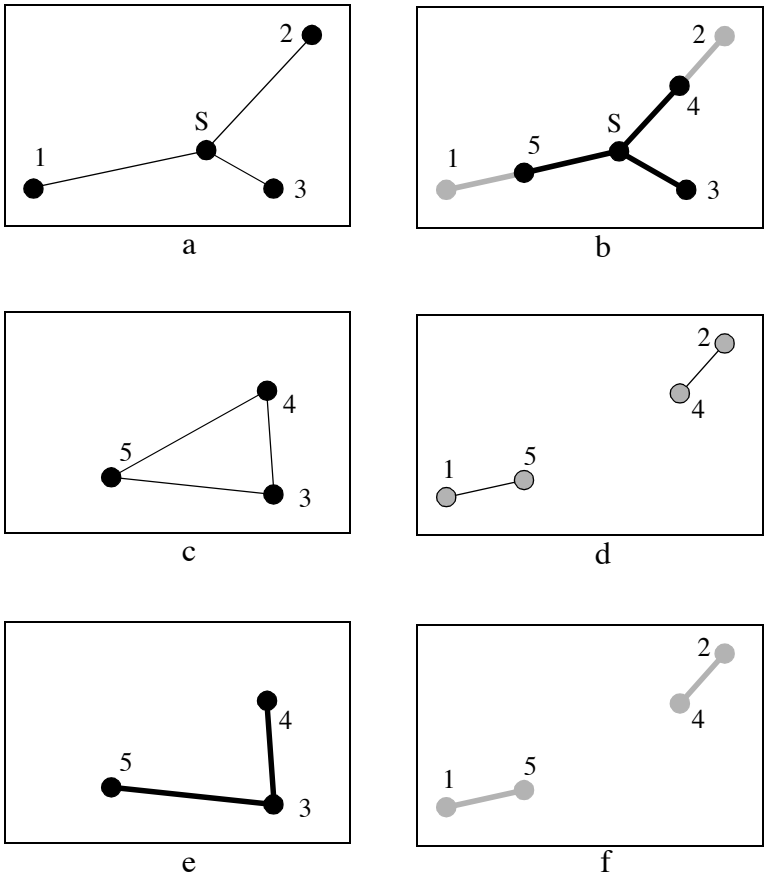


FIGURE 32 - The corresponding solution of a configuration. In this simple example, a design with a single net and three terminals 1, 2, and 3, is to be assigned to two layers. The net is first decomposed into three 2-Nets (a) using a Steiner point *S*, and then each 2-Nets is assigned to one of its candidate assignments (b). The assignments of the 2-Net in this example introduce two vias marked 4 and 5. The corresponding LAP solution for the configuration in (b) is shown in (c) and (d) for the black and the gray layers respectively. The black layer has a single component with three terminals {3, 4, 5}, and the gray layer has two components {1, 5} and {2,4} respectively. (e) and (f) show possible embeddings of the assigned layers in (c) and (d) respectively. Note that on the black layer, the branches in the embedding (which is suboptimal, for the purpose of demonstration) do not map one-to-one to the branches defined by the configuration in (b). That is because the single layer router has the freedom to connect the components in arbitrary topology (a better embedding of the black layer would be to have a Steiner point similar to *S*).

The concept of corresponding solution of a complete configuration can be extended to a general configuration. If the configuration is not complete then its corresponding solution is partial in the sense that some connections might be missing.

The goal of the assignment step is to find a complete configuration whose corresponding solution is feasible and has minimal cost. Since by definition the cost of a configuration is the cost of its corresponding solution, it is sufficient to minimize the cost of the configuration found. As for feasibility, by the following simple lemma, it is sufficient to find planar configuration to guarantee feasibility.

Lemma 1 Let q be a complete configuration and S its corresponding solution. If q is planar then S is feasible.

Proof By the method of constructing the 2-Nets and their candidate assignments, S is guaranteed to have all the properties, except from planarity, required of a feasible solution (*Definition 2*). As for planarity, if q is planar then it has a planar embedding, and this planar embedding includes a planar embedding of S (Figure 33).

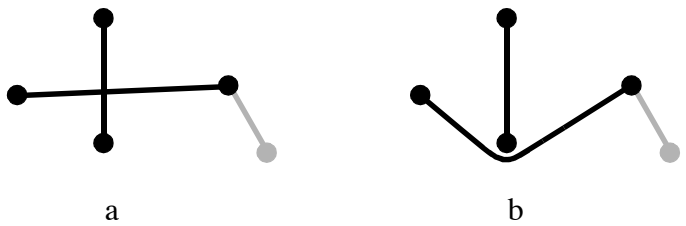


FIGURE 33 - Planar embedding of the corresponding solution. The configuration in (a) assigns three 2-Nets, two to the black layer and one to the gray one. This configuration is planar and has a planar embeddings such as the one in (b). This planar embedding is also a planar embedding for the corresponding solution which is planar as well.

Using Lemma 1 and the relation between planarity and finite cost (*Theorem 1*), the goal of the assignment step can be re-formulated as:

Definition 8 (The Assignment Step Problem - ASP) Given the 2-Nets and their assignment graphs, find a complete configuration with minimal cost.

3.3.4.2 The configuration search algorithm

The assignment step finds the solution by performing a search in the configuration domain. The range of optimization techniques that can be used is wide and includes simulated annealing [30], group migration, steepest descent, and others. Our implementation uses a simple steepest descent that we have found to be both efficient and to result in high quality layouts. We believe that this approach is sufficient to show the merits of the proposed approach though a more advanced search technique may results in better solutions.

The search algorithm (Algorithm 1) consists of two phases, the *assignment* and the *improvement*. The assignment phase starts with the empty configuration (i.e. all the 2-Nets are free) and iteratively assigns a free 2-Net to one of its candidate assignments. The 2-Net and its assignment chosen on each iteration are such that the cost of the resulting configuration will be minimal. When all the 2-Nets have been assigned, the improvement phase tries to further improve the solution by iteratively reassigning 2-Nets, one on every iteration. Again, the algorithm chooses in each iteration a 2-Net and a new candidate assignment of it such that the overall cost is reduced the most. The algorithm terminates when the cost cannot be further reduced by reassigning a single 2-Net.

```
// --- Assignment phase
q = empty configuration;
while q is not complete {
    for each 2-Net n_i free in q do {
        a_i = Best2NetAssignment(n_i, q)
        Δ_i = C(q|n_i → a_i) – C(q)
    }
    j = i minimizing Δ_i
    q = (q|n_j → a_j)
}

// --- Improvement phase
do {
    for each 2-Net n_i do {
        a_i = Best2NetAssignment(n_i, q|n_i → free)
        Δ_i = C(q|n_i → a_i) – C(q)
    }
    j = i minimizing Δ_i
    if (Δ_j < 0) then q = (q|n_j → a_j)
} while Δ_j < 0
```

ALGORITHM 1 - The layer-assignment algorithm. The algorithm operates in two phases, first it assigns all the 2-Nets and then reassigned them to further reduce the cost. This algorithm is greedy and on each iteration it chooses the 2-Net whose its optimal candidate assignment will result in the lowest overall cost. The core of the algorithm is the function *Best2NetAssignment* which finds the optimal candidate assignment of a free 2-Net given a configuration. An optimal algorithm for this function is presented later in this thesis.

Note that the cost of an assignment for a 2-Net depends on the assignments of the other 2-Nets, both free and assigned. As a result, changing the assignment of one 2-Net can change the optimal assignment of other 2-Nets (Figure 34). This dependency is why the improvement phase is required (Figure 35) and why the algorithm recalculates the optimal assignment of the 2-Nets on each iteration. Our implementation reduces the amount of recalculation done by using an incremental approach. On the first iteration, the best assignment for each 2-Net is calculated and

saved. Later, when a 2-Net is assigned or reassigned, only the optimal candidate assignments for the 2-Nets affected by this assignment are recomputed.

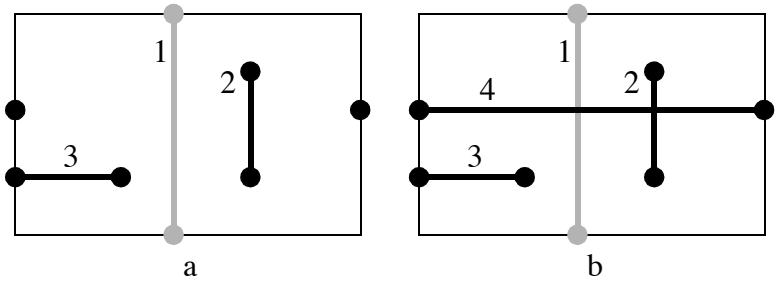


FIGURE 34 - Interdependency between assigned 2-Nets. (a) and (b) show the configurations before and after assigning 2-Net 4 to the black layer. The assignments in (a) of the three 2-Nets 1, 2, and 3 are all optimal. However, after assigning 4, the assignment of 2 is suboptimal as assigning it to the gray layer will result in a lower cost due to its conflict with 4. The other 2-Nets, 1 and 3, are not affected by the assignment of 4 and their assignments are still optimal.

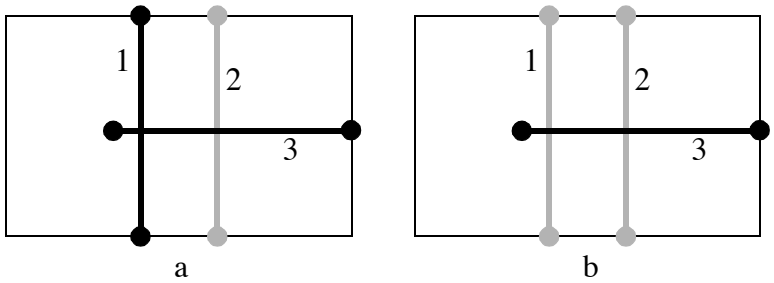


FIGURE 35 - Example of assignment improvement. This figure shows an example of improvement of an assignment. The design has three 2-Nets 1, 2, and 3, which are assigned to the black and the gray layers. The assignment phase assigned the three 2-Nets in order and results in the complete assignment in (a). The improvement phase then reassigns 2-Net 1 to the gray layer so the overall cost is improved. Note the assignment phase initially assigned 2-Net 1 to the black layer arbitrarily because the cost of assigning it to each of the layers was the same.

The core of the algorithm is the function *Best2NetAssignment* which given a configuration q and a free 2-Net n, finds a candidate assignment a of n that minimizes the cost of the configuration after assigning n to a. A naive implementation of that function would be to enumerate all the candidate assignments of n and calculate for each one its effect on the cost of q. This however would be much too time consuming because of the large number of candidate assignments and the complexity of calculating the cost increase due to each assignment. A more efficient algorithm which yields optimal results is presented later in this chapter.

3.3.5 Properties of the configuration search algorithm

Note: the discussion in this section assumes that the algorithm used to compute the function *Best2NetAssignment* results in optimal solutions as done by the 2NAA algorithm (page 59).

The proposed search algorithm is guaranteed[1] to terminate with a complete configuration whose corresponding solution is feasible. To show that, we first show that assigning a 2-Net cannot change the cost of the configuration from finite to infinite.

Lemma 2 Let q be a configuration with a finite cost and let n be a 2-Net, free in q. Under these conditions, n has a candidate assignment a such that the cost of the configuration q, resulting from assigning n to a, $q_1 = (q|n \rightarrow a)$, is finite as well.

Proof Let e_1 and e_2 be the end-points of n. By the construction of the assignment graph, n has at least one candidate via which like all the candidate vias, is internal to the bin. Let v be such a candidate via. Let a be the candidate assignment of n with two branches, $\{e_1, v\}$ on one layer and $\{v, e_2\}$ on an adjacent layer[2]. Via v is internal to the bin and is not connected to any branch other then the two defined by a (each of a separate layer). These two branches do not disconnect

1. It is assumed that the design has at least two layers and that each of the 2-Net has at least one via candidate point.
2. We assume that any design submitted to the layer-assignment algorithm has at least two layers.

the routing area on their respective layers and therefore, their components are pair-wise planar with all the other components assigned to same layer. Q.E.D.

Theorem 2 The search algorithm is guaranteed to end with a complete configuration whose corresponding solution is feasible.

Proof By Lemma 1, it is sufficient to show that the algorithm will result in a complete configuration which is also planar, and by *Theorem 1* it is sufficient to show that the result configuration is complete and has finite cost. The assignment phase starts with a configuration of finite cost and by Lemma 2 it is guaranteed to successfully assign on each iteration a free 2-Net while increasing the cost by only a finite value. This guarantees that the assignment phase will terminate with a complete configuration of finite cost. The improvement phase starts with a complete and finite cost configuration and on each iteration can only decrease the cost by reassigning a 2-Net. This guarantees that the completeness and the finite cost properties of the configuration are preserved. The improvement phase is guaranteed to terminate since the number of complete configurations is finite. Q.E.D.

Note that the bound on the complexity of the improvement phase as given in this proof is too high for practical applications. However, in all of our experiments, the improvement phase has about the same number of iterations as the initial assignment step or less. In case the algorithm will fail to converge, the improvement phase can be modified to terminate when the cost improvement is smaller than a given value or after a certain number of iterations have been completed.

The proposed steepest descent search algorithm is sub-optimal and can be trapped in local minimum as shown in Figure 36. The sensitivity of the search to local minimum could be reduced by using more complex optimization techniques which consider larger steps or do allow temporary increases in the configuration cost.

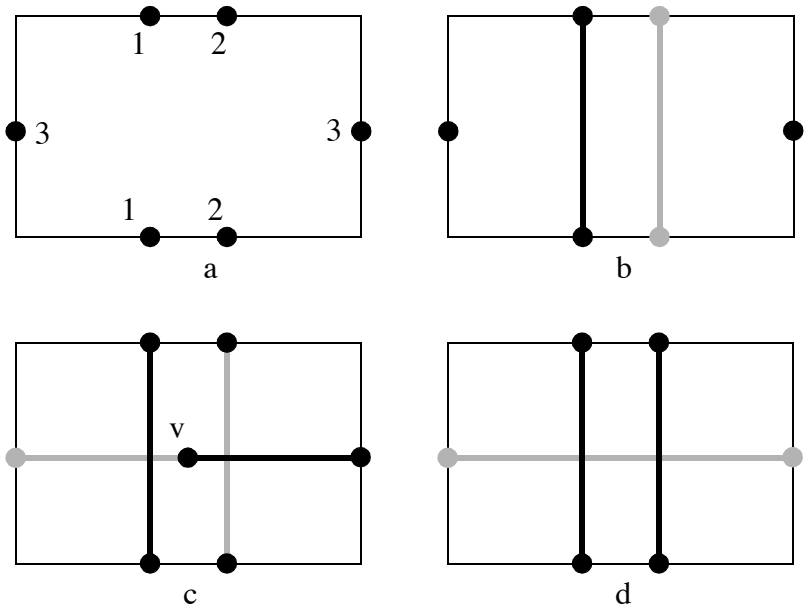


FIGURE 36 - Local minimum in the configuration search. The three 2-Nets in (a), 1, 2, and 3, are to be assigned to two layers. The assignment phase (b) first assigns net 1 to the black layer, and then net 2 to the gray layer. These were arbitrary choices as the optimal assignments of the 2-Net are not unique. Then, it assigns 2-Net 3 using a via as shown in (c). The improvement phase can not make any improvement to the assignment in (c) by reassigning a single 2-Net, and this leaves (c), which is sub-optimal, to be the final solution. An optimal solution is shown in (d).

3.3.6 The 2-Net Assignment Algorithm (2NAA)

The core of the Layer-Assignment Algorithm (Algorithm 1) is the function *Best2NetAssignment* which finds a best assignment for a given 2-Net. This function solves the following problem:

Definition 9 (2-Net Assignment Problem - 2NAP) - Given a configuration q and a free 2-Net n, find a candidate assignment a of n such that $C(q|n \rightarrow a)$, the cost of q after assigning n to a, is minimized.

The proposed LAA uses an algorithm called the 2-Net Assignment Algorithm (2NAA), presented below, which is guaranteed to find an optimal solution for the 2NAP. The 2NAA finds the solution

by first assigning costs to the edges of the assignment graph[1], and then finding a least-cost path in the graph, which by the way the costs are assigned, is guaranteed to represent an optimal solution.

The costs of the edges are set according to their kinds, which can be one of the following:

- ***End-point edges*** - These are edges that connect the source and sink nodes with the nodes representing the end-points of the 2-Net.
- ***Via edges*** - These edges represent layer crossings (shown in Figure 29 as vertical lines).
- ***Branch edges*** - These edges represent branches (shown in Figure 29 as horizontal lines).

The cost of each end-point edge is set to either 0 or ∞ (Figure 37). It is set to 0 when having the end-point edge in the path will not require introduction of a new via at that end-point. Otherwise, it is set to ∞ (which practically eliminates this edge from the graph). The cost is determined as follows. Let u be either the source or the sink node of the graph, let t be the end-point at the u end of the graph, let t_l, $1 \le l \le M$, be the node of t on layer l, and let $e_l = (u, t_l)$ be the end-point edge whose cost is to be determined. First, a set E_t of *existing layers* of t is computed. The set includes $\dot{E}(t)$, the set of existing layers of t, if t represents an input terminal, and any other layer on which a previously assigned 2-Net has a branch incident to t. The set is then is completed to be contiguous such that if it includes two layers, then it also includes all the layers in between. The cost of e_l is set to zero if E_t includes the layer l, or if E_t is empty and set to ∞ otherwise.

1. Because of the non-linearity of the branch cost function, the actual search is done in a modified assignment graph called the *extended assignment graph*. This is discussed in detail, later in this chapter.

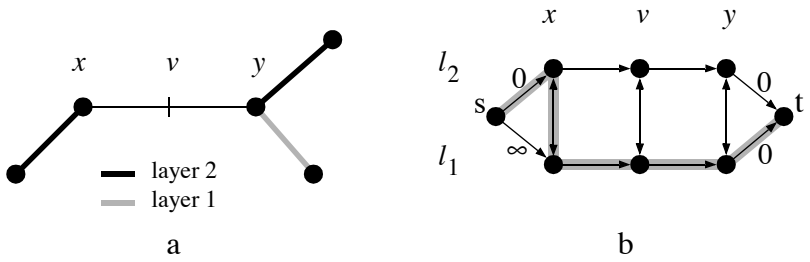


FIGURE 37 - Cost of end-point edges. The 2-Net in (a) between end-points x and y is to be assigned to two layers. The 2-Net has a single candidate via location v and its end-points have incident branches of previously assigned 2-Nets. (b) shows the assignment graph for the 2-Net. The cost of the end-point edge between the source node s and the end-point node on layer l_1 is set to ∞ since having this edge in the path will require a layer crossing to connect to the preassigned branch on layer l_1. The other three end-point edges do not require vias and thus Are assigned zero cost. This cost assignment insures that any least-cost path which defines a branch connected to end-point x on layer l_1 to include also a via edge at x between layer l_1 and l_2 (see shaded path in (b)).

Setting the cost of the via edges is much simpler, they all have the same cost α, which is the increase in the overall cost function (*5*) when a single layer crossing is added.

The cost of a branch edge is set to represent the increase in the overall cost (*5*) when the branch represented by the edge is added to the assignment. Adding a branch to the assignment can either: (a) create a new component, (b) extend an existing component, or (c) join two existing components into a single bigger component, and each of these cases may increase the assignment cost. Figure 38 shows an example of a 2-Net assignment with two branches, one (on the gray layer) forming a new component and the other (on the black layer) extending an existing component.

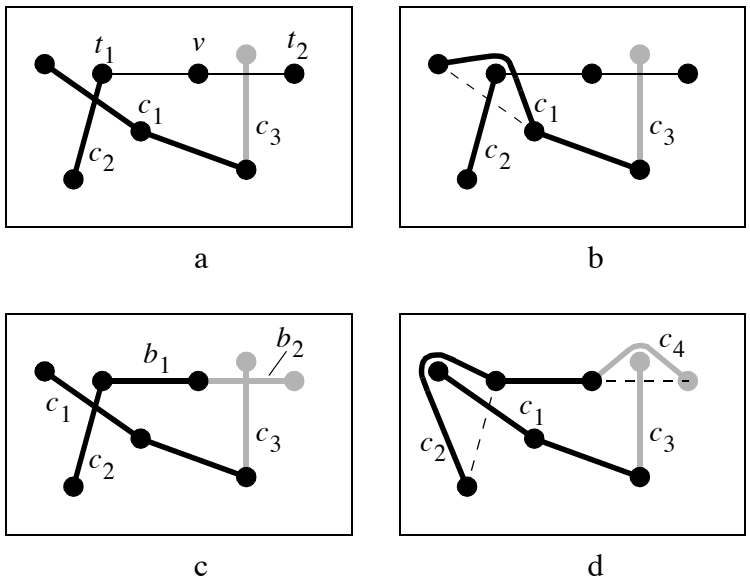


FIGURE 38 - Cost increase due to branch assignment. The configuration in (a) has a free 2-Net between t_1 and t_2, and three components from previously assigned 2-Nets, c_1 and c_2 on the black layer, and c_3 on the gray layer. The components c_1 and c_2 have conflict as shown in (b). (c) shows the configuration after assigning the free 2-Net with two branches b_1 and b_2 on the black and gray layer respectively. This assignment enlarges the component c_2 as shown in (d) which increases its conflict with c_1 and also creates a new component c_4 which conflicts with the existing component c_3.

The increase in the cost function (*5*) when assigning a branch b is as follows. where α is the user's cost function control parameter in (*1*) and C is the set of components *before* the assignment of the branch.

Case (a) - a new component is formed by the branch:

$$C(b) = (1-\alpha)\sum_{c_i \in C} H(b, c_i) \tag{8}$$

Case (b) - an existing component c is extended by the branch:

$$C(b) = (1-\alpha) \sum_{c_i \in (C-c)} \{H(c+b, c_i) - H(c, c_i)\} \quad (9)$$

Case (c) - two existing components c_1 , c_2 are joined by the branch:

$$C(b) = (1-\alpha)\left(-H(c_1, c_2) + \sum_{c_i \in (C-c_1-c_2)} \{H(c_1+c_2+b, c_i) - H(c_1, c_i) - H(c_2, c_i)\} \right) \quad (10)$$

This branch cost function represents correctly the cost increase due to an assignment of a single branch. However when it is used in a context of a path, it can result in underestimation of the cost of the 2-Net assignment because of the non-linearity of the branch cost function. If two branches b_1 , b_2 of a 2-Net to be assigned are incident and are on the same layer, then having both branches in the assignment will result in a single branch $b_1 + b_2$ which is the union of the two branches, and the cost of this branch may be higher then the sum of the costs of the individual branches (Figure 39).

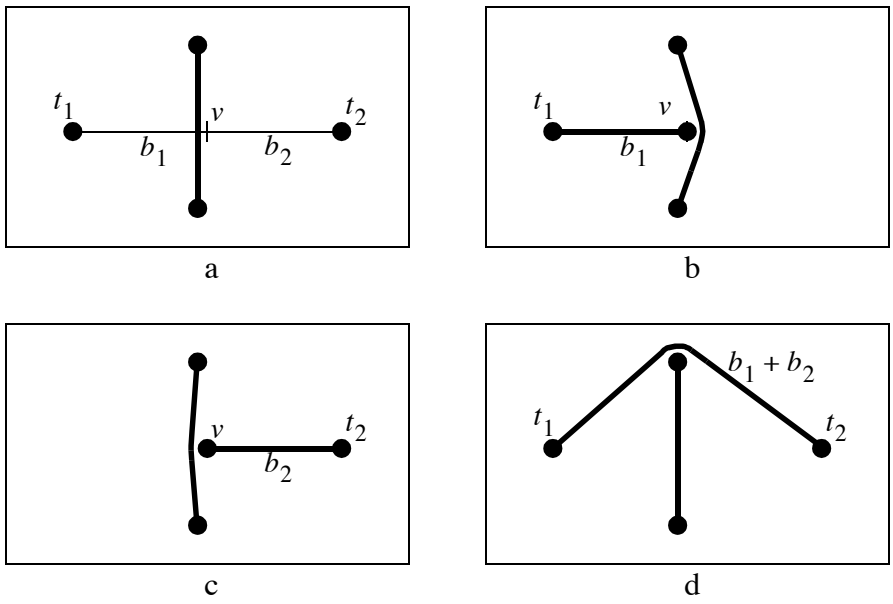


FIGURE 39 - Non-linearity of the branch cost function. The 2-Net in (a) has a single via candidate point v and two branch edges b_1, b_2 on the black layer (the branch edges on other layers are not relevant to this example). A previously assigned 2-Net, shown in (a) as a vertical line, has a branch on the black layer. The branch edges b_1 and b_2 have relatively low cost since they have low conflict with the pre-assigned 2-Net as shown in (b) and (c) respectively. However, if b_1 and b_2 are both included in the assignment, their 2-Net will effectively be a single branch which include both branches, and the conflict of this branch with the other 2-Net is higher than the sum of the conflicts of b_1 and b_2 with that 2-Net (d).

