An Architecture-Independent Unified Approach to FPGA Routing*

Michael J. Alexander and Gabriel Robins

Department of Computer Science, Thornton Hall,
University of Virginia, Charlottesville, VA 22903-2442

Abstract

Field-programmable gate arrays (FPGAs) are an inexpensive and flexible "low risk" design alternative to custom integrated circuits. While FPGA partitioning and technology mapping have been well-studied, FPGA routing has received much less attention. In this paper we propose a unified general framework for FPGA routing, which allows simultaneous optimization of multiple competing objectives under a smooth designer-controlled tradeoff. Our approach is based on a new and general multi-weighted graph formulation, enabling a good theoretical performance characterization, as well as an effective, practical implementation. Our router is architecture-independent, computationally efficient, and performs very well on industrial benchmarks. Finally, our multi-weighted graph formulation is quite general and is directly applicable to many other areas of combinatorial optimization.

1 Introduction

Field-programmable gate arrays (FPGAs) are an inexpensive and flexible design alternative to custom integrated circuits. FPGAs are reusable high density ASICs that can be easily (re)configured by the user [42], which has made them a popular "low risk" way to implement digital designs [37]. There are a number of commercially available FPGA technologies [37], but generally, an FPGA architecture consist of a *symmetrical array* of user configurable logic "blocks" or "cells" (each of which implements a portion of the design logic), and a set of interconnection resources used for routing [5].

Partitioning and technology mapping in FPGAs has been extensively studied by e.g. [7] [10] [14] [23] [34], where a typical goal is to minimize the maximum input-to-output circuit *depth* (which reduces delay), while varying the total number of logic blocks used (which in turn affects placement and routing feasibility), or some tradeoff between these two goals. For example, the *Rmap* algorithm [35] allows

*Corresponding author is Professor Gabriel Robins, Department of Computer Science, Thornton Hall, University of Virginia, Charlottesville, VA 22903-2442, Email: robins@cs.virginia.edu, phone: (804) 982-2207, FAX: (804) 982-2214.

the user to specify parameters that balance logic block utilization with the goal of producing routable designs during technology mapping; this algorithm attempts to reduce both local and global routing congestion by "dilating" a design according to these parameters. More recent work has addressed the issue of FPGA routability prediction during higher levels of the design cycle [3] [6] [35] [38]. *Routability* is the likelihood of a particular technology mapping and placement solution being feasible to route using the available interconnect resources.

While technology mapping and routability have been studied extensively, less attention has been focused on the actual routing. This is surprising, since researchers have already noted that feasibility in FPGA designs is constrained by *routing* resources more than by logic resources [38]. Moreover it was observed that FPGA performance is often limited by routing delays, rather than by logic block and gate delays [3]. Much of the research on FPGA routing has centered around the SEGA [28] detailed routing algorithm and its predecessor, CGE [4] [5]. Both of these methods are used to route symmetrical-array FPGAs, and allow critical nets to be given higher routing priority. SEGA and CGE use a global router [33] which selects a sequence of channel edges for each connection, which are in turn used to assign edges to specific connections, giving higher priority to edges which least affect other connections. In the technology mapping research of [3], Steiner routing is performed by a global router, with each net treated independently of preceding nets. Other research has adopted a more abstract model of FPGA routing connections to explore Steiner routing of FPGAs [29], or explored issues such as *bend reduction* in FPGA routing [39].

We found that previous works on FPGA routing is lacking for several reasons. First, some research adopts models which are too abstract and are thus not directly applicable to any existing FPGA parts [29]. Other works impose artificial global vs. local hierarchical routing dichotomies, resulting in very poor solutions even for 3-pin nets [4] [5] [28] [33]. Finally, some tools attempt to optimize a particular aspect of FPGA routing (e.g., bend minimization), but fail to unify this goal with other objectives, nor do they allow any tradeoff between competing objectives [39].

We propose the first unified general framework for FPGA routing, where multiple competing objectives can be optimized simultaneously under a smooth designer-controlled tradeoff. Our approach is based on a new and general multi-weighted graph formulation, and has the following advantages over previous work. Our approach:

- escapes the pitfalls of the conventional global/local dichotomy of routing by offering a single

unified and effective method;

- is architecture-independent, and is easily adapted to new FPGA architectural regimes;
- is both computationally efficient and relatively easy to code;
- is parallelizable at both the coarse-grain (i.e., concurrent net routing) and medium-grain (i.e., concurrent computation within a single net) levels;
- offers simultaneous optimization of multiple competing objectives (e.g., wirelength, congestion, jog reduction, etc.), according to a flexible designer-controlled smooth trade-off scheme;
- admits additional new optimization criterion in a seemless and orthogonal manner;
- has a solid theoretical foundation, characterized by provably-good performance bounds; and
- is very general and may be directly applied to other areas of CAD, as well as to classic combinatorial optimization problems (e.g., traveling salesman, graph matching, etc.)

The rest of the paper is organized as follows. In Section 2 we describe a typical symmetrical-array FPGA architecture. Section 3 defines and discusses the FPGA routing problem. In Section 4 we discuss the graph model of FPGAs in detail. Section 5 describes our new hybrid approach to FPGA routing. In Section 6 we develop a theory of multi-weighted graphs, and show how it can be used in the simultaneous optimization of multiple objectives. Section 7 establishes the efficacy of our implementation on industrial benchmark designs, and we conclude in Section 8 with future research directions.