This underestimation of a 2-Net assignment cost can occur only when the path includes two branches of the 2-Net which are on the same layer and incident to each other (Figure 40). Note that this can happen even if the two branch edges are not consecutive in the path as shown in Figure 40.

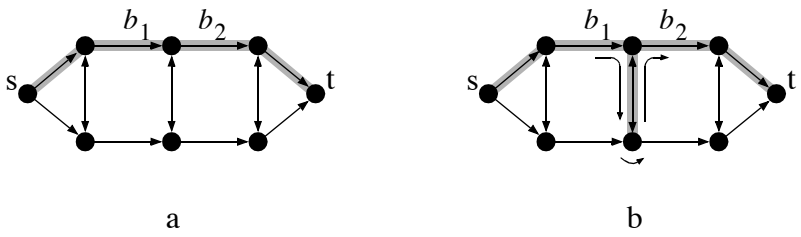


FIGURE 40 - Non-linearity of branch cost in a path. The two examples show two paths with underestimation of the cost of a 2-Net candidate assignment. The path in (a) has two consecutive branches edges b_1 , b_2 which represent two incident branches and thus its cost may be lower than the cost of the 2-Net assignment it represents. The same is true for the path in (b) (when the via cost is low enough) even though the two branch-edges are not consecutive in the path.

To eliminate the cost underestimation, the assignment graph is converted into a directed graph called, the *extended assignment graph*. This graph represents exactly all the candidate assignments represented by the original graph and yet, no path from the source to the sink nodes contains two branches which share an end-point and are on the same layer.

The extended graph is constructed by two modifications to the original assignment graph, adding *short-circuit* branch-edges, and splitting its nodes. A short-circuit branch edge is added for every sequence of consecutive (but not necessarily maximal) branch edges and it connects the origin of the first edge in the sequence with the end of the last edge (Figure 41). This enables the replacement of any sequence of consecutive branch edges in a path with a single short-circuit edge so the cost is computed correctly.

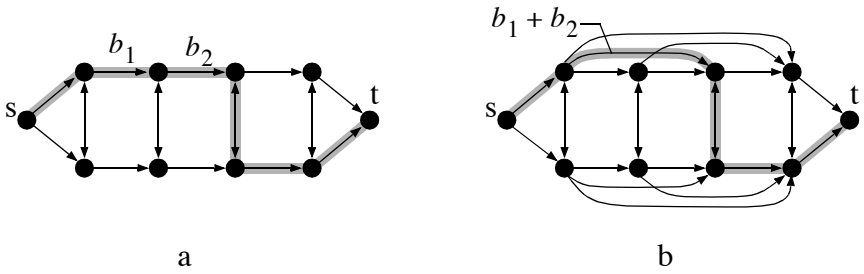


FIGURE 41 - Adding the short-circuit branch edges. (a) and (b) show an assignment graphs before and after adding the short circuit branch edges. By adding these edges, the graph in (b) has a path (shown with shade background) representing the same candidate assignment as the path in (a) but without having incident branches.

The second modification to the graph, splitting its nodes, is done to eliminate exactly all the paths with incident branches and is done as follows. Every node in the graph, except for the source and sink nodes, is replaced with three sub-nodes, called ‘up’, ‘down’, and ‘forward’ (Figure 42) which represent a restriction on how the path can continue from this sub-node. If the path includes a sub-node of type ‘up’ (‘down’) then the edge following it in the path must cross to the layer above (below). If the sub-node is of type ‘forward’ then the following edge must be either a branch (original or short-circuit) or an end-point edge.

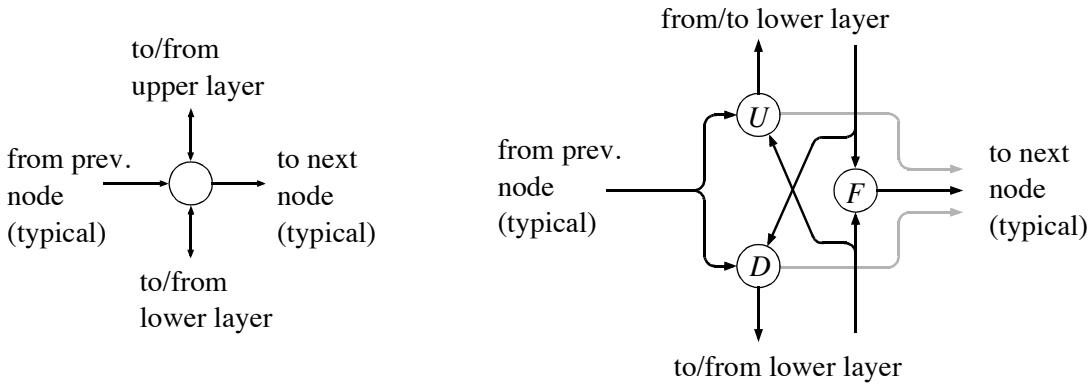


FIGURE 42 - Splitting the graph nodes. A typical graph node before (left) and after (right) the splitting is shown. The gray edges are included in the graph only if the split node represents one of the end-points of the 2-Net. The example shows a node with typical incoming and outgoing forward edges. In the general case, a node can have multiple such edges because of the addition of short-circuit edges. Note that the source and sink nodes are not split.

The splitting of the nodes guarantees that exactly all the paths that include one of the following patterns are removed from the graph:

- A branch edge followed by another branch edge. This is the simplest form of having incident branches in a path as shown in Figure 40 (a).
- An ‘up’ (‘down’) layer crossing followed by a ‘down’ (‘up’) layer crossing. This pattern is eliminated to avoid paths such as the one Figure 40 (b) which includes incident branches even though they are not consecutive on the paths.

Eliminating exactly all the paths with these patterns guarantees that a search for a least-cost path in the extended assignment graph will find a path which represents an optimal solution to the 2NAP.

3.3.7 2NAA complexity Analysis

Let m be the number of layers in the design, and let n be the number of candidate locations for vias of as a 2-Net. The extended assignment graph of the 2-Net has $O(mn)$ nodes, $O(mn)$ via edges and $O(mn^2)$ branch edges (including the short-circuit ones). The complexity of computing the edge costs is dominated by the cost computing of the branch edges which is $O(mn^2p)$ where

p is the complexity of computing the cost of a single branch edge. By using an $O(n\log n)$ shortest path search algorithm, the complexity the search is $O(mn^2(\log m + \log n))$ and therefore, the overall complexity of the algorithm is:

$$O(mn^2(p + \log m + \log n)) \tag{11}$$

As for p, if incremental computation is used (explained below) then the complexity of computing a cost of a single branch edge is $O(wr)$ when w is the average number of components a component intersects with, and r is the average time to route a pair of components. w depends on the size of the nets and their spread across the routing area, and r depends on the size of the components and the routing algorithm used.

SURF Practical Notes: Our experiments with SURF shows that increasing the value of n above five does not result in significant improvement in the routing results and since n can be practically bound by a constant, it does not affect how SURF's run-time scales with increasing design sizes. The complexity p depends on the number of nets and terminals in the bin but since these values are kept bound by SURF's global router it does not affect the asymptotic run-time of SURF. Only m, the number of available layers affects the complexity of the 2NAA which grows as $O(m\log m)$. To reduce the run-time of the 2NAA, SURF computes the cost of a branch edge only upon demand, when the shortest path algorithm requires this value for the first time. Since most searches do not traverse the entire extended assignment graph, the time of computing the cost of the non traversed branch edges is saved.

3.3.8 LAA Implementation Notes

The LAA has been implemented as part of the SURF routing system. It is written in ANSI C and has been developed on Sun SPARC workstation under Motif, and using GNU's *gcc* compiler. It has been ported successfully, as part of SURF, to other platforms such as IBM RS/6000 and DEC Alpha workstations. The implementation includes 19 modules with a total of 15,000 lines of code, comments, and intensive run-time checks.

Some parts of the algorithm have been implemented sub-optimally to speed up the development process and to simplify the code. This includes performing linear searches over lists, and using a simple $O(n^2)$ sorting. A more efficient coding may result in a faster layer-assignment though the routing results are expected be similar.

Various parts of the implementation use the List Management Package written by David Harrison, University of California at Berkeley, with enhancements by David Staepelaere, University of California at Santa Cruz.

3.4 LAA Extensions

The previous description of the LAA focuses on the basic principles of the algorithm. In this section, we discuss several enhancements and extensions to the algorithm.

3.4.1 Constrained Assignment

Formulating the layer-assignment as an optimization problem with a cost function to be minimized makes it possible to address different types of routing problems by merely making simple modifications to the cost function. For example one of the special routing methods we experimented with is *constrained* or *one and a half layer* routing in which wires are embedded in cut-outs in the ground and power planes. By embedding these connections in these planes, the routing can be completed with fewer layers and the manufacturing cost of the MCM is reduced. To preserve the current carrying ability of the planes, the embedded wires should be kept as short as possible and can be viewed as ‘jumpers’ (Figure 43).

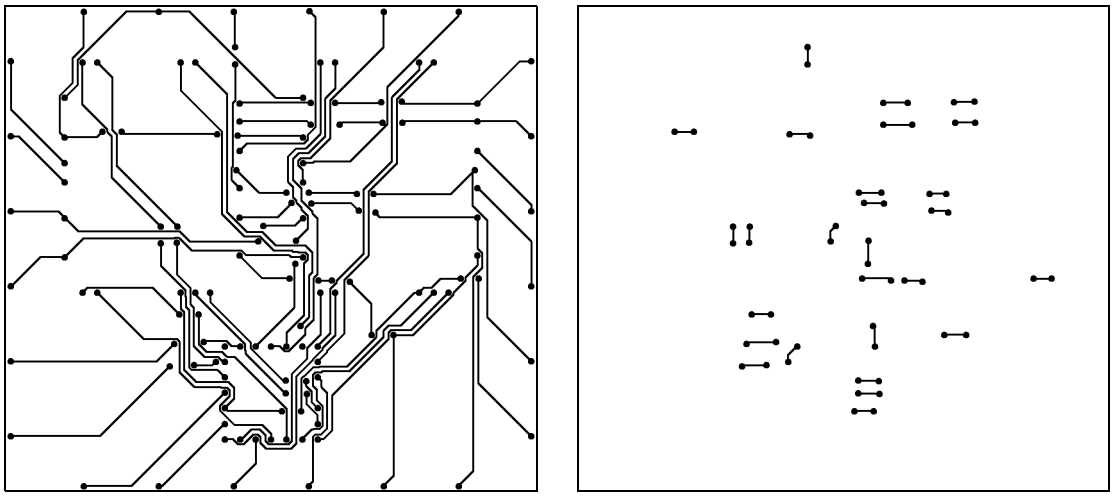


FIGURE 43 - Constrained layer-assignment. By slightly modifying the cost function, the LAA can produces special assignments such as the one and a half layer-assignment in this example. Most of the routing is done on a single layer with only short 'jumpers' on the other layer which are embedded in the ground plane.

The constrained layer-assignment is done by adding a term to the cost function (*5*) that penalizes long wires on constrained layers in such a way that many shorter wires are preferred over fewer longer ones. The cost of a solution S using the modified cost function is:

$$C'(S) = C(S) + \sum_{c} d(c)^{k} \tag{12}$$

where $C(S)$ is the standard cost function (*5*), c is a component assigned to a constrained layer, $d(c)$ is the diameter of component c (i.e. the maximal distance between terminals of c), and $k > 1$ is a constant which controls the penalty. The usage of the exponent favors multiple smaller components over fewer and longer ones as the cost has polynomial growth with increasing component size.

3.4.2 Supporting Various Metrics in the Layer-Assignment

The above presentation of the LAA assumed an *any-angle* routing model and thus uses the Euclidean metric in all distance calculations. Although any-angle routing potentially makes best usage of routing resources and results in the shortest wiring, sometimes it is required, because of production constraint, to restrict the wiring to be *rectilinear* or *octilinear*.

The LAA can be extended to handle rectilinear and octilinear wiring models as well. This is done by using a rectilinear ('Manhattan') or octilinear metrics respectively instead of the Euclidean metric. This change affects the following aspects of the LAA:

1. Whenever a distance between two points is calculated (for example, when computing the conflict), the actual metric is used. With modular implementation, this change affects the single function that calculates the distance between two points.

2. Step I of the layer-assignment breaks each net into 2-Nets by generating a minimum length tree that spans the net's terminals. This tree should be generated using the proper metric.

3. Step II of the layer-assignment generates the assignment graph for each 2-Net by defining via candidate locations along the shortest path between the end-points of the 2-Net. In the case of Euclidean distance, this path is unique and is the straight line between the end-points (Figure 44). When rectilinear or octilinear metrics are used, the shortest path may not be unique and the algorithm can choose one arbitrarily or choose a set of 'well spaced' paths. Considering multiple candidate paths[1] may result in better solutions as the assignment step has more choices in positioning the vias, at the expense of increasing the run-time. Our implementation considers only a single path, the straight line between the terminals, as done with Euclidean metric.

---

1. This can be handled by having multiple assignment graphs per 2-Net and searching all of them for a best assignment of the 2-Net, or by merging all of them into a single assignment graph.

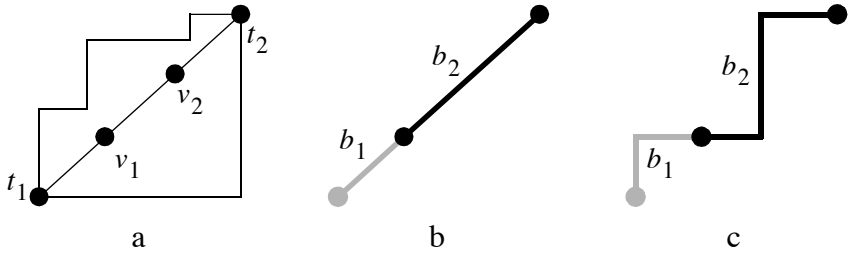


FIGURE 44 - Candidate via location for Rectilinear metric. The 2-Net between t1 and t2 is to be assigned and two candidate via locations v1, v2, are used (a). The shortest path using rectilinear metric between t1 and t2 is not unique and three choices are shown in (a). The LAA uses the straight line between the two terminals and spaces vias evenly along the path. In (b) a possible assignment with two branches, one on the gray layer and one of the black layer is shown. (c) shows a possible embedding for the assignment of that 2-Net. This embedding has the minimal possible rectilinear wire length between the two terminals. A similar approach is used for Octilinear metric.

SURF's implementation of the layer-assignment algorithm supports any-angle, rectilinear and octilinear routing models. It includes the above extensions except for the second one. When breaking the nets into two terminals nets, a Euclidean Steiner tree is generated regardless of the actual routing model in use. Generating a metric specific tree will potentially result with better solutions.

3.4.3 Preferring 2-Net assignment to layers

The constrained assignment technique mentioned earlier is able, by having a minor change to the cost function, to prefer assigning longer branches to the non constrained layers. This approach can be used to achieve many other goals of preferential assignment of certain branches to certain layers. A simple example is when it is preferable to assign a set of critical nets to some layers which have more desirable electrical properties. Other criteria can be preferring layers according to the general direction of the branch to achieve layers on which the branches are mostly in the same direction (for example, x/y pairs in Manhattan routing). These goals can be achieved by two modifications to the LAA. First, the cost function should be modified to preferred the desired

assignments, and second, the assignment graphs of the 2-Nets may need to be modified or extended to contain desire assignments of the 2-Nets (for example, having mostly horizontal/ vertical branches in a Manhattan one-layer-one-direction routing or having short branches in constrain assignment).

3.5 Experimental Results

Since the proposed LAA uses heuristics to perform the assignment, the primary technique to explore its merit and properties is experimentation. In this section we present empirical results of benchmarks of the LAA against another router and of routing with various parameters settings. To reduce external effects on the results, all experiments were done on a single bin so that they are independent of the performance of the Global Router. Nevertheless, to find the actual cost of the embeddings of the bins, a specific single-layer router had to be used. SURF's rubber-band single-layer router was used for this purpose.

3.5.1 Benchmark results

To evaluate the merits of the proposed LAA we compared the results of SURF to those of SLICE. SLICE is a grid-based router [29]. which is characterized by its simultaneous routing of all nets. This makes SLICE insensitive to net ordering as opposed to most existing area routers. Ten two-layer bin problems were routed by both SLICE and SURF and the results are shown in Table 1 and are summarized in Table 2. To make the routing conditions identical for both routers, the routing problems were modified to comply with SLICE 's input requirements which include rectilinear metric, and having terminals only on the top layer. The routing in SLICE was done using its default setup. To allow comparison of both wire-length and number of via, the via-cost parameter α of SURF was adjusted such that SURF's solutions had *both* shorter wires *and* less vias than SLICE's. As shown in Table 1, SURF was able to route the examples with slightly shorter wire length (0.4%) and with 31% fewer vias.

Example			SLICE			SURF		
Name	Nets	Len0	Len	Vias	Det%	Len	Vias	Det%
APEX/01	44	50195	52777	68	5.1	52166	42	3.9
APEX/02	70	113414	122461	174	8.0	121205	99	6.9
APEX/03	41	74157	74440	161	0.4	75337	106	1.6
DS15/01	19	6110	6194	35	1.4	6221	22	1.8
DS15/02	21	7693	8024	26	4.3	7914	25	2.9
DS15/03	22	8946	9204	12	2.9	9193	4	2.8
GDX/01	18	10057	10311	26	2.5	10268	20	2.1
GDX/02	26	12607	12691	26	0.7	12686	12	0.6
GDX/03	24	8970	9269	16	3.3	9184	14	2.4
GDX/04	29	9892	10155	12	2.7	10082	12	1.9
Avg	31.4	30204	31552	55.6	3.1	31425	35.6	2.7

TABLE 1 - Benchmark of SURF vs. SLICE. Ten bin-size problems were routed in rectilinear metric by both SLICE and SURF. Len0 denotes the lower bound on the wiring length and is the sum of the lengths of the rectilinear Steiner trees of the nets of each bin. Det% is the percentage of extra wire length in the bins embeddings compared to the lower bound. The improvement of SURF over SLICE is summarized in Table 2.

Example	Improvement	
Name	Len%	Vias%
APEX/01	1.16	38.24
APEX/02	1.03	43.10
APEX/03	-1.20	34.16
DS15/01	-0.44	37.14
DS15/02	1.37	3.85
DS15/03	0.12	66.67
GDX/01	0.42	23.08
GDX/02	0.04	53.85
GDX/03	0.92	12.50
GDX/04	0.72	0.00
Avg	0.41	31.26

TABLE 2 - SURF improvement over SLICE. This table summarizes the results in Table 1. It shows the improvement in the total wire length and via count when routing the ten bins in SURF. All values are in percents.

3.5.2 Balancing Between Wiring Length and Via Count

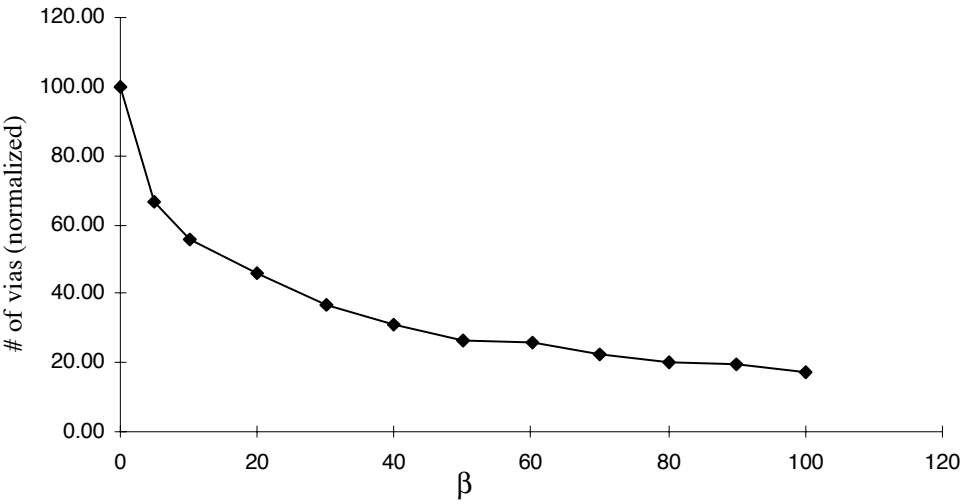
This experiment explores the effect of the via cost parameter β on the number of vias and the total wire length of the routed bins. Ten bins were routed (Euclidean metric, optimizations enabled) with twelve different values of β. The via count and wire length of the routed bins are compared in Table 3 and Table 4 respectively. The results are summarized in Figure 1 and Figure 2 which show the average via count and average wire length respectively, both of which are normalized to 100 for $\beta = 0$. As expected, the over-all trend is for the number of vias to decrease and the wire length to increase as β is increased. The results have several points in which increasing the value β reduces the wire length (for example, the increase from $\beta = 50$ to $\beta = 60$ in Graph 2). A possible cause of this behavior is the steepest descent optimization algorithm getting trapped in a local minimum and failing to find an optimal solution.

	0	5	10	20	30	40	50	60	70	80	90	100
APEX/01	99	71	47	43	39	35	31	31	29	23	21	21
APEX/02	245	167	136	99	89	77	67	67	61	59	59	59
APEX/03	220	118	90	75	57	58	54	54	52	50	48	48
DS15/01	56	36	32	28	22	20	18	16	14	12	12	10
DS15/02	56	51	41	39	34	22	14	14	14	14	14	12
DS15/03	35	27	25	18	16	12	10	10	8	8	8	8
DS15/04	44	27	23	16	13	13	11	11	11	11	11	7
GDX/01	31	17	17	10	10	6	6	6	6	6	4	4
GDX/01	24	18	18	18	7	7	6	6	3	2	2	0
GDX/02	27	15	10	10	8	8	6	6	4	2	2	2

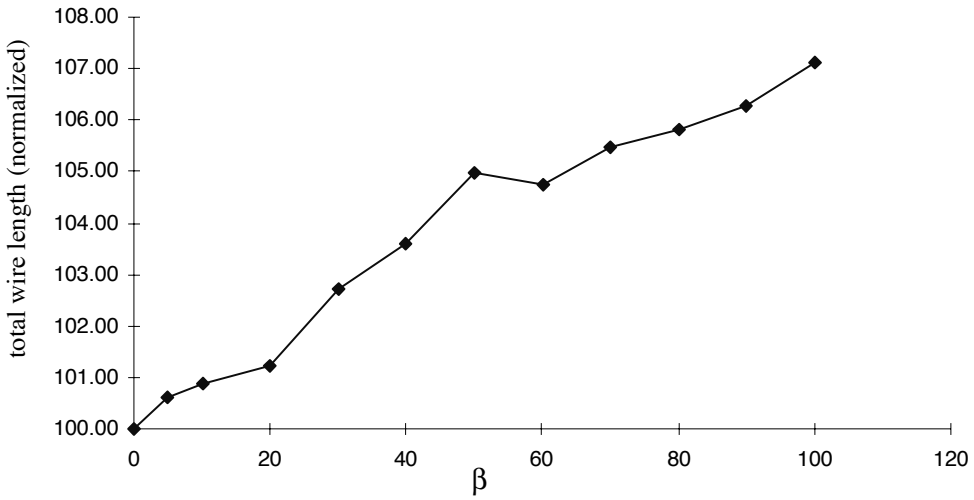
TABLE 3 - Number of vias vs. beta. This table shows the resulting number of vias when routing ten bins, each which twelve different values of β. As expected, the overall trend is having fewer vias when β is increased.

	E	0	5	10	20	30	40	50	60	70	80	90	100
APEX/01	2	442	443	445	445	448	451	453	453	455	471	469	469
APEX/02	3	103	104	104	105	107	111	113	113	112	110	110	110
APEX/03	2	658	667	669	666	693	692	693	693	694	683	692	692
DS15/01	1	528	530	530	533	544	547	551	549	562	561	560	566
DS15/02	1	688	689	530	533	544	547	551	549	562	561	560	566
DS15/03	1	793	794	797	801	804	812	819	819	819	819	819	819
DS15/04	1	918	922	928	930	934	934	938	938	938	938	938	983
GDX/01	2	97	97	98	98	103	103	103	103	103	103	105	105
GDX/01	1	815	815	815	815	833	833	847	847	857	866	866	899
GDX/02	1	834	836	840	840	844	844	894	894	900	910	910	910

TABLE 4 - Wire length vs. beta. The table shows the total wire length of the ten bins routed with twelve different settings of β. To fit in the table, the values were scaled down by 10^E. As expected, the overall trend is having shorter wires when β is decreased.



GRAPH 1 - Number of vias vs. beta. This graph shows the average number of vias of the ten bins in Table 3 versus the via cost parameter β. The via counts are normalized to 100 when $\beta = 0$.



GRAPH 2 - Wire length vs. beta. This graph shows the average wire length of the ten bins in Table 4 versus the via cost parameter β. The wire lengths are normalized to 100 for $\beta = 0$.

3.5.3 Estimated vs. Actual Detour

The cost function used by the LAA estimates the detour length using conflicts between pairs of components. This experiment explores the relationship between the estimated and the actual detours. Ten bins were routed (Euclidean metric, optimizations enabled) and the estimated and the actual detours were compared. To cover a wide range of conditions, each bin was routed with twelve different settings of the via cost parameter β, which has a significant effect on the detour (higher β values result in longer detours).

Table 5 shows the ratio of estimated to actual detours. For low values of β (which result in lower detour) the cost function overestimates the actual detour by a factor of about three. These ratios decrease as β increases (which also increases the detour length) and stabilizes around 1.4.

The convergence of the overestimation to 1.4 suggests pre-factoring of the cost function to compensate for that error. This will reduce the overestimation for low detour to the factor of two (2.9/1.4). This however has no practical benefit as the user specifies the via weight using the parameter β, which is an abstract value anyway. In addition, the large error ratio in assignments with low detour does not present a practical difficulty since the detour is only a small percentage of the total wire length and thus the error is small.

	0	5	10	20	30	40	50	60	70	80	90	100
APEX/01	1.6	1.6	1.7	1.9	1.7	1.5	1.6	1.6	1.6	1.2	1.5	1.5
APEX/02	1.5	1.4	2.0	1.5	1.0	0.9	0.7	0.7	0.9	1.1	1.1	1.1
APEX/03	2.4	1.4	1.5	2.0	0.9	1.0	1.1	1.1	1.1	1.6	1.3	1.3
DS15/01	2.6	2.5	2.8	2.4	1.2	1.2	1.2	1.5	1.2	1.5	1.4	1.5
DS15/02	2.6	2.4	1.9	2.0	1.2	2.4	1.9	1.9	1.9	1.9	1.9	1.9
DS15/03	3.3	3.1	1.3	1.8	1.7	1.5	1.5	1.5	1.9	1.9	1.9	1.9
DS15/04	2.4	1.6	1.2	1.3	1.9	1.9	2.2	2.2	2.2	2.2	2.2	1.8
GDX/01	2.3	1.7	1.7	1.2	1.2	0.6	0.6	0.6	0.6	0.6	0.7	0.7
GDX/02	7.9	4.0	4.0	4.0	1.1	1.1	1.3	1.3	1.3	1.3	1.3	1.2
GDX/03	2.1	1.7	1.8	1.8	2.0	2.0	0.6	0.6	0.9	1.1	1.1	1.1
Avg	2.9	2.1	2.0	2.0	1.4	1.4	1.3	1.3	1.4	1.4	1.4	1.4
Detour%	0.5	0.9	1.1	1.4	2.8	3.5	5.0	5.0	5.4	5.7	6.0	7.0

TABLE 5 - Detour overestimation vs. beta. This table shows the ratio of estimated to actual detour when routing ten bins with various values of β. Avg is the average of the estimated to actual detour ratios for each of the β settings and the Detour% is the average of the percentage actual detour from the basic wire length of the bins.

3.5.4 Comparison of Net Decomposition Methods

The LAA decomposes the multi-terminal nets into 2-Nets by generating a tree that connects the net terminals. Our LAA implementation supports three kinds of trees, minimal spanning tree (MST), minimal Steiner tree (MSTT), and Steiner tree with short edges collapsed (MSTTC). This experiment explores the differences in routing quality and run-time between these three methods. Fifteen bins, each with 25 nets of 4 terminals each were routed (Euclidean metric, optimizations enabled, $\beta = 25$). The locations of the terminals were selected randomly with uniform distribution of the x and y coordinates.

Table 6 shows statistics of routing the bins with the three methods. All the values in the table are averages for the routed bins. The most significant values are the Bin Actual Cost and LAA Time rows, which show the cost of the layouts and the run-time of the LAA respectively. These values

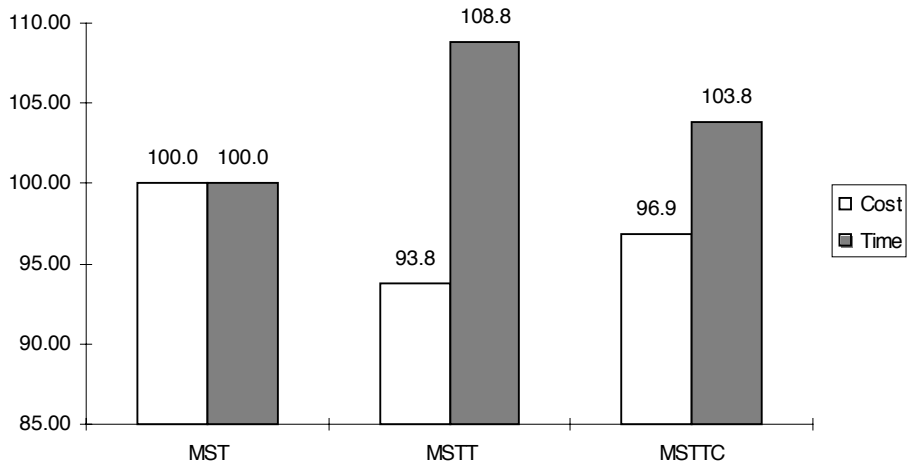
are also compared in Graph 3. As expected the MST has the shortest run-time, the MSTT has the best results (i.e. lowest cost), and the MSTTC provides a balance between the two.

The rest of the values in the table provide some insight into the operation of the LAA when using the three methods. The 2-Nets Intersections column shows the number of 2-Net pairs whose paths intersect. This value indicates the dependency between nets and is correlated to the amount of computation done on each iteration to update the incremental data. The LAA Iterations column shows the number of iterations needed to complete the bin assignment. 2NAA is the number of times the 2NAA algorithm is invoked during the bin assignment to compute a best assignment of a 2-Net. Finally, the Pair Conflicts column shows the number of pair-wise component conflicts computed during the assignment. The trends of all of these values when varying the decomposition method are similar to that of the run-time.

Note that all of the nets routed in this experiment had 4 randomly located terminals. When routing real-life examples, the results are expected to vary. For example, in the extreme case where all the nets have exactly two terminals, the three methods will give about the same results since the decompositions are identical.

	MST	MSTT	MSTTC
nets	25	25	25
2-Nets	75	125	78
2-Net Intersections	345	457	360
LAA Iterations	105	153	110
2NAA	1257	1688	1331
Pair Conflicts	96428	89061	1000865
Bin Actual Cost	2768	2596	2682
LAA Time [sec]	68	73	70

TABLE 6 - Comparison of 2-Net decomposition method. This table summarizes the results of routing 15 random bins with three methods of decomposing the nets into 2 Nets. All values are average for the 15 bins. Graph 3 provide a normalized comparison of the two last rows.



GRAPH 3 - Comparison of net decomposition methods. This graph shows the relative cost and run-when routing bins using the three decomposition methods. The values are based on the results in Table 6 and are normalized to 100 for the case of MST. As seen in the graph, the MST and the MSTT are on the two extremes of routing quality and LA run-time, while the MSTTC provides a balance between the two.

3.5.5 Candidate Via Density Versus Actual Cost

This experiment explores the effect that the number of candidate vias per 2-Net has on the routing results. Ten bins were routed (euclidean metric, $\beta = 5$, optimizations enabled) with the number of candidates varying from 1 to 10. Table 7 and Table 8 show the total wire length and number of vias respectively. Graph 4 shows the average actual cost of the routed bins and Graph 5 shows the average via count and wire length. All are normalized to 100 for the case of 10 candidate vias per 2-Net.

As expected, the actual cost decreases, when the number of candidate vias is increased (Graph 4). The improvement in the cost is significant for low numbers of candidate points and stabilizes around 5 candidates. Note that for low numbers of candidate vias, the average number of vias in the routed bins is significantly lower, resulting in longer wiring (Graph 5). One reason for this is that as the number of via candidates and the granularity of potential via locations is reduced, it becomes less likely that the LAA will consider vias that are “well placed” from the standpoint of

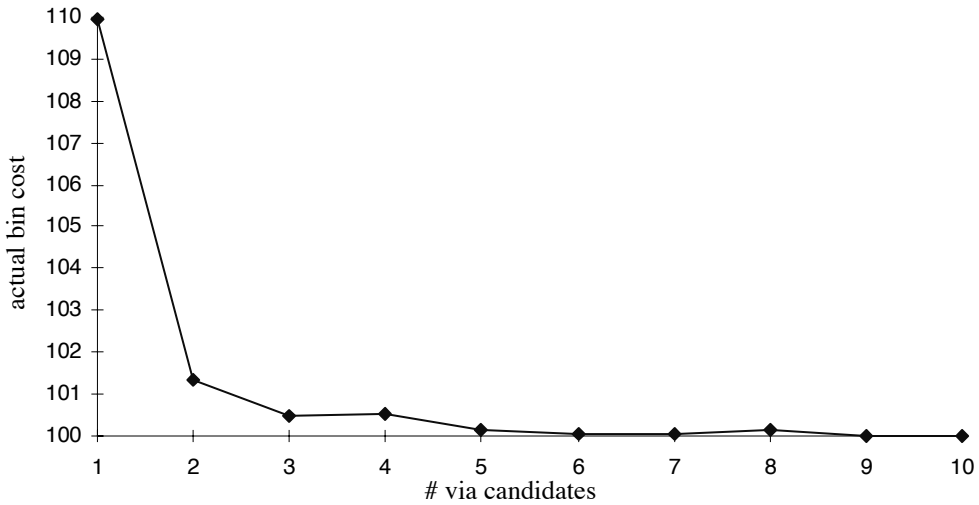
reducing the detour length. As a result, the expected detour reduction will decrease, making it harder to justify the additional via cost.

	E	1	2	3	4	5	6	7	8	9	10
APEX/01	2	515	448	446	443	444	443	441	440	442	441
APEX/02	3	132	105	104	104	103	103	103	104	103	103
APEX/03	2	712	668	667	667	660	659	658	656	656	656
DS15/01	1	595	533	531	530	530	529	530	529	528	528
DS15/02	1	756	713	692	689	687	685	684	684	683	683
DS15/03	1	873	805	795	794	795	794	793	794	794	793
DS15/04	1	977	930	923	922	918	917	917	919	919	919
GDX/01	2	107	98	97	97	97	97	97	97	97	97
GDX/02	1	836	819	815	815	815	815	815	815	815	815
GDX/02	1	841	836	839	836	838	836	836	836	836	836

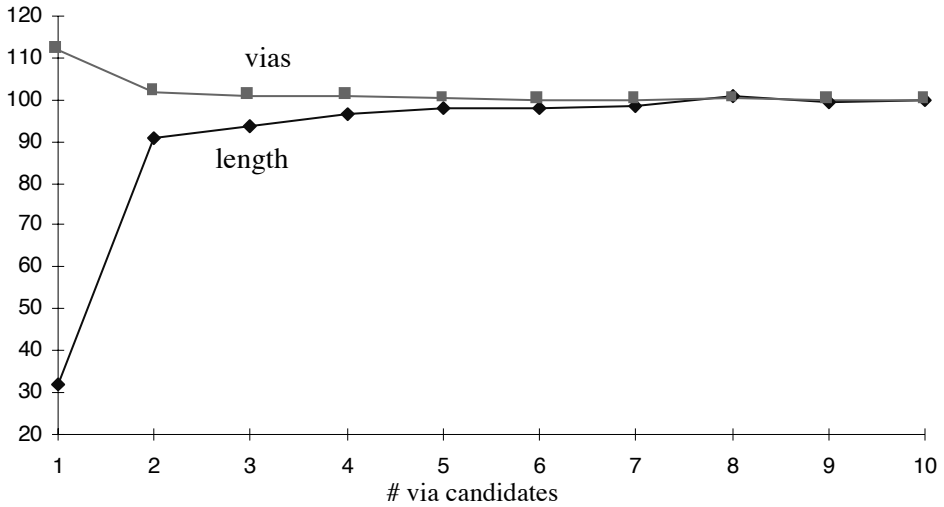
TABLE 7 - Wire length vs. number of candidate vias. The table shows the actual length of routing ten bins with ten different number of candidate vias per 2-Net. To fit in the table, the values have been scaled down by 10^E.

	1	2	3	4	5	6	7	8	9	10
APEX/01	3	49	61	71	73	75	77	81	73	77
APEX/02	70	135	142	167	163	169	173	177	164	175
APEX/03	61	111	115	118	122	124	124	127	130	126
DS15/01	6	43	40	36	40	36	36	36	36	40
DS15/02	6	37	44	51	53	54	52	58	54	52
DS15/03	3	23	29	27	26	26	24	24	28	26
DS15/04	9	21	22	27	27	26	24	24	28	28
GDX/01	3	17	17	17	17	18	18	18	16	16
GDX/02	8	23	22	18	18	18	18	18	18	18
GDX/03	12	17	13	15	14	14	14	14	14	14

TABLE 8 - Via count vs. number of candidate vias. The table shows the total number of vias when routing ten bins with ten different numbers of candidate vias per 2-Net.



GRAPH 4 - Actual cost vs. number of candidate vias. The graph is based on the data in Table 7 and Table 8. The costs of each bin has been normalized to 100 for the case of 10 candidate vias and the graph shows the averages of these costs. Note that the Y axis of the graph starts in 100 and not zero.



GRAPH 5 - Wire length and via count vs. number of candidate vias. The graph is based on the data in Table 7 and Table 8. The values have been normalized to 100 for the case of 10 candidate vias. Note that the Y axis of the graph starts in 20 and not zero

3.5.6 Using Various Routing Metrics

This experiment explores the relationship between the distance metric considered by the LAA and the final wire geometry style in terms of the effect on the actual cost of the embedding. Ten bins were considered. Each was routed with all four combinations of rectilinear and Euclidean distance metric and layout wiring style. The cost of these layouts are compared in Table 9. As expected, when the LAA uses a rectilinear metric to route Euclidean problems the cost is higher, on average by about 7%, than when performing the assignment with the Euclidean metric. However, when routing rectilinear problems, the average extra cost when performing the assignment using the Euclidean metric instead of rectilinear is significantly lower, only about 0.6%. A possible cause for this phenomenon is that a shortest Euclidean path is also a shortest rectilinear path, but not vice versa. Another possible explanation is that our implementation of the LAA does not take full advantage of the properties of rectilinear wiring since it only considers placing candidate vias along the straight path between the end-points of the 2-Net even though the shortest rectilinear path is not unique and other paths may yield better results.

	EE	ER	RR	RE	ER/EE	RE/RR
APEX/01	657.0	711.0	749.9	757.9	1.082	1.011
APEX/02	1464.4	1486.7	1557.6	1620.6	1.015	1.040
APEX/03	1148.7	1399.7	1496.6	1295.4	1.219	0.866
DS15/01	407.1	414.7	441.4	486.2	1.019	1.101
DS15/02	470.8	483.7	513.3	538.5	1.027	1.043
DS15/03	405.2	444.5	462.5	455.5	1.097	0.985
DS15/04	228.9	237.7	252.3	249.4	1.038	0.989
GDX/01	311.3	335.6	401.5	391.9	1.078	0.976
GDX/02	246.6	273.0	285.0	288.8	1.107	1.013
GDX/03	209.8	212.4	245.1	253.6	1.012	1.035
Average					1.069	1.006

TABLE 9 - Effect of the routing metric on the routing results. Ten bins has been routed for both euclidean and rectilinear metrics. For each metric, the LA was done with both 'right' and the 'opposite' metrics and the cost of the bin embeddings are compared. 'EE' denotes, an euclidean routing with euclidean LA, 'ER' denotes euclidean routing but with rectilinear LA, etc. The columns ER/EE and RE/RR show the ratios of the respective columns.

3.5.7 LAA Scalability

This experiment explores the scalability of the LAA in terms of the number of nets per bin. Ninety bins, grouped into nine different bin sizes, were assigned and the run-times of the LAA compared. Half of the nets of each bin had two terminals and half had three terminals. The average number of 2-Nets per net was about 2[1]. The terminals were placed randomly with uniform distribution over the X and Y axis. The statistics for the size groups are shown in Table 10. The values of variable shown in each of the table columns were fitted by a polynomial function of the form an^b where n is the problem size in terms of nets. The order b of the best fitted function is shown in the last row of the table.

1. Since Steiner tree decomposition has been used, the 3 terminal nets were typically decomposed into three 2-Nets and the two terminal nets into a single 2-Net.

Nets	2-Nets	LAA Iterations	2-Net Intersect.	2NAA	Exp. Branch Edges	Pair Conflicts.	Time [Sec]
1	3	5	3	9	54	0	0.03
5	11	13	15	42	307	175	0.19
10	20	23	34	92	712	743	0.60
20	40	46	104	277	2458	5298	3.38
35	71	87	339	932	11546	47507	25.34
50	100	127	617	1743	25164	135677	69.72
60	119	149	783	2113	30059	185045	97.74
80	158	212	1457	4399	69091	617100	311.56
100	199	274	2211	6855	121061	1227431	6845.58
P	n^1	$n^{1.1}$	n^2	n^2	$n^{2.5}$	$n^{3.5}$	$n^{3.5}$

TABLE 10 - LAA scalability. The table shows the statistics of running the LAA over 90 bins. The bins are of nine different sizes from 1 to 100 nets per bin. All values are averages for the bins in each size group. 2-Net Intersection are the number of pairs of 2-Net whose paths of candidate vias intersect. 2NAA is the number of time the 2NAA algorithm has been invoked during the LA of a bin. Expanded Branch Edges is the number of branch edges expanded by the 2NAA. Pair Conflict is the number of time a component pair-wise conflict was computed while assigning a bin. The last row shows the order of growth of each of the variables as a function of the bin size n when fitted by a polynom.

The results indicates that the run-time of the LAA, in the tested range of bin sizes, grows proportionally to $n^{3.5}$ where n is the number of nets in the bin. As seen in Table 10, the number of 2-Nets grows linearly with the bin size, and the number of LAA iterations grows 'almost' linearly', about $n^{1.1}$. As expected, the number of intersections between 2-Nets grows proportionally to n^2 which suggests that a 2-Net intersects on average a number of 2-Nets proportional to n. As as a result, in every iteration, after a 2-Net has been assigned, the number of 2-Nets whose best assignment is computed is proportional to n, and this is confirmed by the growth in the number of invocations of the 2NAA which is proportional to n^2. The LAA run-time and the computations of the pair-wise conflicts grow proportionally to $n^{3.5}$ which confirms that, as expected, the computation of a single pair-wise conflict is independent of the problem size. The

ratio of computed pair-wise conflicts to the number of 2NAA invocations grows proportionally to $n^{1.5}$. This is because the number of branch edges expanded during each 2NAA grows proportionally to $n^{0.5}$, and because the average number of components intersecting an expanded branch edge with grows linearly with n. The growth in the number of branch edges expanded during each 2NAA is attributed to the increased density of nets which increases the chance of a conflict between components and thus causes the 2NAA to explore more paths in the assignment graph until a shortest path is found. Since the number of branch edges in a 2-Net assignment graph is upper- bounded by the graph size and is independent of the bin size, the growth of number of expanded branch edges during the LAA would grow proportionally to n^2 when the bins are large enough and this will result in run-time proportional to n^3. This however happens only in bins which are too large to be practically solved by the LAA and thus we consider the practical run-time of the LAA to grow proportionally to $n^{3.5}$.

3.6 Conclusion and Future Work

We have considered the problem of layer-assignment. That is, decomposing a multi-layer routing problem into a set of single-layer sub-problems. The problem is a key building block in our topological rubber-band based router. We have presented a simple and efficient cost function which can be used to predict the wire length of the actual embeddings without the need to perform explicit routing, and proved that the cost is finite if and only each of the sub-problems is planar. This cost function allows us to formulate the layer-assignment problem as an optimization problem. We presented the Layer-Assignment Algorithm (LAA) that solves this optimization problem and an original algorithm (2NAA) which finds an optimal assignment for a given two terminal net, and proved that the LAA is guaranteed to terminate and to find a planar solution if such a solution exists. Experimental results show that the run-time of the LAA is proportional to $n^{3.5}$ where n is the number of nets in the bin. We demonstrated the flexibility of LAA by showing two examples of how a simple change to the cost function can achieve various routing goals such as different metrics of the wire length and constrained (one and a half layer) routing. We presented experimental results of our implementation of the LAA that support the merits of the algorithm and give insight into some of its properties.

Our implementation of the LAA uses a steepest descent optimization technique which tends to be sensitive to local minimums. A more complex technique such as simulating annealing, or group migration, can possibly improve the quality of the solutions. Another possible improvement is using a cost function that considers conflicts between larger sets of components, rather than only pairs. This may better capture the dependency between conflicts and give more accurate detour estimation at the expense of longer run-time. The LAA can also be extended to consider a wider range of candidate vias for each 2-Net, to better support existence of obstacles in the routing area, to consider congestion and routability, and to handle critical nets and electrical requirements.

4 TOPOLOGICAL NET ORDERING

4.1 Introduction

The Layer-Assignment Algorithm (LAA) generates for each bin a set of single-layer routing problems, one for each layer. These bin layers are routed independently by the single-layer router which embeds the nets to form a rubber-band sketch. The routing is done one two-terminal-net (2-Net) at a time, each routed on a least cost path between its end-points in the rubber-band sketch. Since the order in which the 2-Nets are routed affects the planarity and final wiring length, it is important to order the nets so that a planar routing with minimal wire length can be achieved.

The rest of this chapter deals with the problem of ordering the 2-Nets. First we discuss the decomposition of the nets into 2-Nets and the goals and limitations of 2-Net ordering. Then we present the two components of the proposed ordering algorithm, the wire length minimization, and the Planarity Enforcement Operator (PEO). These two components are combined to achieve a planar order which results in low wiring length. The wire minimization algorithm orders the 2-Nets to reduce the wire length and the PEO transforms that order into an order that guarantees a planar embedding. The problem of ordering the 2-Nets to minimize the wire length is formulated as an optimization problem. The cost function estimates the wiring detour in the embedding and is based on ordered pair-wise conflicts between nets. Then, an efficient heuristic to solve the optimization problem is presented. Finally we present experimental results of our implementation of the proposed algorithms.

4.2 Decomposition into 2-Nets

The LAA defines for each layer a set of components, each of which is a set of terminals to be interconnected within the bin boundary and on that layer[1]. The terminals include the original terminals of the design as well as vias and bin boundary crossing points.

---

1. In the context of a single layer routing problem the components can be viewed as *nets*. However, for consistency, we will adhere to the notation in the Layer-Assignment chapter and will call them *components*.

Since the embedding is performed a 2-Net at a time, the components need to be decomposed into 2-Nets. This decomposition can be done in several ways, including the methods described in the Layer-Assignment chapter. The decomposition does not have to be the same as the one used by the LAA. Using a different decomposition may even improve the routing results since the new decomposition can consider the results of the LAA, information which was not available when the LAA decomposition was performed. In our implementation, for sake of simplicity, we use the same decomposition is used by the LAA.

4.3 2-Net Ordering

Since the order in which the 2-Nets are routed affects the wiring length of the embedding, (Figure 45) an order that minimizes the wiring length is desired. Note that number of vias is not affected by the 2-Net routing order since the vias are determined by the layer-assignment step.

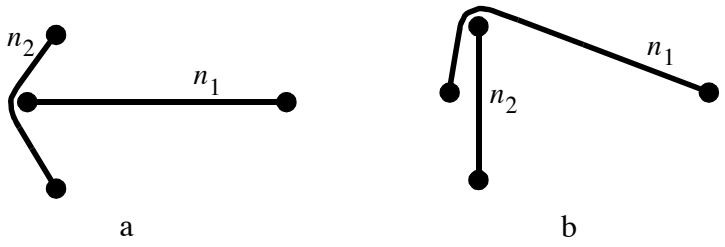


FIGURE 45 - Net ordering and wiring length. The two 2-Nets in this example were routed in two different orders. In (a) n_1 was routed before n_2 and in (b) n_1 was routed after n_2. The order in (a) results in shorter wiring.

Routing the 2-Nets one at a time on a least cost path between their end-points has its limitations. In some cases, no order will result in an optimal embedding. Figure 46 shows an example problem that cannot be solved optimally, for any net ordering followed by a shortest path search. SURF handles this limitation by using a Rip-Out-And-Reroute (ROAR) post-processor (described elsewhere in this document) that reduces the wiring length. The ROAR achieves the optimal solution for the problem in Figure 46.

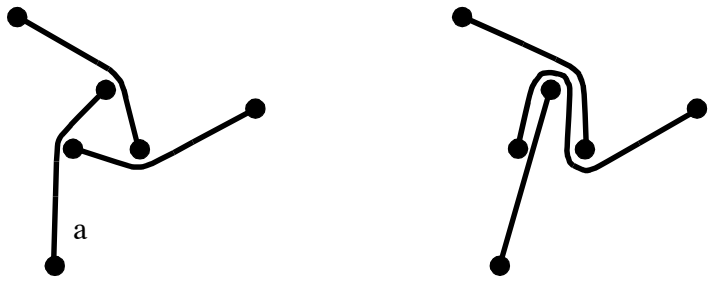


FIGURE 46 - The Triangle problem. This problem has three symmetrical 2-Nets and its optimal embedding is shown in (a). The optimal embedding cannot be achieved by routing 2-Net-at-a-time on a shortest path. Any such routing will result in a sub-optimal solution similar to the one in (b). SURF handles this limitation by post-processing the sketch with the ROAR optimizer.

In addition to minimizing the wiring length, the routing order needs to guarantee that a planar embedding can be achieved. Figure 47 shows an example where a 2-Net cannot be routed because its two end-points are disconnected by previously routed 2-Nets and the bin boundary. Such non-planarity can happen only in the presence of nets that connect terminals located on the bin boundary (*external* terminals). The 2-Nets should be routed in an order that guarantees that no connections are blocked by nets routed earlier in the sequence.

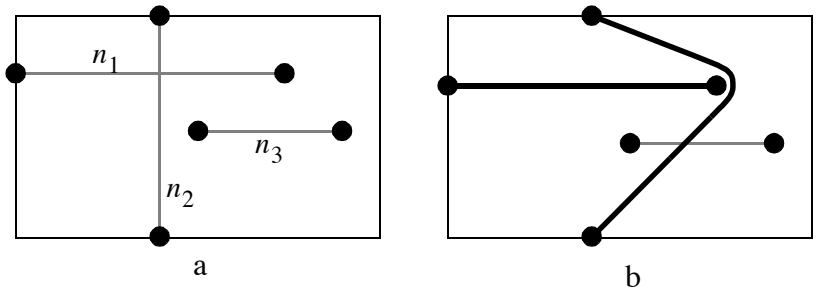


FIGURE 47 - Non-planar 2-Net routing order. The single-layer routing problem in (a) has three 2-Nets. Routing the 2-Nets in the order n_1 , n_2 , n_3 fails because a planar routing of n_3 is impossible as shown in (b). Routing the 2-Net for example in the order n_1 , n_3 , n_2 is guaranteed to succeed.

The proposed ordering is done in two steps

- First, the 2-Nets are ordered to minimize a cost function which estimates the wiring length when embedding the 2-Nets in that order. This minimization step accepts the (unordered) set of the bin layer 2-Nets and generates a complete order of the 2-Nets. This order however does not guaranty successful planar embedding.
- The minimizing order is then constrained to guarantee planarity. This is done by the *Planarity Enforcement Operator* (*PEO*, or *PE operator*) which accepts a 2-Net ordering and generates 2-Net order which is based on the input order and guaranties successful planar embedding. The order modification that the PEO does is guaranteed to preserve the wire length estimation of the order.

These two steps are presented in the rest of the chapter. Since the minimization step considers the operation of the PEO performed on its output, we present the PEO first, followed by presentation of the minimization step.

4.4 Planarity Enforcement Operator (PEO)

The PEO accepts an order of the 2-Nets and generates an order that guarantees successful planar embedding of the 2-Nets. The PEO does not make any specific assumption about the algorithm used to embed the individual 2-Nets except that it is guaranteed to find a planar path if one exists.

The PEO is based on the classification of 2-Nets into ‘closed’ and ‘open’ 2-Nets:

(closed and open 2-Nets) A 2-Net is said to be *closed* in a given order of 2-Nets if it completes a path between two external terminals when considering the 2-Nets preceding it in the order (Figure 48). Otherwise the 2-Net is said to be *open* in that order.

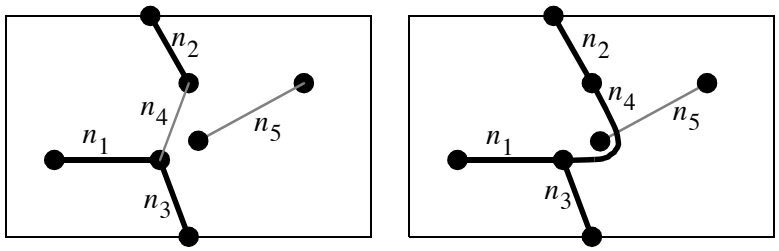


FIGURE 48 - Closed 2-Nets. The 2-nets in this example are ordered as $n_1..n_5$. The only 2-Net which might disconnect the end-points of non routed 2-Net is n_4 which closes a path between two external terminals (a). (b) shows how a *possible* path of n_4 can separate the end-points of n_5.

Note that the definition ignores any specific embedding of the 2-Nets and considers only the 2-Net order and connectivity

The significance of identifying the closed 2-Nets is that they partition the routing area into disconnected regions and could potentially separate the end-points of an un-routed 2-Net. The planarity enforcement is done by identifying the two subsequences of 2-Nets open and closed respectively in the order and then forming a new order which is a concatenation of the subsequence of the open and closed 2-Nets respectively (Algorithm 2). The PEO preserves the relative order of the 2-Nets closed in the input order and the relative order of the 2-Nets open in that order and positions all the 2-Nets closed in the original order after all the 2-Nets open in the original order. The PEO modifies the order only if the input order contains a closed 2-Net followed by an open 2-Net, otherwise it preserves the input order. Note that the fact that a 2-Net is closed (open) in the input order, does not intuitively imply that it is also closed (open) in the output order since the order has been modified. This however is true and later we show that the PEO preserves the closeness (openness) of the 2-Nets (which implies that every result of the PEO is also a fix point of it).

Let π be the input order of 2-Nets
Let $\pi_o = n_1..n_j$ be the subsequence of open 2-Nets in π
Let $\pi_c = m_1..m_k$ be the subsequence of closed 2-Nets in π
The output order is $\pi^* = n_1..n_j, m_1..m_k$

ALGORITHM 2 - The Planarity Enforcement operator. The PEO identifies the two subsequences of 2-Nets open and closed respectively in the input order and generates an order in which the 2-Nets closed in the input order follows the 2-Nets open in the input order. The PEO preserves the relative order of the open and closed 2-nets respectively.

To prove that the output of the PEO guaranties planar embedding, we first prove a sufficient condition for a planar embedding of an ordered set of 2-Nets:

Lemma 3 Let $\pi = n_1..n_k$ be an order of a planar set of 2-Nets[1]. If π does not have a close 2-Net followed (either immediately or not) by an open one then routing the 2-Nets in order by π is guaranteed to result in a planar embedding.

Proof We prove this by contradiction. For the purpose of contradiction we assume when routing the 2-Nets in order π, the embedding of a 2-Net n_i (the first one) failed because there is no planar path between its end-points. This implies that the end-points of n_i are separated from each other by previously routed 2-Nets, and therefore, at least one closed 2-Net has already been routed (open 2-Nets do not disconnect the routing area). Since open 2-Nets do not follow closed ones in π, n_i is closed, and therefore if it would be routed this would close a path between two external terminals a, b (Figure 49). Since the end-points of n_i are disconnected, the previously routed 2-Nets forms a path between two external terminals c, d that separates between the end-points of n_i. This path also separates a and b. This implies that there are two crossing paths between external terminals (Figure 49) and therefore the routing problem cannot have a planar embedding. This contradicts the planarity of the set of 2-Nets. Q.E.D

1. In the context of SURF, the planarity of the set is guaranteed by the Layer-Assignment Algorithm.

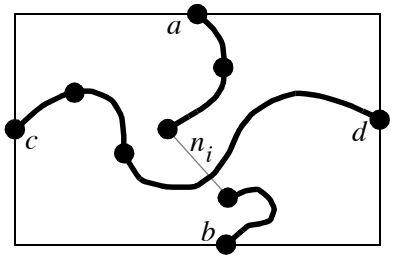


FIGURE 49 - Non planar external paths. n_i is a closed 2-Net that is disconnected by previously routed 2-Nets and therefore the 2-Net set includes paths between external 2-Nets which crosses each other. This implies that the routing problem is non-planar.

Corollary 1 An open 2-Net cannot be blocked by previously routed open 2-Nets. A closed 2-net cannot be blocked by any 2-Net (assuming planarity of the 2-Net set). The only possibly blockade is of an open 2-Net by a closed one.

In the general case, the sufficient condition of Lemma 3 is necessary as a path of the first closed 2-Net can be constructed such that it separates the end point of an open 2-Net following it (Figure 50). However, if some assumptions about the 2-Net embedding algorithm are made (for example 2-Net embedding on a shortest planar path), this condition may not be necessary for planar embedding.

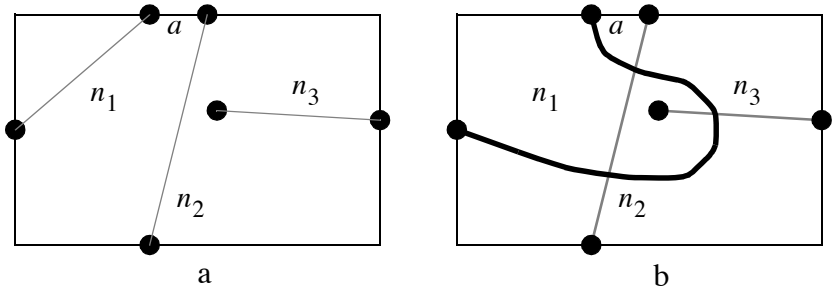


FIGURE 50 - Necessary condition for planar embedding. The example in (a) has 3 2-Nets ordered as n_1, n_2, n_3. The 2-Nets n_1 and n_2 are closed in that order and n_3 is open. If the 2-Net embedding algorithm can potentially select any planar path (and not necessarily the shortest one), it can choose for n_1 (the first closed 2-Net) a path which disconnects open 2-Nets following it (n_3 in this example) as shown in (b). Note that n_2 (which is a closed 2-Net) cannot be blocked by n_1.

Next we will prove that the output of the PEO satisfies the sufficient condition in Lemma 3. The planarity of the 2-Net set is guaranteed by the LAA algorithm which generates this set and therefore it is sufficient to show that the output of the PEO does not have an open 2-Net followed by a closed 2-Net. We show that by proving that the order modification done by the PEO does not affect the closeness/openness of the 2-Net. In other words, a 2-Net that was closed (open) in the input order of the PEO is also closed (open) in the output order. The proof is done in two steps, first we show that the closeness/openness of the 2-nets is preserved by a swap operation between a closed 2-Net and by an open 2-Net immediately following it, and then we show that the output order of the PEO can be achieved by performing a finite sequence of swapping operation of pairs of adjacent 2-Nets, closed and open respectively in the input order.

Lemma 4 Let $\pi_1 = n_1..n_k$ be an order of the 2-Nets, let n_i, n_{i+1} be two successive 2-Nets closed and open respectively in π_1, and let π_2 be the order π_1 with the 2-Nets n_i, n_{i+1} swapped. A 2-Net is closed (open) in π_2 if an only if it is closed (open) in π_1.

Proof For every 2-Net n_j other than n_i and n_{i+1}, the sets of 2-Nets ahead of it in π_1 and in π_2 respectively are identical therefore its closeness/openness is preserved. n_i is closed in π_1 and by definition it closes a path between two external terminals. This path is composed of n_i and

possibly some of the 2-Nets $n_1..n_{i-1}$ preceding it in π_1. Since these 2-Nets also precede n_i in π_2, n_i closes an external path in π_2 and is closed in π_2. As for n_{i+1}, it is open in π_1 and by definition, it does not close an external path in π_1 (considering the 2-Nets $n_1..n_i$ preceding it in π_1). This 2-Net set $\{n_1..n_i\}$, is a super set of the 2-Nets $\{n_1..n_{i-1}\}$ preceding n_{i+1} in π_2, and therefore n_{i+1} does not close an external path and is open in π_2. Q.E.D

Lemma 5 Let π and π^* be the input and output orders respectively of the PEO. A 2-Net is closed in π^* if and only if it is closed in π

Proof π can be transformed into π^* by a finite sequence of sweepings between pairs of adjacent 2-Nets, the first and the second are closed and open respectively in π (note that the closeness/ openness of the 2-Nets is determined in regard to π and not the intermediate orders between the swap operations). The swapping ends when no such pair is found and the resulting order is exactly π^*. by Lemma 4, each such swap preserves the closeness/openness of the 2-Nets, and therefore a 2-Net is closed (open) in π^* if and only if it is closed (open) in π. Q.E.D.

Theorem 3 If the set of 2-Nets is planar then the output order of the PE operator is guaranteed to result in successful planar embedding.

Proof Let π and π^* be the input and output order s respectively of the PEO. By the way π^* is constructed (Algorithm 2), it does not contain pairs of successive 2-Nets, the first of them is closed in π and the second is open in π. By Lemma 5 this also holds when the openness/closeness is determined by the order π^*, and by Lemma 3 the π^* guaranties planar embedding. Q.E.D.

We have shown how an arbitrary order can be made planar by the PE operator. Next will present how an order which reduces the detour is achieved.

4.5 2-Net Ordering Problem (2NOP) Formulation

The 2-Net Ordering Problem (2NOP) could be defined as follows: given a set of 2-Nets, find an order of the 2-Nets that minimizes the wiring length when it is made planar by the PEO and then routed one 2-Net at a time. This definition however would require running the embedding

algorithm during the optimization process, which would require intensive computation. To keep the run-time of solving the 2NOP practical, we use a cost function that estimates the total detour length of the wiring. This function is similar to the one used by LAA and is based on pair-wise conflicts (defined below) between components. However it differs from the LAA function in that it considers a specific 2-Net routing order while the LAA function assumes the order that yields the minimal detour.

Definition 10 (Ordered Pair-Wise Conflict) Let c_1, c_2 be components, let $\pi = n_1 \ldots n_j$ be an order of the union of the 2-Nets of c_1 and c_2, let π_p be the order π after planarity enforcement by the PEO, and let S be the embedding of the 2-Nets when routed in the order π_p. The conflict of c_1, c_2 in order π is the total length of the wiring in S minus the basic lengths of the components c_1, c_2. This conflict is denoted as $H(c_1, c_2, \pi)$. If the 2-Nets have no planar embedding, we say that $H(c_1, c_2, \pi)$ is infinite[1].

The cost of an order of 2-Nets is the estimated total detour when the 2-Nets are routed by the order after planarity enforcement. This cost is defined as follows:

Definition 11 (Order Cost) let $\pi = n_1 \ldots n_k$ be an order of the 2-Nets of a bin layer. The cost of π is the sum of the pair-wise ordered conflict of all the component pairs:

$$C(\pi) = \sum_{\{c_i, c_j\}} H(c_i, c_j, \pi_{ij}) \tag{13}$$

where:

- $\{c_i, c_j\}$ is a pair of components, $i \neq j$.
- π_{ij} is the union of the 2-Nets of c_i and c_j ordered in the same order as they are in π.

Figure 51 shows an example of an order cost. In this example, the estimated cost is equal to the actual detour. In the general case however, similar to the estimation function used by the LAA, the

1. Since the output of the LAA is guaranteed to be planar, the routing is guaranteed to succeed (*Theorem 3*). In the rest of the chapter we assume that the pair-wise ordered cost is always finite

estimated detour is not an upper nor a lower bound on the actual detour, nor is its error bounded by a constant.

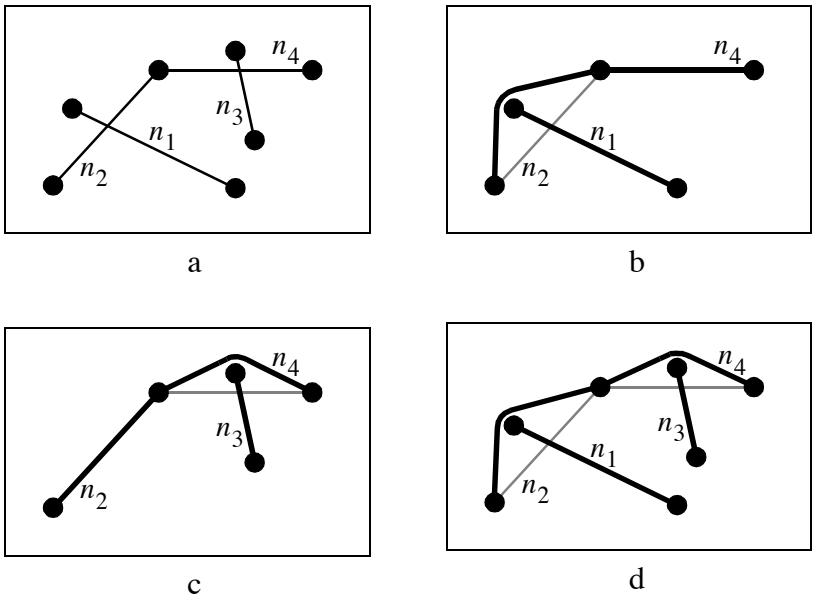


FIGURE 51 - 2-Net order cost. The 2-Nets in (a) form three components and are ordered as n_1, n_2, n_3, n_4. The cost of this order is the sum of the ordered conflicts of the components pairs. (c) and (b) show the ordered conflict of two pairs of components. The conflict of the third pair is zero. (d) shows the total conflict when routing the 2-Nets in this order. In this example, the estimated detour is equal to the actual one.

Using the order cost function, the 2-Net ordering problem is defined as:

Definition 12 (2-Net Ordering Problem - 2NOP) Given a set of 2-Nets $n_1..n_k$ of a bin layer , find an order π of the 2-Nets that minimizes the cost $C(\pi)$.

In the following sections we discuss the complexity of the 2NOP and propose an heuristic to solve it but first we discuss an important property of the PEO related to the order cost function defined above. The PEO, while enforcing the planarity of an order, preserves its cost. That is, the cost of the output order of the PEO is exactly the cost of the input order. This property implies that while transforming the output of the 2NOP by the PEO, things are not getting worse.

Lemma 6 Let π be an order of the 2-Net and π_1 be a subsequence of it that contains entire components (i.e. if a 2-net is in π_1 then all the 2-net of its components are also in π_1) then a 2-Net is closed (open) in π_1 if and only if it is closed (open) in π.

Proof Let n be a 2-Net in π_1. If it is closed in π then it closes an external path with possibly some of the 2-Nets ahead of it in the order. Any such 2-Net (i.e. on the closed path) is of the same component as n and therefore it is also in π_1, and since it is ahead of n in π, it is also ahead of n in π_1. This implies that n closes the same path in π_1 and therefore it is also closed in π_1. In a similar way, a 2-Net of π_1 that is open in π is also open in π_1. Q.E.D.

Theorem 4 The Planarity Enforcement operator preserves the cost of the 2-net order.

Proof Let π and π^* be the input and output orders respectively of the PEO. Since π and π^* contain exactly the same set of components it is sufficient to prove that the ordered pair-wise cost of every component is identical in π and π^*. Let c_1 and c_2 be two arbitrary components in π and let set c_{12} be the union of the 2-Nets of c_1, c_2. Let π_a be the order in π of c_{12}, and let $\pi_a{}^*$ be the order π_a after operated by the PEO. In a similar way, let π_b be the order in π^* of c_{12}, and let $\pi_b{}^*$ be the order π_b after operated by the PEO. By the definition of the pair-wise cost, it is sufficient to prove that the wire length when routing the 2-Nets c_{12}, in order $\pi_a{}^*$ is the identical to the wire length when routed in order $\pi_b{}^*$. We will prove that by showing that $\pi_a{}^* = \pi_b{}^*$. First we prove that each 2-Net in π^*, π_a, π_b, $\pi_a{}^*$, and $\pi_b{}^*$ is closed in that order if and only it is closed in π. For π^* this holds because the PEO preserves the closeness/openness of the 2-Nets (see Lemma 5). For π_a and π_b, this holds by Lemma 6 and the fact that (as we have just shown) the closeness/openness in π^* is identical to π, and since it holds for π_a, π_b, by Lemma 5 it holds also for $\pi_a{}^*$, $\pi_b{}^*$. The orders $\pi_a{}^*$, $\pi_b{}^*$. contain exactly the same 2-Nets (i.e. c_{12}) and since their 2-Nets have the same closeness/openness as in π, a 2-Net is open (closed) in $\pi_a{}^*$ if and only if it is open (closed) in $\pi_b{}^*$. In addition, $\pi_a{}^*$, $\pi_b{}^*$ are both results of the PEO, and therefore they contain the closed 2-Nets after the open ones. The relative order of the closed (open) 2-Nets in $\pi_a{}^*$ is identical to their relative order in π_a (the relative order is preserved by the PEO) which is identical to the one in π (the order is preserve when extracting a subsequence). In a similar way,

the relative order of the closed (open) 2-Nets in π_b* is identical to their relative order in π_b (preserved by the PEO) and therefore to their order in π (preserved when extracting a subsequence). π_a* and π_b* have exactly the same 2-Nets, the same sets of closed and open 2-Nets, the same relative order of closed and open 2-Net respectively, and the all closed 2-Nets following the open ones, and therefore they are identical. Q.E.D.

Corollary 2 For any given order of the 2-Nets, there is an order of the same cost which guaranties planar embedding. Such an order can be found by the PEO.

4.6 2NOP Complexity

Let 22NOP be the sub-problem of 22NOP in which each component consists of a single 2-Net. Since 2NOP contains 22NOP as a special case, we know that its complexity is at least that of 22NOP. The 22NOP can be viewed as a minimization of the sum of a lower triangle of a square matrix using simultaneous permutations of rows and columns:

Definition 13 (Matrix Permutation Problem - MPP) Given an $n \times n$ matrix $A = (a_{ij})$ with non-negative real values, find a matrix $B = (b_{ij})$ obtained from A by simultaneously row and column permutations which minimizes $\sum_{1 \le i < j \le n} b_{ij}$.

The 2NOP is mapped to the MPP by setting n to the number of components and setting a_{ij} to the ordered pair-wise conflict $H(n_i, n_j, n_i n_j)$ when $i \neq j$ and to zero when $i = j$.

The MPP is NP-hard since the maximum Likelihood Ranking Problem (LRP) which is known to be NP-C [17], can be reduced to it by setting $a_{ij} = max(0, -a_{ij})$ where a_{ij} and a_{ij} are the matrix entries of the MPP and the LRP respectively. It is not clear though if a MPP can be reduced back to a 22NOP or if the geometrical characteristics of the pair-wise conflicts used by the 22NOP have special properties that cause the 22NOP to have lower complexity than the MPP. Considering this, the question of the complexity of the 22NOP and the 2NOP is still open.

4.7 Solving the 2NOP

The problem of finding a minimal cost order can be solved by various optimization techniques such as Simulated Annealing [30], Group Migration, and Genetic Algorithms. We have chosen to implement a greedy algorithm which is sufficient to show the merits of the proposed cost function. More advanced optimization methods may result in better ordering.

The proposed minimization algorithm considers only non-interleaving permutations of the 2-Nets. These are permutations in which the 2-Nets of each component are grouped together. This restriction may eliminate optimal solutions as shown in Figure 52, but on the other hand, it simplifies the solution by performing the ordering in two steps, first ordering the 2-Nets within each component, and then ordering the components themselves.

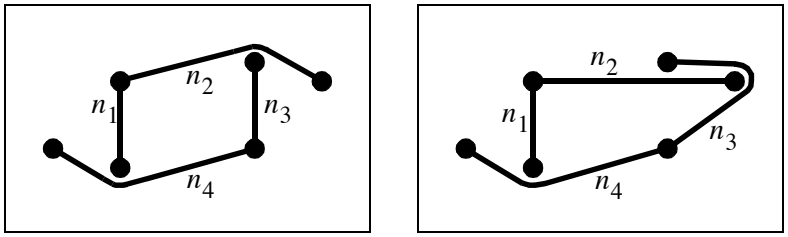


FIGURE 52 - Limitations of non-interleaving order. (a) shows an embedding of two component with two 2-Nets each. The 2-Nets were routed in the interleaving order n_1, n_3, n_2, n_4 which minimizes the cost. Any non-interleaving order will results in higher cost such as the order in (b)

Ordering the 2-nets within each components can be done in several ways. This order however is less critical then the component order as any conflicts between 2-Nets of the same component can be resolved by adding a Steiner point at the crossing point and removing redundant wires to eliminate cycles in the component. Therefore, our implementation uses an arbitrary order of the 2-Nets within each component.

The component ordering is performed by transforming the component ordering problem to an instance of MPP and then solving the MPP. The MPP instance is constructed as follows, n is the

number of components, the $n \times n$ matrix is $A = (a_{ij})$ where $a_{ij} = H(c_i, c_j, \pi_i\pi_j)$ when $i \neq i$ and $a_{ij} = 0$ when $i = j$. π_i and π_j are the internal orders of the 2-Nets of components c_i and c_j respectively and $\pi_i\pi_j$ is the concatenation of π_i and π_j. Intuitively, the value of a_{ij} represents the cost of routing component c_i before component c_j. When the MPP problem is solved, the permutations of its rows in reverse defines the component routing order.

Since the MPP is NP-Hard, we cannot expect to have a practical optimal algorithm for it and therefore we use heuristics instead (Algorithm 3). The algorithm uses greedy approach to iteratively pick rows (and matching columns) from the input matrix for the output matrix. The rows are picked in descending order in the solution matrix such that the last row of the solution is picked first and the first row is picked last. On each iteration the algorithm selects a row with minimal sum in the not selected yet columns. The iteration continues until all rows have been selected. The algorithm picks components in the order they will be routed. On each iteration it chooses a component with lowest sum of conflicts with the non selected components.

```
let A = (a_ij) be the n × n matrix of the MPP
let T = A
for k = 1..n do {
    let m = n – k + 1 .        // m is k in reversed order
    let S_i be the sum of row i of T , 1 ≤ i ≤ n
    select a not selected yet j , 1 ≤ j ≤ n , which minimizes S_j
    let b_m = j
    set to zero the entries of row and column j of matrix T
}

The solution is matrix A with permutation b_1..b_n of rows and columns
```

ALGORITHM 3 - Algorithm to solve the MPP. This greedy algorithm iteratively picks from the input matrix the rows and matching columns of the solution matrix from last to the first. On each iteration, it picks a row that will increase the least the sum of the lower triangle of the solution matrix.

The heuristics used to solve the 2-NOP which include considering only non-interleaving orders, fixing in advance the order of the 2-Nets within each component, and using a greedy algorithm to solve the MPP, are likely to reduce the quality of the final solution. Nevertheless, our experiments show that this approach is good enough to demonstrate the merit of the proposed cost function. If a more advance optimization technique is used, the solution found by the algorithm presented above can be used as a good starting point for the search.

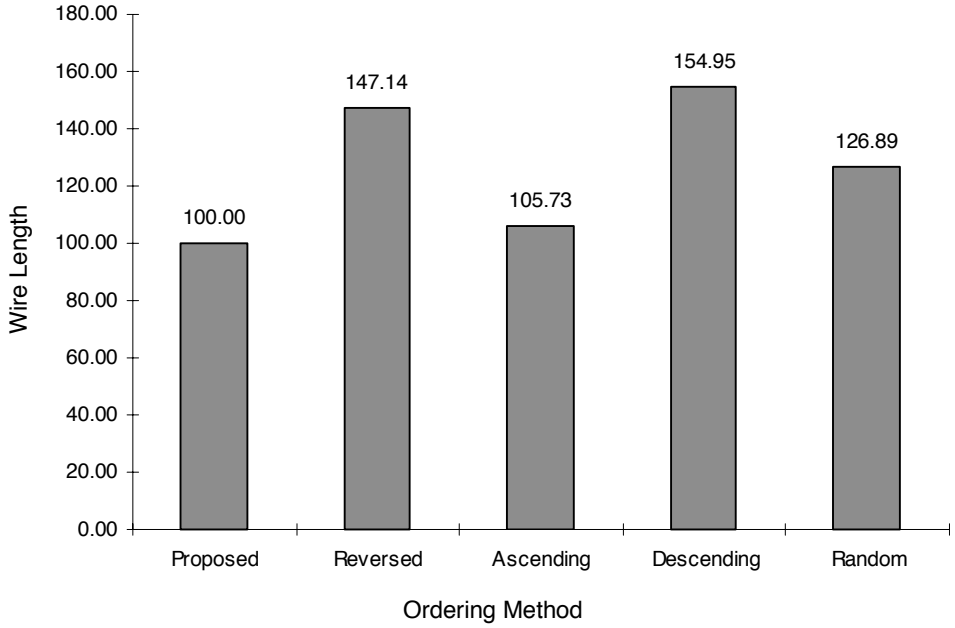
4.8 Experimental Results

To explore the merit of the proposed cost function and ordering algorithm, ten bins have been routed[1], each using five different ordering methods and the wiring length of the solutions have been compared. The ordering methods used are the proposed one, the proposed order reversed, the 2-Nets sorted in ascending and descending lengths, and a random order. All bins are from two-layer designs and their wiring lengths are the sums of the wiring length of each layer. The 2-Nets were embedded by a rubber-band shortest-path algorithm describe in Chapter 5. The relative wire lengths are shown in Table 11 and summarized in Graph 6. The wiring length of routing with the random order are averages with five different random orders per bin. As seen in the table, the proposed order has the lowest wiring length with the ascending 2-Net length order following it with 5.73% longer wiring. This supports our contention that the pair-wise cost function properly captures the dependency between the nets.

1. Euclidean metric, $\beta = 15$, no optimizations.

	2-Nets	Proposed	Proposed Reversed	Ascending Length	Descending Length	Random
APEX/01	48	100.00	225.01	108.41	239.71	139.67
APEX/02	140	100.00	171.00	104.17	119.21	164.06
APEX/03	136	100.00	182.20	101.12	164.20	144.33
DS15/01	36	100.00	130.47	103.66	245.87	134.12
DS15/02	31	100.00	138.03	102.03	155.53	112.28
DS15/03	27	100.00	118.38	111.36	119.92	118.82
DS15/04	24	100.00	122.29	103.67	121.68	113.09
GDX/02	29	100.00	157.49	118.84	164.74	133.84
GDX/02	25	100.00	104.02	102.05	99.54	101.29
GDX/03	26	100.00	122.48	102.00	119.12	107.46
Average		100.00	147.14	105.73	154.95	126.89

TABLE 11 - Routing with various orders of the 2-Net. The tables shows the relative wiring length when routing each of the ten bins using five methods of ordering the 2-Nets: the proposed one, the proposed one reversed, ascending and descending order of 2-Net lengths and random order. The values has been normalized to 100 in the case of the proposed order. The values of the random order are averages of five different orders for each bin, using different seeds for the random number generator. The values of the last rows are shown graphically



GRAPH 6 - Wiring length vs. ordering method. This graph shows the average wiring length for the 5 nets ordering methods in Table 11. The values are normalized to 100 for the proposed order. The proposed order results in the shortest wiring, and is shorter by more than 5% than the 'shorter nets first' order (Ascending column).

4.9 Conclusion and Future Work

I have considered the problem of ordering 2-Nets to guarantee planar embedding and to minimize wiring length. This problem is important when routing the 2-Nets one-at-a-time as done by many routers. The wire length minimization is done by formulating the problem as an optimization problem with cost function to be minimized and then solving it. The cost function estimates the length of the wiring detour when the 2-Nets are routed in a given order and is based on conflicts between pairs of components. We proposed a heuristic to solve the minimization problem by first transforming it to a problem of ordering components and then transforming that problem into an MPP which is then solved by a greedy algorithm. The presented Planarity Enforcement operator guarantees planar embedding by classifying the 2-Nets into two categories of 'open' and 'closed' 2-Nets and deferring the routing of the closed 2-Nets until after the open ones. We have shown that

this reordering preserves the cost of the order and therefore is not expected to increase the wiring length. Finally, we presented empirical results that show the proposed ordering method results in shorter wire length than several other methods including shortest net first.

The cost function of the 2NOP considers only dependencies between pairs of components and ignore dependencies between larger sets of components. Considering larger sets of components might improve the accuracy of the cost function and may result in lower wiring length. Other useful extensions to the cost function would be to consider congestion, wiring density, electrical requirements such as cross talk, and upper and lower bounds on nets lengths. Improvements can also be made to the algorithm used to solve the 2NOP. These include considering interleaving orders and employing a more sophisticated optimization technique which is less likely to be trapped in a local minima.

5 TOPOLOGICAL PATH SEARCH

A common approach to route a set of nets is routing them one at a time. After the nets have been decomposed into a set of two terminal nets (2-Nets) and the order of routing the 2-Nets has been determined, the router iteratively searches for an appropriate planar path for the next 2-Net and embeds it in the sketch on that path. The criteria for a desired path can consider for example, wire length, spacing requirements, and electrical properties of the interconnect. The search for a desired path is done in the specific representation of the interconnect used by the router.

In the rest of this chapter we discuss the problem of finding a least-cost planar topological path. First we formulate the general problem of finding a least-cost planar topological path. Then we present an exact formulation of the Rubber-Band Sketch (RBS) based on geometric considerations and real analysis. Then we present the concept of *regions* in the RBS and show the relationship between planar paths in the RBS and sequences of region. Following it, we present an algorithm that searches for a planar path in the domain of sequences of regions. The algorithm is optimal and finds a shortest planar path in $O((T^2 + S)\log(T + S))$ time where T and S are the number of terminals and wire segment respectively in the RBS. The algorithm can be modified to use a smaller search graph such that it is guaranteed to find a planar path in $O((T + S)\log(T + S))$ time and the path is likely to be short.

5.1 Least-Cost Topological Path Problem (LCTP)

The general problem of finding a least-cost path in a topological sketch can be formulated as follows:

Definition 14 (least-cost topological path problem LCTP) given a topological sketch, a cost function, and two terminals in the sketch, find a least-cost planar path in the sketch between the two terminals. It is assumed that the cost function has the property that if a planar path exists then there exists a planar path of minimal cost.

Solving the LCTP depends on the cost function and the sketch representation used. The focus in this chapter is on finding a shortest path in a RBS, we will present an algorithm that solves it optimally.

The LCTP could be generalized such that the path is to be found between two disjoint *sets* of terminals (i.e. connecting between two terminals of the two sets respectively). This may be useful when connecting existing sub-nets. For clarity, the discussion in this chapter focuses on the simpler version of the problem but we will show how the proposed algorithm can handle the extended version as well.

5.2 Rubber-Band Sketch Formulation

Before we present the planar shortest path algorithm we formally define the idea of rubber-band sketch. This formulation is used later to analyze the algorithm. Intuitively a Rubber-Band Sketch (RBS) represents a minimal-length member of a topology class such that on one hand zero spacing is allowed, and on the other hand the topology is preserved. However, when zero spacing is used, the result may be non-planar, and in this case, the RBS is not a member of the topology class, nor is it a valid geometric sketch. To overcome this ambiguity, we formulate the concept of RBS as follows.

Definition 15 (Rubber-Band Sketch - RBS) For a given topology class and $\varepsilon > 0$, we defined the ε*-sketch* of ε spacing of a topology class as a minimal-length sketch of that class whose spacing is at lest ε. The RBS of the class is defined as an ε-sketch of unspecified, infinitesimally small, $\varepsilon > 0$.

If ε is too large, the class may have no member of spacing $\geq \varepsilon$. However, every class, for small enough $\varepsilon > 0$, has an ε-sketch of ε spacing, and therefore, every class has an RBS. In general, we say that an RBS of some topology class has a property P if there is an $\eta > 0$ such that for every ε, $\eta > \varepsilon > 0$, the ε-sketch of the class with ε spacing has property P.

The paths in an ε-sketch are composed of arcs (called *attachment arcs*) and straight segments (Figure 53). The arcs have radius of integral number of ε's and are centered at terminal locations.

The decomposition of the branches into arcs and segments is identical in all ε-sketch of small enough $\varepsilon > 0$.

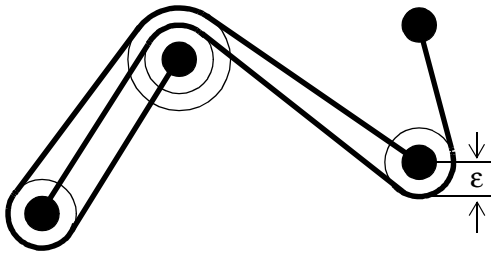


FIGURE 53 - ε-sketch. An ε-sketch is a minimal-length sketch having the spacing of at least ε for some $\varepsilon > 0$. The paths in an ε-sketch are composed of arcs and segments. The arcs are centered around terminals and have a radius of integral number of ε's.

For a given ε-sketch of ε spacing, we define the ε-*neighborhoods of a terminal* as a circle of radius c_ε, centered at the terminal location. c_ε is defined as ε plus the maximal radius of an attachment arc among all the terminals in the ε-sketch (Figure 54). If the ε-sketch does not have attachment arcs, c_ε is defined as ε. Note that by definition, the ε-neighborhoods of all the terminals in an ε-sketch have the same radius. The ε-neighborhood of each terminal strictly bounds the terminal and all of its attachment arcs. $c_\varepsilon = k\varepsilon$ for some integer $k > 0$, and it goes to 0 when ε goes to zero. Therefore, for small enough $\varepsilon > 0$, the ε-neighborhood of different terminals do not intersect with each other.

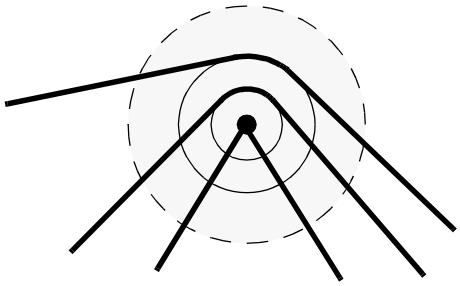


FIGURE 54 - Terminal's ε-neighborhood. The ε-neighborhood (shaded) of a terminal is a circle that strictly contains the terminal and its attachment arcs. The radius of the ε-neighborhoods of all the terminals in a sketch is identical and is determined by the maximal radius of an attachment arc in the sketch. The radius is proportional to ε (for small enough $\varepsilon > 0$) and goes to zero when ε goes to zero.

In a similar way, we define the ε*-neighborhood* of a cut between two terminals. Let t_1, t_2 be two terminals, $t_1 \neq t_2$. The ε*-neighborhood* of the cut (t_1, t_2) is defined as the closed domain formed by the ε-neighborhoods of t_1, t_2 and the two tangent lines connecting them (Figure 55).

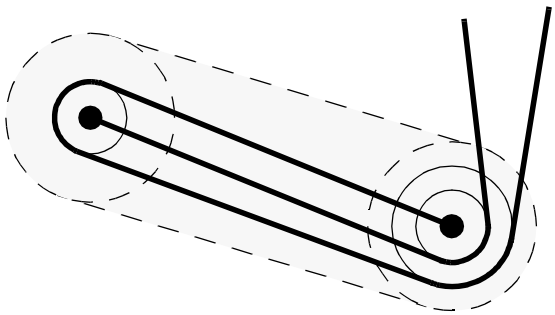


FIGURE 55 - Cut's ε-neighborhood. The ε-neighborhood of a cut is the closed domain (shaded) formed by the two ε-neighborhoods of the cut's terminals and the two tangent segments connecting them. The ε-neighborhood of the cut strictly contains all the segments between the two terminals.

A branch in an RBS is said to be *incident* to a terminal if the terminal is one of its end-points. A branch can also be *attached* to a terminal. We defines three cases of attachment between a branch b and terminal t (Figure 56). *Case 1*: b contains an arc centered at t and the limit of the angle of the arc when ε goes to zero is > 0. *Case 2:* similar to case 1 but the limit of the arc angle is 0.

Case 3: b has a segment s such that the end-points of s are not in the ε-neighborhood of t, and s intersects with the ε-neighborhood of t. Attachments of case 1 are called *explicitly attached* and attachments of cases 2 and 3 are called *implicitly attached*. Implicit attachments occur only in the presence of co-linear terminals. In case of an implicit attachments of case 3, we consider the segment s of the branch to be composed of two segments, one of each side of the attachment, connected by a degenerate attachment arc of 0 angle, centered at t. Note that a branch can be implicitly or explicitly attached to a terminal multiple times.

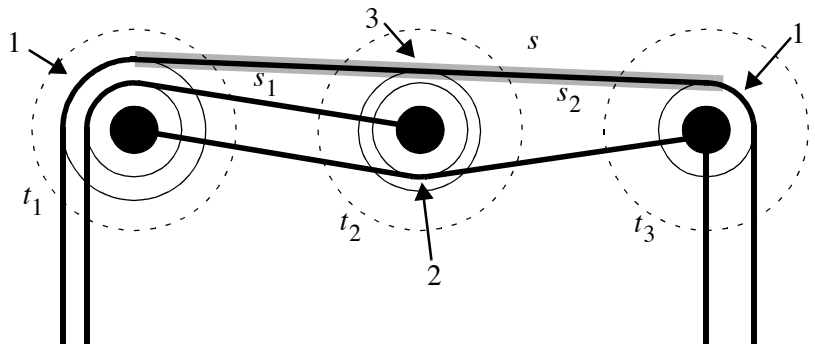


FIGURE 56 - Attachments in RBS. The figure shows a portion of an ε-sketch with three co-linear terminals t_1, t_2, t_3 whose ε-neighborhoods are indicated by the broken circles. The branch of segment s (shaded) is said to be *explicitly attached* (attachment case 1 in the text) to terminals t_1 and t_3. It is also said to be *implicitly attached* (case 3) to terminal t_2 since it intersects with its ε-neighborhood (for convenience, segment s is considered to be composed of two segments s_1, s_2, connected by a 0 angle arc centered at t_2). The branch connecting t_1 and t_2 is said to be *implicitly attached* (case 2) to terminal t_2 because it has an attachment arc at t_2 but the limit of the arc angle goes to zero when ε goes to zero. Implicit attachment (both cases 2 and 3) occur only in the presence of co-linear terminals.

The intersection of a net segment with the ε-neighborhood of a terminal that it is incident to defines an *incident local net* (Figure 57-a). In a similar way, every attachment of a branch to a terminal defines an *attached local net* which includes the arc and the portions of the two segments that intersect with the ε-neighborhood of the terminal (Figure 57-b).

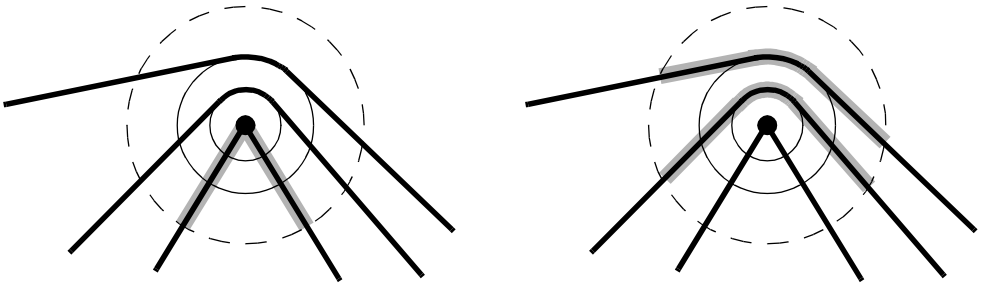


FIGURE 57 - Local nets. (a) shows the two incident local nets (shaded) of a terminal and (b) shows the two attached local nets of the same terminal.

Some properties of the branches in an RBS such as lengths, and the angle between a pair of segments connected by an arc, are defined as their respective limits when ε goes to zero. Since the limit the length of the arcs is zero, the length of a branch is the sum of the limits of the lengths of its segments. The limit of a segment length when ε goes to zero is the distance between the two terminals it is attached or incident to. In a similar way, the angle between two segments of a branch connected by an attached arc (Figure 58) is the angle between the two rays originated at the center terminal and intersecting with the terminals on the other ends of the two segments respectively.

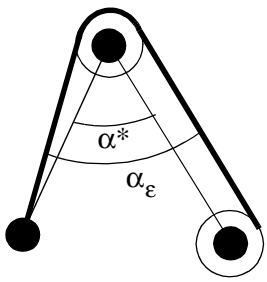


FIGURE 58 - Angle between branch segments. This portion of an RBS shows a branch attached to a terminal. The angle α_ε is the angle between the two segments in an ε-sketch of ε spacing. The angle α^* is the limit of α_ε when ε goes to zero

The representation of an RBS in computer memory can be implemented in various ways. SURF uses the following representation [9] (which is not a contribution of this research). The set of

terminals inside the routing area is triangulated using Constrained Delaunay Triangulation (CDT) [3] [41]. An edge between two terminals is constrained if there is a net segment connecting the ε-neighborhood of the two terminals (in this case we say the segment is *along* the edge). Every triangulation edge has a possibly empty list of the net segments along it (the orientation of the order is arbitrary). Each terminal is augmented with information about its local nets and the net segments they connect. The incident local nets are stored in a cyclic ordered list and the list of attached local nets are kept in a linear list in an inside-out order, starting from the inner-most arc. If a terminal has only implicit attachments, it may have up to two separate lists of attached local nets, representing the local nets on opposite sides of the terminal (Figure 59). If a terminal has an explicit attachment then it has exactly one ordered list of local attached nets since it cannot have attached local nets on opposite sides.

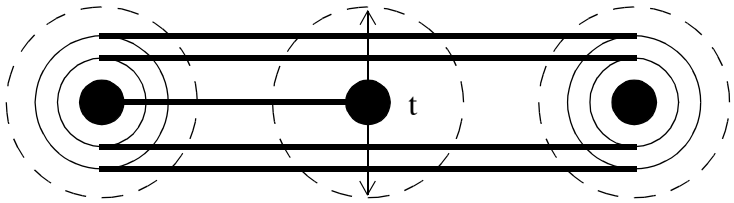


FIGURE 59 - Implicit attached local nets. A Terminal (such as *t* in this example) that has only implicitly attached nets can have up to two lists of attached local nets, one on each side of the terminal (the two arrows). If the terminal has an explicit attachment, it has exactly one list of attached nets.

5.3 RBS Regions

The proposed algorithm for finding a shortest planar path in an RBS is based on the concept of *regions* (defined below). As we will show later, the problem of finding a shortest path in an RBS can be reduced to a search in the domain of sequences of regions.

The attached and incident nets of a terminal separate the ε-neighborhood of the terminal into *regions*, each is a maximal set of connected points which does not intersect with nets or the

terminal (Figure 60). A point is said to be on the *interface* of a region if it does not intersect with any net, and it is on the intersection of the boundary of the region and the boundary of ε-neighborhood of the terminal. The interface of a region is non-empty and it defines one or more disjoint continuous sections on the boundary of the terminal's ε-neighborhood. These open-ended[1] sections are called the *ports* of the region (Figure 60). The interfaces of the regions of a terminal partition the boundary of the ε-neighborhood of the terminal such that every point on the boundary either intersects with a net segment or is on an interface of exactly one region. If a region is adjacent to its terminal or the terminal is fully contained inside the region, the region is said to be an *incident* region, otherwise, it is said to be an *attached* region. In general, a terminal has $max(1, n)$ incident regions and m attached regions where n, m are the number of incident and attached local nets respectively, of the terminal. For clarity of the presentation, the definition of regions and regions visibility ignores the RBS boundary. later we will show how terminals on the RBS boundary can be handled in a similar way.

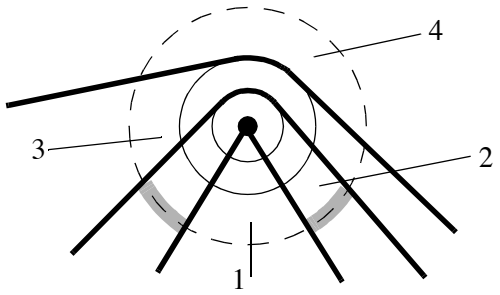


FIGURE 60 - Regions. The RBS terminal in the example has two incident nets and two attached nets. The two solid circles indicate the circles of the arcs of the attached nets and are of radius ε and 2ε respectively. The broken circle indicates the boundary of the ε-neighborhood of the terminal. The segments and the arcs of the nets partition the ε-neighborhood of the terminal into 4 non-connected areas called *regions* (marked 1 to 4). Regions 1,2 are incident to the terminal and therefore are called *incident regions*. Regions 3,4 are not incident to the terminal and therefore are called *attached regions*. The shaded arcs indicate the *ports* of the *interface* of region 2. The interfaces and ports of the other regions are defined in a similar way.

1. With the exception of an incident region of a terminal with no attached or incident local net. The port in this case is the entire boundary of the ε-neighborhood of the terminal.

The above definition is for a region in a specific ε-sketch. Since there is a one-to-one correspondence between regions of ε-sketches of the same topology class (for small enough $\varepsilon > 0$), we can refer to regions in an RBS in general. If a terminal in the RBS has more than one incident region, they can be uniquely identified by the incident local net bounding them in CW direction. If a terminal has more than one attached region, they can be uniquely identified by the attached local net bounding them in the direction toward the terminal.

Let t_1, t_2 be two terminals, $t_1 \neq t_2$, such that the cut between them does not intersect with a third terminal, and let r_1, r_2 be two regions of t_1, t_2 respectively. We say that regions r_1, r_2 are *visible* to each other if there exists a *visibility link* connecting them. A visibility link connecting r_1, r_2 is a line segment connecting between two points of the interfaces of r_1, r_2 respectively such that it is strictly contained within the ε-neighborhood of the cut (t_1, t_2) and does not intersect with terminals, nets, or inner points of the of the ε-neighborhoods of t_1, t_2 (Figure 61). Note that if two regions are visible, then by definition they must be of different terminals. Note also that a visibility link connecting two regions is not unique. Since the definitions of visibility between any two regions is insensitive to the choice of the ε-sketch (as long as $\varepsilon > 0$ is small enough), we can use these terms in general in the RBS.

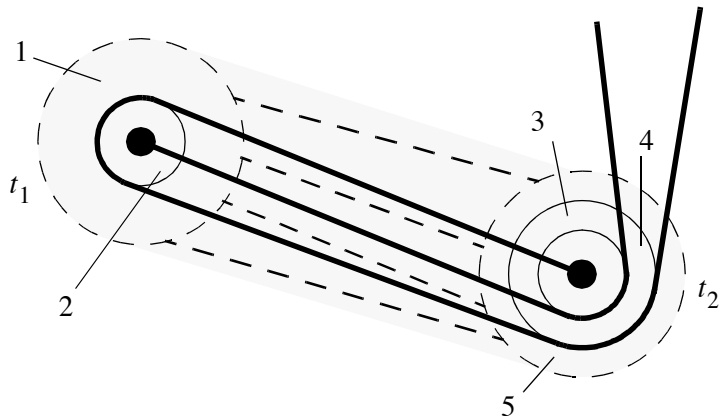


FIGURE 61 - Region Visibility. In this example, terminal t_1 has two regions (1,2) and t_2 has three regions (3,4,5). The broken circles show the ε-neighborhoods of the terminals and the shade area indicates the ε-neighborhood of the cut (t_1, t_2). In this example the pairs of visible regions are (1,3), (1,5), (2,3), and (2,4), and a visibility link for each of the pairs is shown (broken lines).

Lemma 7 Let r_1, r_2 be two regions in an RBS, visible to each other, and let t_1, t_2 be their terminals respectively. There exists $\eta > 0$ such that for every ε, $\eta > \varepsilon > 0$, the ε-neighborhood of the cut (t_1, t_2) does not intersect with the ε-*neighborhood* of any terminal other than t_1, t_2.

Proof By definition of visibility between r_1, r_2, $r_1 \neq r_2$ and $t_1 \neq t_2$, and the fact that the cut (t_1, t_2) does not intersect with any terminal other than t_1, t_2. We will show that for every terminal t other than t_1, t_2, there is an $\eta_t > 0$ that satisfies the requirements in regard to the ε-neighborhood of t. Since the number of terminals in the sketch is finite, $\eta = min(\eta_t)$ will satisfy the requirements of the lemma[1]. Let d be the distance between t and the line segment connecting t_1, t_1 (Figure 62). Since the segment (t_1, t_2) is close ended, d exists, and because t does not intersect with the segment, $d > 0$ (note that d is independent of ε). Let c_ε be the radius of the ε-neighborhood of the terminals in an ε-sketch of spacing ε. Since c_ε goes to 0 when ε goes to 0, there exists $\eta > 0$ such that for every ε, $\eta > \varepsilon > 0$, $c_\varepsilon < \frac{d}{3}$ and therefore, for every ε, $\eta > \varepsilon > 0$, the ε-neighborhood of the cut (t_1, t_2) does not intersect with the ε-neighborhood of t. Q.E.D.

1. If t_1, t_2 are the only terminals in the sketch, the minimum is undefined but in this case, any $\eta > 0$ will do.

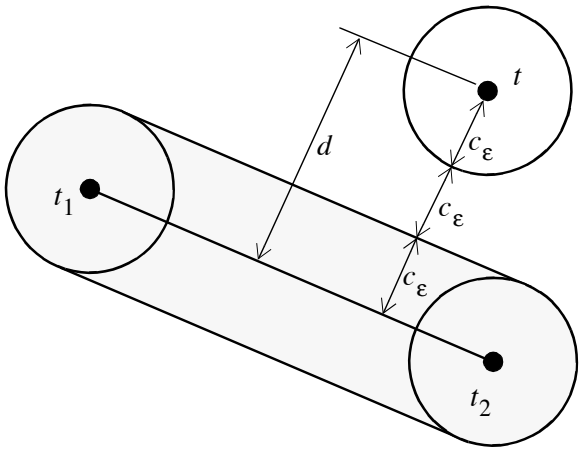


FIGURE 62 - Neighborhoods intersection. The circles indicate the ε-neighborhoods of terminals t, t_1, t_2. Terminal t is known not to intersect with the segment (t_1, t_2). The shade area is the ε-neighborhood of the cut (t_1, t_2). The distance between t and the close ended segment (t_1, t_2) is $d > 0$, which is independent of ε. If c_ε, the radius of the ε-neighborhood of the terminals is equal to $d/3$ or smaller, the ε-neighborhood of the cut (t_1, t_2) and the ε-neighborhood of t do not intersect. Note that this holds even if the point of the segment (t_1, t_2) that is closest to t is t_1 or t_2.

Lemma 8 Let l_1, l_2, be two visibility links in an RBS. If there is a terminal t and two regions $r_1 \neq r_2$ of t such that one end of l_1 is connected to r_1 and one end of l_2 is connected to r_2, then l_1, l_2 do not intersect.

Proof (Figure 63) Let r_3, r_4 be the regions at the other ends of l_1, l_2 respectively, let t_3, t_4 be the terminals of r_3, r_4 respectively, and let a, b be the end points of l_1, l_2, respectively at their t side. By definition of regions visibility, $t \neq t_3$ and $t \neq t_4$ but it is possible that $t_3 = t_4$. We will prove for the case that $t_3 \neq t_4$. The proof for the case when $t_3 = t_4$ is similar (whether $r_3 = r_4$ or not). Since $r_1 \neq r_2$ there is a net segment w coming out of the ε-neighborhood of t between[1] a, b (otherwise, r_1, r_2 would be the same region). Let p be the end-point of w at its end outside the ε-neighborhood of t. Point p, like any end-point of a net segment must be strictly inside an ε-

1. Since the ε-neighborhood of t is a circle, the term 'between' is ambiguous. In this context we refer to the points on the boundary of the ε-neighborhood of t that are strictly inside the triangle formed by a, b, and the intersection point of the two links.

neighborhood of some terminal. However, by Lemma 7, the union of the ε-neighborhoods of cuts (t, t_3), (t, t_4) do not intersect with the ε-neighborhood of any terminal other than t, t_3, t_4. This implies that the closed domain A (Figure 63), formed by the two links and the boundary of the ε-neighborhood of t, does not intersect with an inner point of the ε-neighborhood of any terminal of the RBS, and therefore point p must be outside of A. If p is outside A then w must intersect with at least one of the two links, and this contradicts the definition of visibility links. Q.E.D.

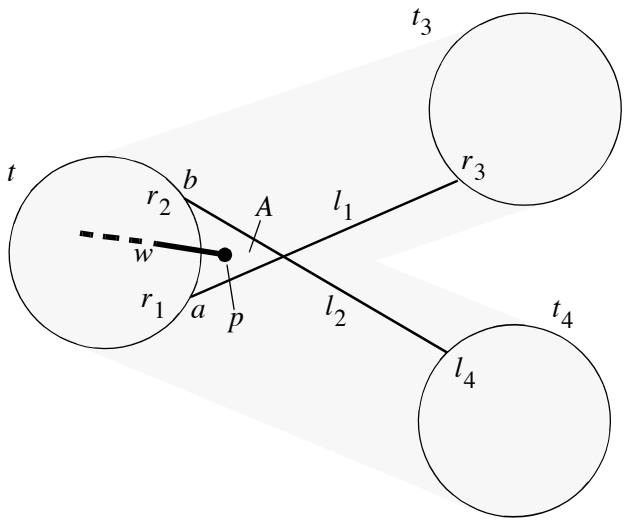


FIGURE 63 - Link intersections. This figure shows a hypothetical case where two visibility links l_1, l_2 that are connected to two regions $r_1 \neq r_2$ respectively of a terminal t, intersect with each other. In this case, the terminals t_3, t_4 at the other side of the links are assumed to be two distinct terminals, $t_3 \neq t_4$. The points a, b are the end-points of l_1, l_2 respectively. The circles indicate the ε-neighborhood of the terminals, and the shaded area is the union of the ε-neighborhood of the cuts (t, t_3) and (t, t_3). Since $r_1 \neq r_2$ there must be a net segment w separating them, and its end-point p must be in an inner point of an ε-neighborhood of some terminal. However, since the shaded area does not intersect with ε-neighborhood of any terminal other than t, t_3, t_4 (based on Lemma 7), w must intersect with at least one of the links. This contradicts the definition of visibility links, and therefore, this case is impossible.

5.4 Shortest Planar Path in RBS

Let S_1 be an RBS and let S_2 be the RBS S_1 after the insertion of some branch b connecting a pair of terminals t_1, t_2. The path of b splits some of the regions of S_1 into multiple regions of S_2 while the rest of the regions of S_1 are maintained with no change (Figure 64). Let r_1, r_2 be

regions in S_1, S_2 respectively. We say that r_1 *contains* r_2 if r_2 is exactly r_1 or if r_2 is a result of a split of r_1. Every local net of b in S_2 (either incident or attached) is adjacent to two (possibly identical) regions, one on each of its side, and these two regions are always contained in the same region of S_1. Therefore, the path of b in S_2 defines a sequence of regions of S_1, one region for each of its local nets, in the direction from t_1 to t_2. Every pair of consecutive regions in the sequence are visible to each other since they can be connected by a planar line segment similar to the net-segment of b connecting the two local nets of b that define the two regions.

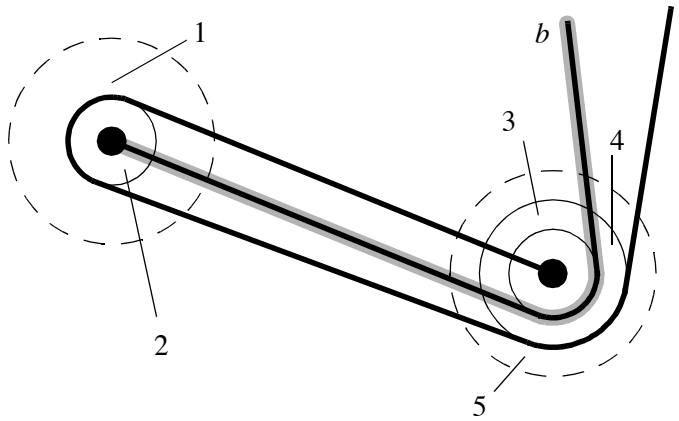


FIGURE 64 - Region split by a branch. This example shows a portion of an RBS S_2 which is the RBS S_1 with a new branch b inserted (shaded). The two terminals shown have 2 and 3 regions (of S_2) respectively, marked 1 to 5. Regions 3, 4 are the result of a split of a region of S_1 by the insertion of b, while the regions 1, 2, 5 existed in S_1 as are. Note that an insertion of a new branch can split a region of an RBS into more than two regions of the new RBS. This can happen if the branch has multiple attachment arcs inside the same region.

Let $r_1..r_n$, $n \geq 2$ be a sequence of regions in a RBS S (with possible repetition of the same regions). We say that $r_1..r_n$ is a *sequence of regions* between regions r_1 and r_n. If $r_1..r_n$ is between two incident regions of a pair of terminals t_1, t_2 respectively, we say that the $r_1..r_n$ is *between terminals* t_1, t_2. A consecutive pair of regions in the sequence, (r_j, r_{j+1}), $1 \leq i < n$, is said to define a *segment* s_i of the sequence. If the two regions of a segment are visible to each other than the segment is said to be a *visible segment*. If all the segments in a sequence are visible, the sequence is said to be a *visible sequence*. The *length* of a segment of a region sequence is

defined as the distance between the terminals of its two regions. The *length* of the entire sequence is defined as the sum of the lengths of its segments[1].

The sequence of regions defined by a branch path as describe earlier is a visible sequence between the branch's end-terminals. Further more, the length of the branch and the length of the sequence are equal. Later we use this similarity to reduce the problem of finding a shortest planar path to the problem of finding a shortest visible sequence between two terminals. Before we do that, we will prove several properties of a shortest visible sequence.

Lemma 9 A shortest visible sequence cannot include the same region more than once.

Proof By definition of region visibility, the two regions of a visible segment of the sequence must be of two different terminals. Therefore the length of a visible segment > 0 and a shortest visible sequence cannot contain the same region more than once otherwise it could be made shorter. Q.E.D..

Lemma 10 Let $r_1..r_n$, $n \geq 2$, be a shortest visible sequence, and let $s_1..s_{n-1}$ be its segments. There exist a set $l_1..l_{n-1}$ of visibility links for $s_1..s_{n-1}$ respectively, such that any pair of consecutive links l_i, l_{i+1} do not intersect with each other.

Proof $r_1..r_n$ is a visible sequence of regions and by definition of region visibility, there is a set $l_1..l_{n-1}$ of visibility links for $s_1..s_{n-1}$ respectively. If any pair of consecutive links do not intersect, then $l_1..l_{n-1}$ satisfies the requirements. Otherwise, we fix $l_1..l_{n-1}$ as follows. Let l_j, l_{j+1}, be two consecutive links intersecting each other. The link l_j connects regions r_j, r_{j+1} and the link l_{j+1} connects r_{j+1}, r_{j+2}. Let t_i, $1 \leq i \leq n$ be the terminal of region r_i respectively. Regions r_j, r_{j+1} are visible to each other and by the definition of region visibility, $t_j \neq t_{j+1}$. For similar considerations $t_{j+1} \neq t_{j+2}$. By Lemma 8 $t_j \neq t_{j+2}$ (otherwise the region $r_j \neq r_{j+2}$ are of the same terminal and this would contradict Lemma 8). Let a, b be the end-points of l_j, l_{j+1}, respectively at their end near terminal t_{j+1} (Figure 65). Points a, b and any point between[2]

1. Note that 'length' of a sequence does not refer to the number of regions in the sequence. We use the term 'size' for that purpose.

them, on the boundary of the ε-neighborhood of t_{j+1}, must be on the same port of region r_{j+1} otherwise there would be a net segment between a, b that must intersect with the visibility links l_j or l_{j+1} (similar to w in the proof of Lemma 8). Let P be that port of r_{j+1}. The intersection of port P with the inner points of the ε-neighborhood of the cuts (t_j, t_{j+1}), (t_{j+1}, t_{j+1}) defines a non empty, open-ended arc that intersects with both a, b. Let $P^* \subseteq P$ be that arc. We choose (as defined below) two points $a^* \neq b^*$ on P^* and define a new line segment l^*_j (l^*_{j+1}) as the segment of lj (l_{j+1}) with the end-point a (b) replaced by a^* (b^*) (the broken lines in Figure 65). The points a^*, b^* are two arbitrary points on P^* oriented such that l^*_j, l^*_{j+1} do not intersect[1]. The line segments l^*_j, l^*_{j+1} do net intersect with net segments (otherwise there would be a net segment intersecting l_j, l_{j+1}, or P^*) or terminals (based on Lemma 7), and they connect the same pairs of regions as l_j, l_{j+1} respectively, and therefore they can replace l_j, l_{j+1} in the set of visibility links. This operation does not create a new intersection between consecutive links in the set since the new link l^*_j (l^*_{j+1}) intersects with its previous (next) link l_{j-1} (l_{j+2}) if an only if the old link l_j (l_{j+1}) intersects with l_{j-1} (l_{j+2}) (based on Lemma 9 and the observation that $r_{j+1} \neq r_{j-1}$ and $r_{j+1} \neq r_{j+3}$). Therefore, if we will repeat this operation for every pair of consecutive links intersecting each other, we will end up with a set of links that satisfy the requirements. Q.E.D.

---

2. Since the ε-neighborhood of t_{j+1} is circular, the term 'between' is ambiguous. We use here a definition of 'between' similar to the one used in the proof of Lemma 8.

1. Points a and b be may be the same point, otherwise we could simply define a^*, b^* as b, a respectively.

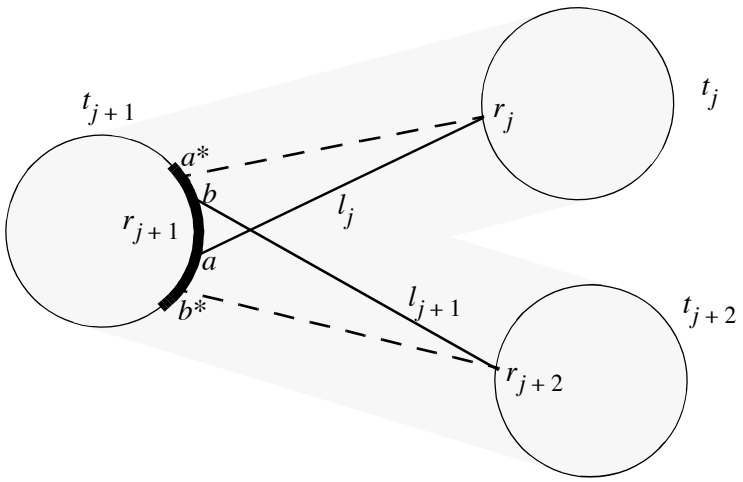


FIGURE 65 - Intersection of consecutive visibility links. This example shows how an intersection between two consecutive links of a shortest visible path can be eliminated. The three circles indicate the ε-neighborhoods of terminals t_j, t_{j+1}, t_{j+2}. The shaded area is the union of the ε-neighborhoods of the cuts (t_j, t_{j+1}), (t_{j+1}, t_{j+2}). The ends of the two intersecting links l_j, l_{j+1} are connected to a port of region r_{j+1} (the dark arc) at points a, b respectively (possibly $a = b$). By replacing the end points a, b of the two links with points a^* and b^* respectively, we get an alternative pair of links (the broken lines) that do not intersect. This operation does not create a new intersection between a pair of consecutive links.

The previous lemma eliminates intersections between consecutive links in $l_1..l_{n-1}$. Now we will extend this for intersections between *any* pair of links in $l_1..l_{n-1}$.

Lemma 11 Let $r_1..r_n$, $n \geq 2$ be a shortest visible sequence, and let $s_1..s_{n-1}$ be its segments, There is a set $l_1..l_{n-1}$ of visibility links of $s_1..s_{n-1}$ respectively such that any pair of links do not intersect.

Proof By Lemma 10 there is a set of visibility links $l_1..l_{n-1}$ of $s_1..s_{n-1}$ such that consecutive links do not intersect. We will prove by contradiction that non consecutive links do not intersect as well. For the purpose of contradiction we assume that there are two non-consecutive links l_j, l_k that intersect. We assume without loss of generality that $j < k$. The link l_j is between regions r_j, r_{j+1}, and the link l_k is between regions r_k, r_{k+1}. Let t_1, $1 \leq i \leq n$, be the terminal of the respective region r_i. By definition of visibility between r_j, r_{j+1}, $t_j \neq t_{j+1}$ and in a similar way

$t_k \neq t_{k+1}$. The two links can share 0, 1 or 2 terminals but do not share any region (they are not consecutive and by Lemma 9 a region has at most one instance in $r_1..r_n$).

Case 1: the links l_j, l_{j+1} share no terminal (Figure 66). In this case, we will show that the regions $r_1..r_n$ can be replaced by a shorter visible sequence between r_j, r_{k+1} and this contradicts the assumption that $r_1..r_n$ is a shortest visible sequence between r_1, r_n. The new sub-sequence includes the regions r_j, r_{k+1}, and between them a (possibly empty) sub-sequence of regions representing attachments to terminals between t_j, t_{k+1}. These terminals are on the convex hull of the terminals in the triangle formed by t_j, t_{k+1} and the intersection point of the cuts (t_j, t_{j+1}), (t_k, t_{k+1}).

Case 2: the links l_j, l_{j+1} share one or two terminals. This case is contradicted by Lemma 8. Q.E.D.

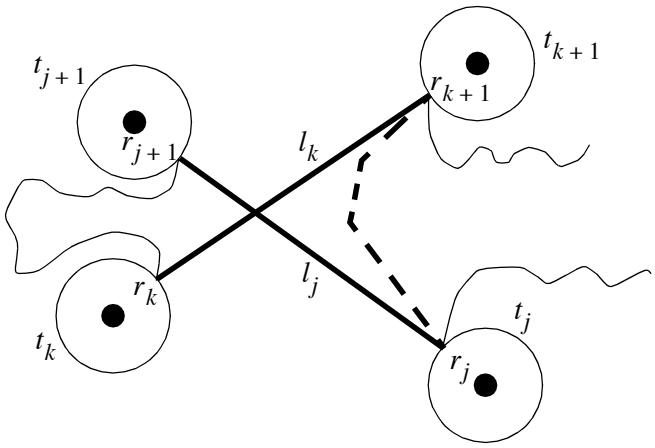


FIGURE 66 - Intersection between non-consecutive links. This figure shows a hypothetical example where l_j, l_k are two non-consecutive visibility links of a shortest visible sequence that intersect. The sequence of regions can be shorten by replacing the regions r_j, r_{k+1} with a shorter visible sub-sequence (the broken line), and therefore, this case is impossible.

Lemma 12 Let $r_1..r_n$ be a shortest visible sequence of regions between terminals t_1, t_2. The sequence defines a planar path connecting t_1, t_2. The path is composed of a set of visibility links, one for each segment of the sequence, and of connections of infinitesimally small lengths, one in each of the sequence regions.

Proof By Lemma 10 there is a set of visibility links $l_1..l_{n-1}$ of $s_1..s_{n-1}$ such that the links do not intersect with each other. By definition of region sequence between terminals, regions r_1 is an incident region of terminal t_1 and therefore the first end-point of link l_1 can be connected to terminal t_1 inside region r_1 (Figure 67a). The connection can be selected such that its inner points do not intersect with any net, terminal, or a boundary of an ε-neighborhood of a terminal. In a similar way, terminal t_2 can be connected inside region r_n to the end-point of link l_{n-1}. As for the interim regions r_i, $1 < i < n$, by Lemma 9 each region r_i can appear exactly once in the sequence and therefore there are exactly two consecutive visibility links l_{i-1}, l_i whose end-points are on the boundary of r_i and they can have a planar connection within that region (Figure 67 b). The connections can be made such that their length goes to zero when ε goes to zero, and this results in a planar path between t_1, t_2 that satisfy the requirements. Q.E.D.

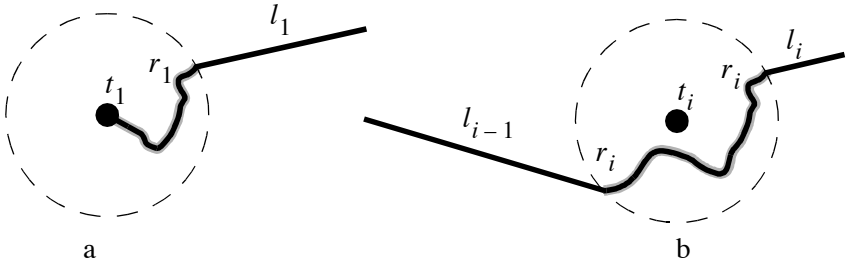


FIGURE 67 - Link interconnection within regions. This figure shows two cases of connecting visibility links within a region to form a planar path. The circles indicate the ε-neighborhoods of terminals and the straight line segments indicate portions the visibility links. In (a) the beginning of the first link is connected to the terminal t_1 such that the connection is contained within the incident region r_1. In (b) two consecutive links are connected within the region r_i. In both cases the connections can be made such that their length goes to 0 when ε goes to zero. The connections do not intersect nets or other connections (every region of the sequence has exactly one connection).

Theorem 5 Let t_1, t_2 be two terminals in RBS S_1, and let $P = r_1..r_n$ be a shortest visible sequence of regions between t_1, t_2. P defines a shortest[1] planar path between t_1, t_2.

1. Note that the length of a path in a RBS is the limit of its length in ε-sketch of ε spacing when ε goes to zero.

Proof By Lemma 12 the sequence P defines a planar path P connecting t_1, t_2 such that the lengths of P and P are identical, $|P| = |\overline{P}|$. For the purpose of contradiction we assume that there is a shorter planar path $\overline{P_1}$ connecting t_1, t_2 such that $|\overline{P_1}| < |\overline{P}|$. Let S_2 be the RBS of the topology class of S_1 with the path $\overline{P_1}$ inserted. S_2 contains a rubber-band branch b representing the path $\overline{P_1}$ and therefore its length is at most the length of $\overline{P_1}$, $|b| \le |\overline{P_1}|$. The branch b defines a visible sequence P_2 of regions of S_1 between t_1, t_2, and the lengths of the branch and the sequence are equal, $|P_2| = |b|$. This implies that $|P_2| = |b| \le |\overline{P_1}| < |\overline{P}| = |P|$ and therefore P_2 is shorter than P. Contradiction. Q.E.D.

Based on *Theorem 5*, finding a shortest planar path between a given pair of terminals t_1, t_2 in an RBS is relatively simple because it is sufficient to find a shortest visible sequence of regions between the two terminals. This can be reduced to the problem of finding a least-cost path in a graph $G = (V, E)$ with positive costs. The nodes in the graph correspond to regions of the RBS. An edge is defined between two nodes if their corresponding regions are visible to each other, and the cost of an edge is the distance between the terminals of its two regions. The starting and the termination nodes are those representing the incident regions of t_1, t_2 respectively. This approach can be generalized to find a shortest path between two sets of terminals in the RBS. This is done by defining the starting and the termination nodes of the graph as the nodes that correspond to incident regions of the two sets of terminals respectively. In both cases, based on *Theorem 5*, the algorithm is guaranteed to find an optimal solution.

The following is an analysis of the size of the graph G. Let T, B, and S be the number of terminals, branches, and net segments, respectively in the RBS. Every branch has at least one segment and therefore $S \ge B$. The sketch has $2B$ incident local nets and $S - B$ attached local nets. The sketch with all the branches removed has exactly T incident regions and no attached regions. An incident local net can contribute at most a single incident region and therefore the number of incident regions in the sketch $\le T + 2B$. In a similar way, an attached local net contributes exactly one attached region, and therefore the number of attached regions is $S - B$. The total number of regions $\le (T + 2B) + (S - B)$ and therefore the graph has $O(T + S)$ nodes. As for the graph edges, the sketch with all branches removed has $\le \frac{T(T-1)}{2}$ visible pairs of

regions[1]. Every incident or attached local net increases the count of visible pairs by at most one[2] and therefore the total number of edges $\leq \frac{T(T-1)}{2} + 2B + (S-B)$ and the graph has $O(T^2 + S)$ edges**.** By using an $O(E \log V)$ search algorithm, a shortest path can be found in $O((T^2 + S)\log(T + S))$ time.

All the properties of the shortest visible sequence of regions hold for the Euclidean, octilinear and rectilinear metrics and therefore the shortest path algorithm can be used for any of the these three metrics. If the RBS has terminals on the RBS boundary, only the portion of the ε-neighborhood of the terminal that is strictly inside the sketch boundary is considered to contain regions (Figure 68). All the properties of shortest visible sequences of regions shown above hold as well. In a similar way, the concept of regions can be extended to handle disjoint obstacles, each a (possibly concave) finite simple polygon. A pseudo terminal is added at every corner of the polygon whose external angle $\geq \pi$, and a pair of regions is considered for visibility only if the cut between their terminals does not intersect with an inner point of an obstacle. Note that similar to the case of the terminals on the sketch boundary, a net segment can be on an edge of a obstacle boundary but it cannot intersect with an inner point of an obstacle.

---

1. If no three terminals in the RBS are co-linear, this is the exact number of visible pairs of regions.

2. This is a conservative approach. Practically, the insertion of a branch reduces the number of visibility edges because they block the visibility between terminals on the two sides of its segments. This observation however does not affect the worst case analysis.

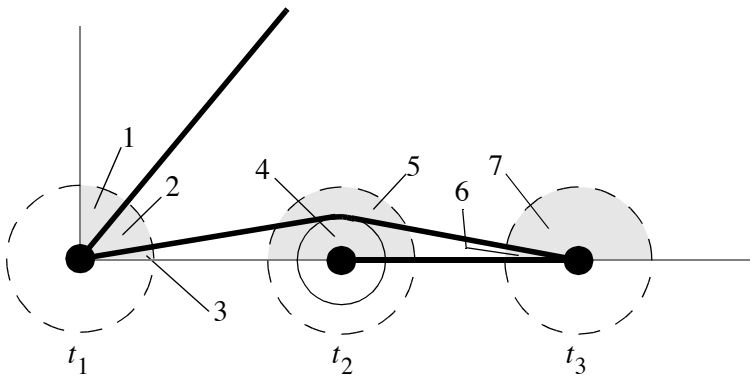


FIGURE 68 - Region on sketch boundary. This example shows a lower left corner of a rectangular RBS. Three terminals are shown, t_1 on the sketch corner, and t_2, t_3 on the sketch lower boundary. Only points which are strictly inside the sketch boundary can be members of a region. The shaded areas show the union of all the regions (1 to 7). Note that regions 1,3 are separated by the sketch boundary otherwise they would be the same region. It is valid for a net segment to be on the sketch boundary similar to the one connecting t_2, t_3, but net segments cannot intersect with points outside the sketch. A corner terminal like t_1 can have incident nets but cannot have any attachment since the internal angle of the corner is less than π.

5.5 Reducing the search graph size

The optimal shortest path algorithm described above searches in the graph $G = (V, E)$ that has $O(T + S)$ nodes and $O(T^2 + S)$ edges and therefore the search time is $O((T^2 + S)\log(T + S))$. To reduce the search time, SURF uses a smaller graph that has the same set of nodes and only $O(T + S)$ edges, and therefore the search time in the reduced graph is $O((T + S)\log(T + S))$. The algorithm is guaranteed to find a planar path if one exists but the path found may be not a shortest path. The algorithm uses the graph $G^* = (V^*, E^*)$ that has the same set of nodes as G, $V^* = V$, and only a subset of the edges of G, $E^* \subseteq E$. The cost of the edges E^* is the same as their cost in E. The terminals of the RBS are triangulated within the routing area and an edge $e \in E$ is in E^* if the terminals of its two regions are on the two ends of a triangulation edge (Figure 69). The triangulation is constrained to include an edge between any pair of terminals that have a net segment between them (either attached or incident). The set of constrained edges is planar (each can intersect with terminals or other edges only at its end-point) and therefore such a constrained

triangulation exists[1]. Since the triangulation has only $O(T)$ edges (a triangulation is a planar graph), G^* has only $O(T+S)$ edges. Later on, we will prove that searching in G^* is guaranteed to find a planar path in the RBS if one exists.

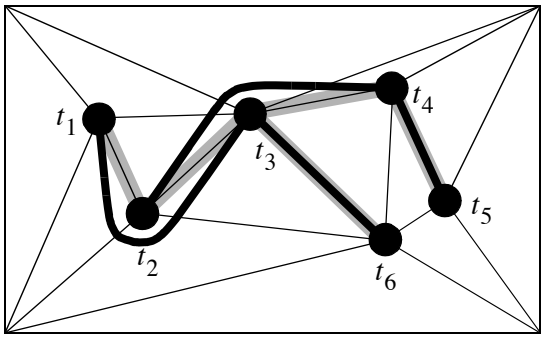


FIGURE 69 - RBS triangulation. This figure shows a triangulation, within the routing area of the terminals of an RBS. The shades indicate the constrained edges of the triangulation. These are edges between pairs of terminals having a net segment between them. Note that as opposed to G, the graph G^* does not contain an edge between the regions of t_3 and t_5 because there is no triangulation edge between them.

Lemma 13 Let $r_1..r_n$, $n \geq 2$, be a shortest visible sequence of region in an RBS with some triangulation. Let $l_1..l_{n-1}$ be a planar set of visibility links of the segments of $r_1..r_n$ (by Lemma 11 it exists). Let l_j, l_k, $j \neq k$, be two links of the sequence, and let $e = (t_1, t_2)$ be a triangulation edge that does not have a common end-terminal with l_j and l_k. Under these conditions, if both l_j, l_k cross e, then the RBS has a net segment crossing e between l_j, l_k.

Proof Let a, b be the intersection points of l_j, l_k respectively with e (Figure 70). Since the two links do not intersect, $a \neq b$. If no net crosses e between a, b, we can connect points a, b, and remove any cycles created in the path defined by $r_1..r_n$. This creates a new planar path connecting the same terminals of the original path. When ε goes to zero, the limit of the distance between the edge e and each of the end-points of l_j, l_k is > 0 because e does not share end-terminals with l_j, l_k, and like any triangulation edge, e does not intersect with a third co-linear terminal. Therefore,

1. Note that a triangulation is not necessarily unique.

the limit of the length of the new path, when ε goes to zero, is lower than the limit of the path defined by $r_1..r_n$, and this contradicts *Theorem 5*. Therefore e must be crossed by a net segment between a and b . Q.E.D.

Proof

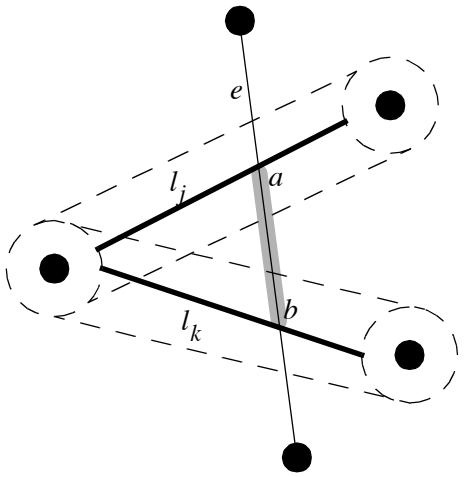


FIGURE 70 - Edge intersections. The two planar links l_j , l_k are of a shortest visible sequence, and they intersect with the triangulation edge e at points a , b respectively. If no net segment crosses e between points a , b the path could be made shorter by connecting a , b with a straight line segment (shaded) and removing any cycle(s) it creates in the path. This would contradict the fact that the path represented by a shortest visible sequence is of shortest length. The figure shows the case where the two links shares a single terminal. The cases where the links share 0 or 2 end-terminals are similar.

Definition 16 (Corridor[1]) Let t_1 , t_2 be two terminals in a RBS with some triangulation of its terminals, such that the open segment (t_1, t_2) does not intersect with any terminal, and the triangulation does not contain an edge for the cut (t_1, t_2) . In this case we say that the cut (t_1, t_2) defines a *corridor* between t_1 and t_2 . The corridor is a simple polygon that includes the triangles that intersect with the open ended segment (t_1, t_2) (Figure 71).

1. The definition here of a corridor is a special case of the same term used in [35]. Here it is defined by the straight line segment between the two terminals as opposed to a general piece-wise linear path in [35].

.

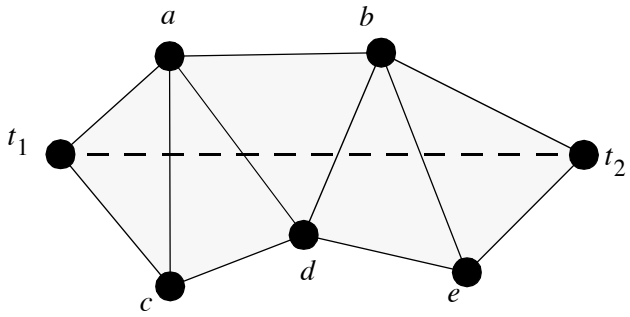


FIGURE 71 - Corridor. The corridor of the cut (t_1, t_2) includes the shaded area. The sequences of terminals on the two sides of the corridor are t_1 , a , b , t_2 and t_1 , c , d , e , t_2 respectively.

The corridor of the cut (t_1, t_2) defines two sequences of terminals along the corridor boundary, one sequence on each side. Each of the sequence starts with t_1 , ends with t_2 and has at least one terminal in between (Figure 71).

Definition 17 (Corridor internal point) A point is said to be *internal* to the corridor in a given ε -sketch if it is inside the corridor's boundary, and it does not intersect with the ε -neighborhood of any cut between consecutive terminals on the corridor boundary (Figure 72).

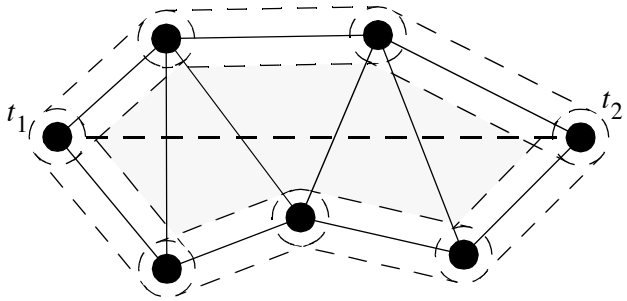


FIGURE 72 - Corridor internal point. The shade area indicates the internal area of the corridor between terminals t_1 , t_2 .

Lemma 14 Let $r_1..r_n$, $n \geq 2$, be a shortest visible sequence of regions in an RBS with some triangulation, and let $l_1..l_{n-1}$ be a planar set of visibility links of the segments of $r_1..r_n$ (it exists

by Lemma 11). Let l_j, $1 \le j < n$, be an arbitrary link of the sequence, and let t_j, t_{j+1} be the terminals of the two end regions of l_j. Under these conditions, if the triangulation does not have an edge between t_j, t_{j+1}, then l_j defines a corridor between t_j, t_{j+1}, and no net-segment or other link l_k, $1 \le k < n$, $k \ne j$, intersects with any internal point of the corridor.

Proof First we will show this for a visibility link and then for a net segment. We assume for the purpose of contradiction that there is a link l_k, $k \ne j$, that intersects with an internal point of the corridor. The link l_k can share 0, 1 or 2 terminals with the boundary of the corridor, and in each these cases, there must exist an edge e within the corridor that does not share any end-terminal with l_k, and is crossed by l_k (see example in Figure 73 of the case where l_k shares two terminals). This is based on the observations that l_k cannot be between two consecutive terminals of the corridor boundary (otherwise it will not intersect with an internal point), l_k cannot cross l_j (because the set of links is planar), l_k cannot be co-linear with an edge inside the corridor (otherwise it will intersect with l_j), and as any visibility link, for small enough $\varepsilon > 0$, it does not intersect with ε-neighborhood of any terminal other than its two ends (Lemma 7). Therefore, the crossing point must be an internal point of the corridor. By applying Lemma 13 to links l_j, l_k, and e, there must be a net-segment crossing e between l_j, l_k, and the crossing point is at an internal point of the corridor. However, no net segment can intersect with the inner point of the corridor (it must be on a triangulation edge and l_j crosses all the triangulation edges inside the corridor). This is a contradiction. As for a net segment, by similar considerations, if a net segment intersects with an inner point of the corridor then it must intersect with an edge inside the corridor that does not share an end-terminal with it, and this implies that two edges of the triangulation (the constrained edge of the net segment and the crossed edge) cross each other. Again, this is a contradiction. Q.E.D

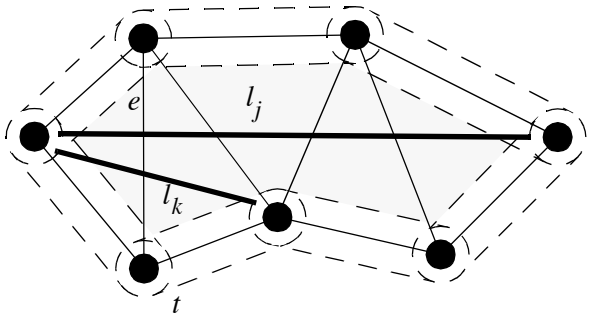


FIGURE 73 - Link/Edge intersection. l_k is a planar visibility link that intersects with an inner point of the corridor. l_k crosses the edge e inside the corridor, and e, l_k do not share an end-terminal. In this example, the link l_k shares two terminals with the corridor boundary, and since it intersects with an inner point of the corridor, there must be at least one terminal between them (marked t) on the corridor boundary.

Next we will show that every planar path defined by a shortest visible sequence of regions in G has a corresponding planar path of the same topology that is defined by a path along edges of G^*.

Lemma 15 Let $r_1..r_n$, $n \geq 2$, be a shortest visible sequence of regions between two terminals, and let P be a path defined by it (by Lemma 12). The RBS has a path P^*, made of visibility links of G^* that has the same topology as P (i.e. inserting P or P^* the sketch will result in the same topology class).

Proof Let $l_1..l_{n-1}$ be the sequence of visibility links of P. For every link l_j, $1 \leq j < n$, between regions r_j, r_{j+1}, that does not have an edge in G^*, we replace it with a sequence of visibility links of edges in G^* as follows. Since there is no triangulation edge between the end-terminals of l_j, the link l_j defines a corridor between its end-terminals. Let $t_1..t_k$, $k > 2$, be the sequence of terminal on an arbitrary side of the corridor. t_1, t_k are the terminals of the two regions respectively of l_j (Figure 73). By Lemma 14, no net segment or link other than l_j intersects with an inner point of the corridor. For every cut (t_i, t_{i+1}), $0 \leq i < k$, along the boundary of the corridor between t_1, t_k we define a visibility link l_i^* connecting between a region of t_i and a region of t_{i+1} as follows. The link l_i^* is selected such that it does not intersect with any net segment or any other link that is between terminals t_i, t_{i+1}, and no net segment of other link

between t_i, t_{i+1} is between l_i* and the internal area of the corridor (Figure 74). The new links and the link l_j form a cycle. The links of the cycle do not intersect with any net segment or other link (otherwise Lemma 14 would be contradicted), and for similar reasons[1] they can have planar connections within the regions. Furthermore, the inner part of the cycle do not intersect with any net, link, or terminal, and if we will remove l_j, we will end up with a planar path of the same topology as P. If we will repeat this step for all the links that are not on edges of G*, we will end up with a path P* that satisfies the requirements. Q.E.D.

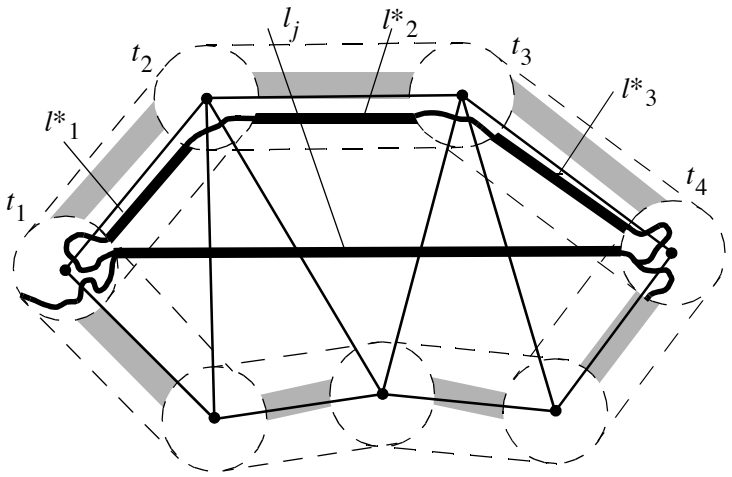


FIGURE 74 - Replacing a G link with G* links. This example shows how a visibility link of an edge in G can be replaced by a sequence of links of edges of G* while maintaining the planarity and topology of the path. The broken circles indicates the ε-neighborhoods of the terminals. The shade lines indicate possible links and net segments. Since no net segment or link intersects with the inner part of the corridor, the new path can be made planar.

In the other direction, we show that every shortest path in G* defines a planar path in the RBS (but not necessarily a shortest path).

Lemma 16 Any shortest path in G* between incident regions of two terminals defines a planar path that connects the two terminals.

---

1. We assume, without loss of generality, that the construction of the first and the last links l^*_1, l^*_{k-1} is such that they do not intersect with l_j. Such construction is always possible.

Proof This lemma is similar to Lemma 12 except that here the sequence of regions is shortest in G^* but not necessarily in G. Lemma 9, Lemma 10, Lemma 11 and Lemma 12 apply also to a shortest sequence of regions in G^* and can be proven in a similar way. The only difference is case 1 in the proof of Lemma 11 (Figure 66). In the case of a shortest path in G^*, this case is contradicted by the fact that triangulation edges cannot cross each other. Q.E.D.

Theorem 6 Searching in the G^* graph will result in a planar path if and only if one exists.

Proof If there is a planar path in the RBS, then by *Theorem 5*, G contains a path that defines a planar path in the RBS, and by Lemma 15, G^* contains a path as well that defines a planar path in the RBS. The other direction is proved by Lemma 16. Q.E.D.

Let S_1 be an RBS with some triangulation of its terminals, let $r_1..r_n$ be a shortest sequence of regions between two terminals in the graph G^* of S_1, and let P^* be a path defined by that sequence. The length of P^* is equal to the length of the sequence $r_1..r_n$. Let S_2 be the RBS of the topology class of S_1 with the path P^* inserted. P^* has a corresponding branch b^* in S_2 that connects the same pair of terminals. The paths of P^* and b^* have similar topology but b^* can be shorter than P^* (Figure 75). This implies that the length of the sequence $r_1..r_n$ is an upper-bound of the length of b^* (or the *actual* length of P^*).

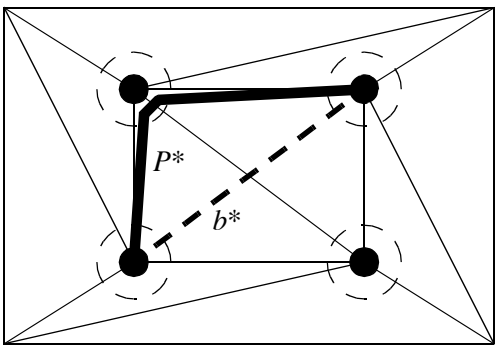


FIGURE 75 - A branch path of P^*. P^* is a path defined by a shortest visible path along triangulation edges. The RBS of the topology class with path P^* inserted has a branch b^* (broken line) representing the path P^*. The paths of P^* and b^* have the same topology in respect to the other nets and terminals but b^* is shorter than P^*.

When SURF inserts the path P^* into the RBS, it first computes the net segments and local nets of the new branch b^*. This process is called a *validation* of P^* and is performed by iteratively applying the *attachment* operator (described below) to every *invalid* attachment of P^*, until P^* contains no invalid attachments. A connection between two links in P^* is said to be an *invalid* attachment if the limit of the angle between the links, on the side that does not include the terminal, is $< \pi$ (Figure 76). Note that this definition is independent of ε, the actual choice of the links between the regions, and the paths of the connections inside the regions, as long as the sequence of regions along the path and the topology are maintained. Further more, the validity of an attachment can be determined given the locations of the three terminals, and the topology of the attachment.

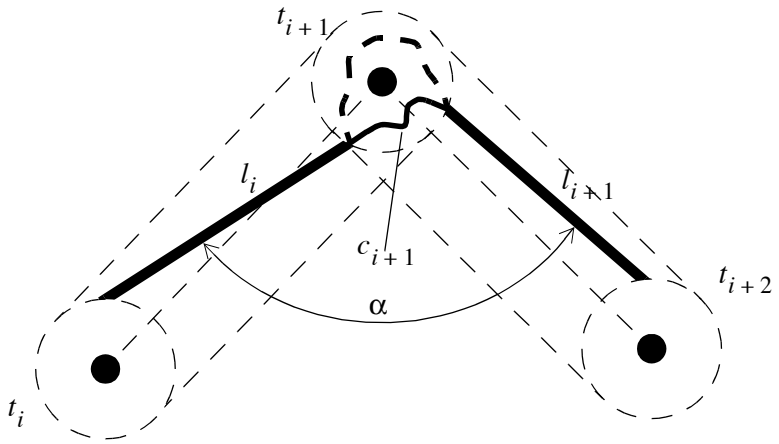


FIGURE 76 - Invalid attachment. The thin broken lines indicates the cuts (t_i, t_{i+1}), (t_{i+1}, t_{i+2}) and their ε-neighborhoods. The two links l_i, l_{i+1} are connected inside the ε-neighborhood of terminal t_{i+1} by the connection c_{i+1}. The connection c_{i+1} partitions the ε-neighborhood of t_{i+1} into two areas and one of them include the terminal t_{i+1}. α is the angle between l_i, l_{i+1} such the terminal t_{i+1} is on the other side of the connection c_{i+1}. The attachment in this example is said to be invalid because the limit of α when ε goes to 0 is $< \pi$. If c_{i+1} would be replaced with the alternative connection (broken line) the attachment would become valid (assuming it does not intersect with existing local nets).

The *attachment* operator when applied to an invalid attachment between two links l_i, l_{i+1}, replaces the two links in the path with a new sub-sequence of one or more links, and connections of them, such that the new sub-sequence forms a planar path together with the head and tail of the

original path (Figure 77). The new path has the same topology as the original path and is always shorter. If there is more than one new link, all the connections between consecutive new links have valid attachments. Let (t_i, t_{i+1}), (t_{i+1}, t_{i+2}) be the two cuts of the end-terminals of l_i, l_{i+1} respectively. The new links are found by computing the convex hull of the terminals, other than t_{i+1}, intersecting with the triangle t_i, t_{i+1}, t_{i+2} (Figure 77). The attachment operator maintains the topology of the path and the validation process is guaranteed to terminate since the reduction in the length of P^* by each application of the attachment operator has a positive lower bound independent of ε [1].

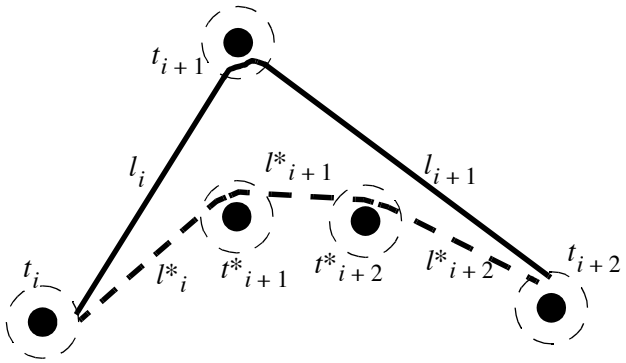


FIGURE 77 - Attachment operator. In this example, the attachment operator is applied to an invalid attachment between links l_i and l_{i+1}. The operator replaces the two links in the path with the sub sequence of new links l^*_i, l^*_{i+1}, l^*_{i+2}. The connections between the new links represent valid attachment to the terminals t^*_{i+1}, t^*_{i+2}. The new links can be found by computing the convex haul of the terminals, other than t_{i+1}, intersecting with the triangle t_i, t_{i+1}, t_{i+2}.

When validating a path, SURF allows the validation process to change the topology of the path in order to further reduce its length. If an attachment between two consecutive links of the path is valid but can be made invalid while maintaining the planarity of the path, the attachment is modified to become invalid (Figure 78). This results in the attachment operator applied to that attachment, reducing the length of the branch.

---

1. The lower-bound can be computed by considering the finite set of all the pairs and triplets of terminals in the RBS.

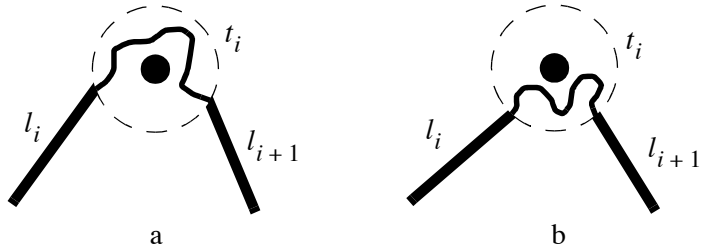


FIGURE 78 - Invalidating an attachment. During the path validation, if a valid attachment between two links of the path (a) can be made invalid while maintaining the planarity of the path (b), SURF change the attachment to be invalid. This causes a further reduction in the path length when the attachment operator is applied to this attachment.

The path P* found by searching G^* is not necessarily a shortest planar path (see example in Figure 75). Even the actual length of P^* (i.e. after it is inserted as a rubber-band branch) is not guaranteed to be a shortest path because the search in G^* can possibly miss a path with shorter actual length because all the paths in G^* representing it are longer than P^* (note that by Lemma 15 a shortest path always has a path in G^* representing it). We are not aware of any bound on the error of the actual length of P^* compared to the shortest planar path. All the empirical results of SURF presented in this thesis were achieved using a search in G^*. A search in G will possibly achieve better results at the expense of longer run-time. To increase the likelihood of the search in G^* resulting in paths of shorter actual length, SURF uses a Constrained Delaunay Triangulation (CDT) [3] [41] which is likely to have shorter paths along triangulation edges. Non-constrained Delaunay Triangulations (DT) [60] [33] [34] are known to closely approximate the complete Euclidean graph with an upper-bound of about 2.42 [28] on the ratio between the distance over CDT edges and the Euclidean length. We are not aware of a similar bound of CDT. That is, a bound on the ratio between the length of a shortest path over CDT edges compared to the length of the shortest path in the complete Euclidean graph excluding edges that intersect with internal points of the constrained triangulation edges.

Table 12 shows experimental results of the ratio between the actual length of P^* and its length in G^*. 10 bins with total of 427 branches where routed[1] using a shortest path search in G^*. The table shows the minimum, maximum and average of the ratios in each of the bins. The lower ratio

in this experiment was 77.72% which implies that a path in P^* was about 29% longer than its length after validation. The maximum of 100% indicates that in each bin, at least one path in G^* was already validated. The average ratio of 95.73% suggest that the length of a shortest path in G^*, on average, is a close approximation to its actual length.

	Branches	Min	Max	Avg
APEX/01	46	79.63	100.00	98.52
APEX/02	105	77.88	100.00	97.47
APEX/03	102	79.69	100.00	98.05
DS15/01	31	83.97	100.00	98.07
DS15/02	28	83.23	100.00	97.67
DS15/03	25	77.87	100.00	96.83
DS15/04	21	85.16	100.00	98.02
GDX/01	27	77.72	100.00	96.51
GDX/02	22	86.91	100.00	97.42
GDX/03	20	88.20	100.00	98.13
Total	427	77.72	100.00	97.73

TABLE 12 - Path actual length / length. This table shows the ratio of paths actual length to their length in G^*. All values are in percentages. The Min, Max, and Avg columns show the minimal, maximal and average ratios respectively of the branches of each of the ten designs. The Total row shows the total minimum, maximum, and average of the respective columns (the total average is weighted relatively to the number of branches in each of the bins).

5.6 Conclusion and Future Work

The concept of Rubber-Band Sketch can be formulated using geometric terms and the concept of limit when spacing approaches zero. This provides a sound framework for proving properties of the RBS. A shortest path in a RBS is closely related to a shortest visible sequence of regions in the RBS, and it can be found in $O((T^2 + S)\log(T + S))$ time by searching in the region visibility

1. Two layers, Euclidean metric, user via cost parameter $\beta = 15$, no optimizations.

graph. By considering only a planar subset of the edges of the graph, the algorithm is guaranteed to find in $O((T+S)\log(T+S))$ time a planar path, if one exists, that is likely to be short.

The proposed algorithm searches for a shortest planar path in the RBS. In practical routing problems, other considerations may be required as well. This includes routability and electrical properties such as cross-talk and delays. Other desired improvements would be finding a least-cost path between two given sub-nets with possible insertion of Steiner points, and a 3-dimensional least-cost path search in a multi-layer RBS.

6 TOPOLOGICAL SKETCH OPTIMIZER (ROAR)

6.1 General

In a addition to the initial routing of the design nets, the algorithm for finding a planar shortest path in an RBS has other applications. One of them is the Rip-Out And Reroute (ROAR) optimizer which accepts a single-layer RBS and tries to find an alternative planar topology with shorter wiring. The ROAR optimization is useful for two reasons, first it compensates for possible bad decisions made by the net ordering step, and second, it overcomes an inherit limitation of sequential routing of the nets on a shortest path that in some cases, no order of the nets will result in an optimal solution (Figure 79).

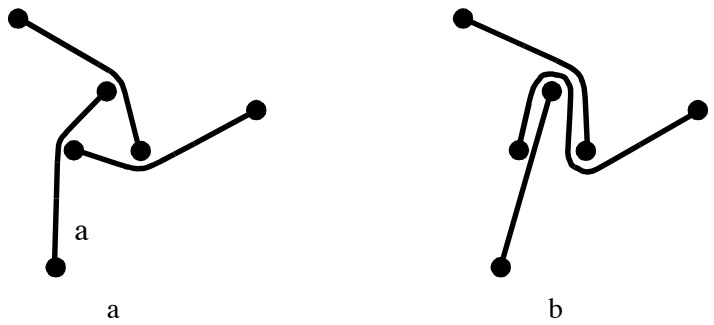


FIGURE 79 - The Triangle problem. This problem has three symmetrical branches and its optimal embedding is shown in (a). The optimal embedding cannot be achieved by routing branch-at-a-time on a shortest path. Any such routing will result in a sub-optimal solution similar to the one in (b). SURF compensates for this limitation using the ROAR optimizer.

The operation of the ROAR optimizer is relatively simple. It picks a terminal t which has an attached local net and one or more incident local nets. Then it removes the branch with the inner-most attachment to t, and all the branches incident to t. Then it reroutes the removed branches using the shortest planar path algorithm, first the branch that was attached, and then the incident ones (the order in which the incident branches are routed is arbitrary). If the re-routing is successful (i.e. planar) and the total wire length of the RBS is reduced then the new RBS is maintained. Otherwise, the operation is undone. This operator is applied iteratively to terminals of

the RBS until the total wire length of the RBS cannot be reduced anymore. Figure 80 shows how the ROAR optimizer improves the example in Figure 79b.

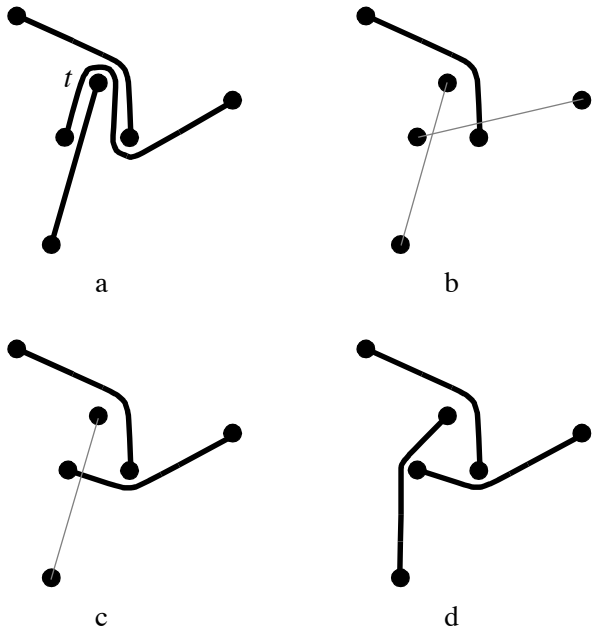


FIGURE 80 - The ROAR operator. This example shows how the optimal solution of the example in Figure 79 is achieved by applying the ROAR operator to terminal *t* of the RBS (a). First, the two branches, attached and incident respectively to *t* are removed (b). Then, the branch that was attached is routed on a shortest path between its end-terminal (c), and then the other branch is routed in a similar way (d). The re-routing is planar and the total wire length is reduced and therefore the new topology is kept. Applying the ROAR operator to any of the other terminals, or to the terminal *t* in (d) will not reduce the total wire length and therefore will be rejected.

6.2 Experimental Results

Table 13 shows experimental results of performing ROAR optimization on 10 bins[1] (total of 427 branches in 20 bin layers). The table shows the *detour* before and after the optimization. The detour of a sketch is defined as the extra wire length compared to its *basic length*. The basic length of a sketch is defined as the sum of the lengths of its nets when they are optimally routed independently of each other. The basic length of the sketch is a lower bound on its total wire length

1. Two layers, Euclidean metric, user via weight parameter $\beta = 35$.

and typically, the basic length is strictly lower than the wire length of the optimal routing (i.e. shortest planar co-routing of the nets). The average detour before the ROAR was 8.84% and it was reduced to 5.18% after optimization. This implies that the wire length was reduced by about 3.4% and that about 38% of the detour length was eliminated.

	Branches	Steps	Detour Before [%]	Detour After [%]	Improv. [%]
APEX/01	76	11	5.06	4.39	0.68
APEX/02	165	66	19.90	11.46	8.44
APEX/03	142	41	11.85	6.60	5.25
DS15/01	49	9	11.10	4.95	6.16
DS15/02	42	4	4.90	4.53	0.37
DS15/03	33	6	12.34	4.78	7.56
DS15/04	31	1	3.05	3.00	0.04
GDX/01	33	11	12.33	7.04	5.29
GDX/02	28	3	4.34	2.99	1.35
GDX/03	24	5	3.56	2.03	1.53
AVG			8.84	5.18	3.67

TABLE 13 - ROAR optimization. This table summaries the result of performing the ROAR optimization on 10 2-layers bins. The Steps column shows the number of successful ROAR operations done. Detour Before and Detour After are the extra wire length compared to the basic length of the sketch. Improv. is the reduction in the detour due to the ROAR optimizer (higher is better).

7 CONCLUSION AND FUTURE WORK

A multi-layer, topological local router was presented. This is the first ever reported router that uses a rubber-band sketch (RBS) to represent the interconnect. The local router is part of SURF, a routing system for multi-chip modules and VLSI that was designed to efficiently handle large multi-layer problems. The local router supports various routing goals and can generate layouts for rectilinear, octilinear and any-angle wiring rules. It performs the routing in four steps of layer-assignment, net-ordering, sequential net embedding, and wire-length reduction using the ROAR optimizer.

The layer-assignment step partitions the multi-layer problem into a set of single-layer sub-problems that are routed independently. It uses a new approach of unconstrained layer-assignment that makes better usage of the routing resources by considering a continuous metric of the conflict between nets as opposed to the binary go/no-go approach previously used. The layer-assignment is formulated as an optimization problem and various routing goals such as wire-length and via minimization or constrained layers can be achieved by simple modifications to the cost function. Our layer-assignment algorithm (LAA) uses a simple optimization technique to solve the layer-assignment problem and uses an optimal algorithm to determine the assignment of individual two-terminal nets.

The net-ordering algorithm uses a continuous conflict metric similar to the one used by the layer-assignment and it results in shorter wiring than the 'shortest first' approach. The nets are embedded sequentially using an optimal algorithm for shortest planar path in an RBS. The algorithm finds a shortest planar path in $O((T^2 + S)\log(T + S))$ time by searching in the region visibility graph. By considering only a planar subset of the edges of the graph, the algorithm is guaranteed to find in $O((T + S)\log(T + S))$ time a planar path (if one exists), that is likely to be short. The ROAR optimization is a simple application of the shortest path algorithm and it uses the 'attachment' relation between nets and terminals in the RBS to determine which nets to re-route. It is guaranteed to maintain the planarity of the sketch it potentially reduces the total wire length. A mathematical formulation of the of RBS was also presented and was used to prove the correctness

of the shortest path algorithm. This is the first exact analysis of RBS ever publish. Empirical results were also shown and they demonstrate the merit and properties of the proposed router.

Several extensions to the our local router are likely to increase the quality of the layouts it generates and will enable it to address a wider range of routing goals. Using more advanced optimization methods that are less likely to be trapped in local minimum will potentially improve the quality of the layer-assignment and the net-ordering. Optimization for electrical properties of the layout, such as cross-talk and signal delays, will make the router more useful for modern high-speed designs. Consideration of routability will result in fewer design-rule violations and will require less manual editing to complete the layout. A post-processing step that uses a 3-dimensional least-cost planar path search can improve the routing by adding vias when necessary, and compensating for ‘bad’ decisions made earlier by the layer-assignment.

8 REFERENCES

[1] R. Bruce, W. Meuli, and J. Ho, *Multichip modules - an Overview*. In Proc. of the 26th Design Automation Conf., pp. 389-393, 1989.

[2] Chen, H.-F.S., Lee, D.T., *A faster algorithm for rubber-band equivalent transformation for planar VLSI layouts*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.15, (no.2):217-27, Feb. 1996.

[3] L. P. Chew, *Constrained Delanuay triangulations*, in Algorithmica, vol 4, pp. 97-108, 1989.

[4] J. D. Cho, S. Raje, M. Sarrafzadeh, M. Sriram, S. M. Kang, *Crosstalk-Minimum layer assignment*, In Proceedings of the IEEE 1993 Custom Integrated Circuits Conference, San Diego, CA, 1993, pp 29.7.1-29.7.4.

[5] R. Cole, A. Siegal, *River Routing every which way, but loose*. In Proc. of 25th Annual Symposium on Foundations of Computer Science. pp 65-73, 1984.

[6] J. Cong, C. L. Liu, *On the k-Layer planar subset and topological via minimization problems*. In IEEE Transactions on Computer-Aided Design, Vol 10, No. 8, Aug 1991.

[7] F. Curatelli, *Switchbox routing with rerouting capabilities in VLSI design*, In IEEE Proceedings, Vol. 137, Pt. G, No. 3, June 1990.

[8] W. M. Dai, R. Kong, J. Jue, M. Sato, *Rubber-band routing and dynamic data representation*. In IEEE Int. Conf. on Computer Aided Design, 1990.

[9] W. M. Dai, T. Dayan, D. Staepelaere, *Topological routing in SURF: generating rubber-band sketch*, In Proc. ACM/IEEE 28th Design Automation Conf. pp 41-44, 1991.

[10] W. M. Dai, R. Kong, M. Sato, *Routability of a rubber-band sketch*, in Proc. ACM/IEEE 28th Design Automation Conf. pp. 45-48, 1991.

[11] T. Dayan, D. W. M. Dai, *Force-driven constrained wiring optimization*, Technical report UCSC-CRL-91-40, CMPE, Univ. of California Santa Cruz, 1991.

[12] T. Dayan, D. W. M. Dai, *Layer assignment for rubber-band routing*, Technical report UCSC-CRL-92-50, CMPE, Univ. of California Santa Cruz, 1992.

[13] P. de Dood, J, Wawrzynek, E. Liu, R. Suaya. *A two dimensional topological compactor with octagonal geometry*. In Proc. of the 28th Design Automation Conf. pp 727-731. June 1991.

[14] D. Z. Du and F.K Hwang, *A proof of the Gilbert-Pollack conjecture on the Steiner Ratio*, Algorithmics 1992 Vol 7.

[15] Shimon Even, *Graph Algorithms*, Computer Science Press Inc., Rockville, Maryland USA, 1979, ISBN 0-914894-21-8.

[16] S. Gao, M. Kaufmann, F. M. Maley, *Advances in homotopic layout compaction*, In Proceedings of the ACM Symposium on Parallel Algorithms and Architectures, 1989, pp 273-82.

[17] M. R. Garey, D. S. Johnson, *Computers and Intractability A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, N. Y., N. Y. USA. pp. 281.

[18] E. N. Gilbert, H. O. Pollak, *Steiner minimal trees*, Bull. Inst. Math. Siam J. Appl. Math. Vol 16, pp. 1-29, 1968

[19] A. Hanafusa, Y. Yamashita, M Yasuda, *Three-dimensional routing for multi-layer ceramic printed circuit boards*, In Proc. IEEE Int. Conf. Computer-Aided Design, pp 386-389, Nov 1990.

[20] S. Haruyzama, D. F. Wong, D. Fussell, *Topological channel routing,* In Proc. 25th ACM IEEE Design Automation Conference, pp 406-409, 1988.

[21] [D. W. Hightower, *A solution to line routing problems on the continuous plane*, In Proc. of the Sixth Design Automation Workshop, pp 1-24, IEEE, 1969.

[22] J. M. Ho, M. Sarrafzadeh, G. Vijayan, C. K. Wong, *Layer assignment for multi-chip modules*, IEEE Trans. CAD, Vol. 9. No. 12, pp 1272-1277, Dec 1990.

[23] J. M. Ho, G. Vijayan, C. K. Wong, *Planar topological routing of pad nets*, In Integration, the VLSI Journal, Vol. 11, pp 295-316, 1991.

[24] C. P. Hsu, *Minimum-via topological routing*, IEEE Trans. Computer Aided Design, Vol. CAD-2, pp 235-246, Oct. 1983.

[25] T. A. Hughes Jr., *Topological routing problems*, Ph.D. dissertation, Dept. of Electrical and Computer Engineering, North Carolina State University, Raleigh, NC, 1992.

[26] Donald A. Joy, Maciej J, Ciesielsky, *Layer assignment for printed circuit boards and integrated circuits*, In Proceedings of the IEEE, Vol. 80, No. 2, Feb. 1992, pp 311-331 (prolog in pp 310).

[27] Kawamura, K.; Shindo, T.; Shibuya, T.; Miwatari, H.; and others. *Touch and cross router*. In 1990 IEEE International Conference on Computer-Aided Design, Santa Clara, CA, USA, 11-15 Nov. 1990.

[28] J. Mark Keil, Carl A. Gutwin, *The Delaunay triangulation closely approximates the complete Euclidean graph*. ACM Computational Geometry, 1989.

[29] Kei-Yong Khoo, Jason Cong. *A fast multi-layer general area router for MCM designs*. In IEEE Transactions on Circuits and Systems, Vol. 39, No. 11, pp 841-851, 1992.

[30] S. Kirkpatric, C. Gellat, M. Vecchi, *Optimization by simulated annealing*, Science, Vol 220, No. 4598, May 1983, pp 671-680.

[31] Raymond Kong, *Incremental routability test for planar topological routing*, Master's thesis, Computer Engineering, University of California Santa Cruz, Dec. 1992.

[32] C. Y. Lee, *An algorithm for path connection and its applications*. IRE Transactions on Electronic Computers, EC-10(3), pp 346-365, 1961

[33] D. T. Lee, B, Schachter, *Two algorithms for constructing a Delaunay triangulation*, Int. J. Comp. Inform. Sci. 9 (1980) pp 219-242

[34] D. T. Lee, A. K. Lin, *Generalized Delaunay triangulation for planar graphs*, Discrete and Computational Geometry, 1 (1986), pp 201-217

[35] C.E. Leiserson and F.M. Maley. *Algorithms for routing and testing routability of planar VLSI layouts*. In Proc. of the 17th Annual ACM Symposium on Theory of Computing, pages 69-78, 1985.

[36] K. F. Kiao, D. T. Lee, M. Sarrafzadeh, *Planar subset of multi-terminal nets*, Integration, The VLSI journal, 10, 1990, pp 19-37.

[37] Ulrich Lauther, *Top down hierarchical routing for channelless gate array based on linear assignment*, In VLSI 87: VLSI Design of Digital Systems, pp 141-151, Elsevier Science Publishers B. B., 1987.

[38] K. F. Liao, M. Sarrafzadeh, *Single-layer global routing*, In IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Jan 1994, Vol. 13, No. 1, pp 38-47.

[39] E. Liu, P. de Dood, R. Suaya, J, Wawrzynek, *A topological framework for compaction and routing*. In Advance Research in VLSI: Proc. of the 1991 University of California Santa Cruz Conf., pp 221-228. The MIT Press, March 1991.

[40] R. Linsker, *An iterative-improvement penalty-function-driven wire routing system*, IBM J. Res. Develop. Vol. 28, No. 5, 1984, pp 613-624.

[41] Yizhi Lu, Wayne Wei-Ming Dai, A *numerically stable algorithm for constructing constrained delaunay triangulation and application to multichip module layout*, Proc. China 1991 Int'l Conf. Circuits and Systems, pp 644-647, 1991.

[42] Yizhi Lu, *Dynamic constrained Delanunay triangulation and application to multichip module layeout*, Master's Thesis, University of California, Santa Cruz, Computer Engineering Dept., Dec. 1991.

[43] F. Miller Maley, *Compaction with automatic jog introduction*. Masters's thesis, MIT, May 1986.

[44] F. Miller Maley, *Single-layer wire routing*, Ph.D.. dissertation, MIT, Cambridge MA, 1987.

[45] F. Miller Maley, *Single-layer wire routing and compaction*, MIT Press, Cambridge MA, 1990.

[46] F. Miller Maley, *Testing homotopic routability under polygonal wiring rules*, Algorithmica, Springer-Verlag, New York, 1996, pp 1-16.

[47] M. Marek-Sadowska, *An unconstrained topological via minimization problem for two-layer routing*, IEEE Trans. Computer-Aided Design, Vol. CAD-3, pp 184-190, July 1984.

[48] M. Marke-Sadowska, *Route planner for custom chip designs*, In IEEE International Conference Computer Aided Design, pp 246-240, 1986.

[49] K. Mehlhorn, S. Naher, *A faster compaction algorithm with automatic jog insertion*, IEEE Transactions on CAD Vol. 9. No. 2, pp 158-166, 1990.

[50] K. Mikami, K. Tabuchi. A *computer program for optimal routing of printed circuit connectors*. IFIPS Proc., H47 pp 1475-1478, 1968.

[51] E. F. Moore, *Shortest path through a maze*, In proc. of the International Symposium on Switching Circuits, pp 285-292, Harvard University Press, Cambridge MA, 1959 (In Annals of the Harvard Computing Laboratory, Vol. 30, Part II).

[52] H. Murata, and Y. Kajitani, *Interactive terminal sliding algorithm for hybrid IC planar layout*, Transactions of Information Processing Society of Japan, Vol.35, No.23, pp.2806-2815, 1994 (in Japanese).

[53] H. Murata, and Y. Kajitani, *Creeping: smooth and flexible layout sketch editing*, to be published.

[54] R. Y. Pinter, *The impact of layer assignment methods on layout algorithms for integrated circuits*, Ph.D. Thesis, EECS Department, MIT, 1982.

[55] R. Y. Pinter, *Optimal layer-assignment for interconnect*, In Proc. IEEE Int. Conf. Circuits and Computers, pp. 398-401, 1982.

[56] R. Y. Pinter, *River routing, methodology and analysis*, In Proceedings of the Third Caltech Conference on VLSI, Computer Science Press, Rockville, MD, 1983, pp 141-163.

[57] M. Raith, M. Bartholomeus, *A new hypergraph based rip-up and reroute strategy*, In Proc. 28th ACM.IEEE Design Automation Conference, San Francisco, CA, USA, pp. 17-21, June 1991.

[58] F. Rubin, *An interactive technique for printed wire routing*, In Proc. 11th Design Automation Workshop, 1974, pp 308-313.

[59] M. Sarrafzadeh, D. T. Lee. *A new approach to topological via minimization*, IEEE Trans. Computer-Aided Design, Vol. 8, pp 890-900, 1989.

[60] M. I. Shamos, D. Hoey, *Closest point problems*, Proc. of the 16th IEEE Symposium on Foundations of Computer Science, pp 151-162, October, 1975.

[61] I. Shirakawa, S. Futagami, A *rerouting scheme for single-layer printed wiring boards*, IEEE Transactions on Computer-Aided Design, Vol. CAD-2, No. 4, October 1983.

[62] David J. Staepelaere, *Geometric transformation for a rubber-band sketch*, Master' thesis, Computer Engineering, University of California Santa Cruz, Sep. 1991.

[63] David J. Staepelaere, Jeffrey Jue, Tal Dayan, Wayne Wei-Ming Dai, *Surf: a rubber-band routing system for multichip modules*, In Proc. IEEE Design and Test of Computers, Dec 1993.

[64] M. Stallmann, T. Hughes, W. Liu, *Unconstrained via minimization for topological multi-layer routing*, In IEEE Trans. on Computer-Aided Design, Vol. 9, No. 9, Sept. 1990.

[65] Jeffrey Z. Su, *Dynamic Updating in Surf Using a Rubber-Band Wiring Model*, 1997, to be published.

[66] J. Valainis, S. Kaptanoglu, E. Liu, R. Suaya, *Two dimensional IC layout compaction based on topological design rule checking*. IEEE Transactions on CAD, 9-3 pp 260-275, march 1990.