2 A Typical FPGA Architecture

A typical *symmetrical-array* FPGA [5] [42] consists of a rectangular array of logic blocks, separated by channels containing routing resources (i.e., channel edges, connection edges and switchboxes), as illustrated in Figure 1. Logic blocks can be software-(re)programmed to implement arbitrary logic functions; the input and output *pins* of these logic blocks may then be connected using the routing resources, in order to realize the complete system logic design. The routing resources are also software-(re)programmable, where edges are linked together into paths via programmable switches.

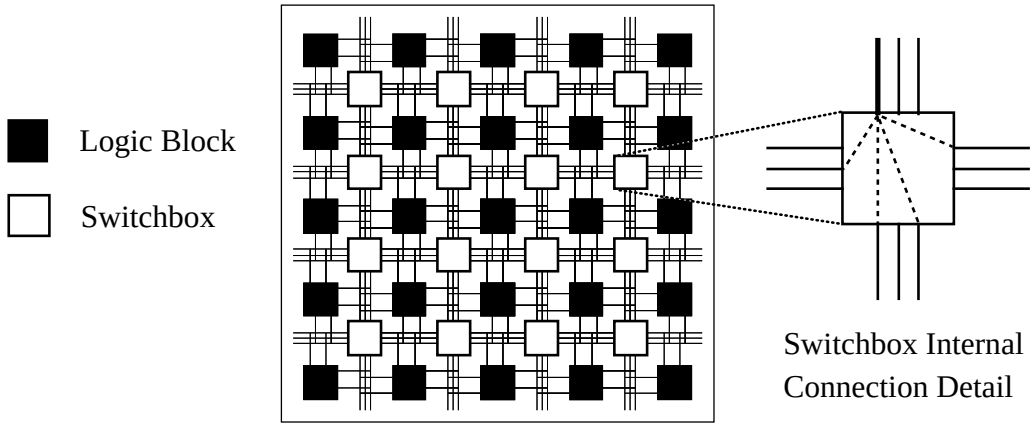


Figure 1: A typical symmetrical-array FPGA architecture; this particular FPGA contains a total of 25 logic blocks arranged in a 5×5 configuration. On the right we see in more detail the switchbox interconnection options for the single channel edge entering the top left of a switchbox; the other edges entering a switchbox have similar interconnection capability.

Each switchbox contains internal edges which can be programmed to connect channel edges on different sides of the switchbox, allowing routing paths to pass through the switchbox (either by going straight through, or by making a right-angle turn). *Switchbox flexibility* [5] is the fanout which the switchbox provides for each channel edge entering the switchbox. Note that in general, a switchbox allows interconnections from a given channel edge to only a subset of the channel edges on the other three sides [5], and this interconnection pattern need not be symmetric [42]. *Connection edges* are used to connect logic block pins to channel edges. *Connection flexibility* [5] is a measure of the number of channel edges to which a logic block pin can attach via connection edges.

Figure 2 illustrates the three types of routing edges: channel edges, connection edges and switchbox edges. These edges are partitioned into *groups*, where a group is a set of edges of the same type which are related by physical proximity. For example, all of the edges inside a switchbox belong to a single group, and all of the channel edges that connect a given pair of switchboxes also belong to a (different) single group. Clearly, a particular symmetrical-array FPGA can be completely characterized via functions describing the switchbox interconnection options, the connection-edge to channel-edge connection options, the number of rows and columns of logic blocks, and channel width (i.e., the number of parallel channel edges per channel).

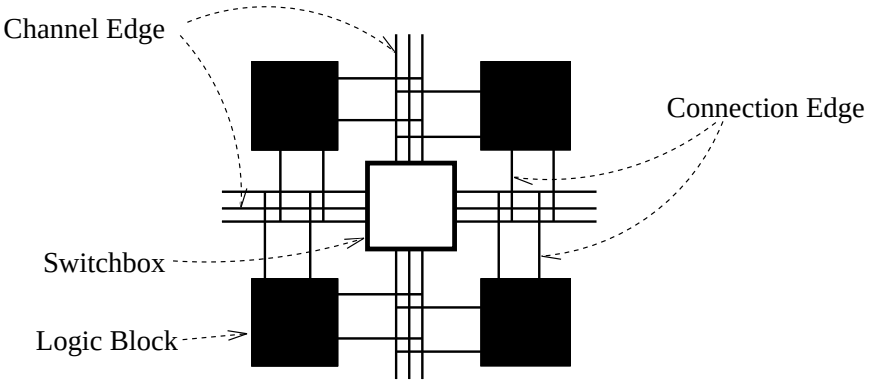


Figure 2: Detail of routing resources for symmetrical-array FPGA showing channel and switchbox edges (edges inside switchbox are not shown).

3 FPGA Routing

VLSI design using FPGAs is a very complex and computationally-intensive task; therefore, in order to bring the combinatorial complexity of this problem down to a manageable realm, FPGA design is traditionally divided into the following general steps: (1) *Partitioning* - the design, often too large to fit on a single FPGA, must be divided into several coarse-grained pieces so that each can fit within one FPGA and be handled independently of the others; (2) *Technology mapping* - for each portion of the design that will be implemented on a single FPGA, that logic must be further divided into many fine-grained fragments, so that each fragment is small enough to be implemented using a single FPGA logic block; (3) *Placement* - each fragment of logic that is to go into each logic block must be assigned an actual free FPGA logic block that will implement it; and (4) *Routing* - the appropriate sets of logic blocks pins (i.e., *nets*) must be connected together using the available FPGA routing resources.

Our work addresses the routing phase of FPGA design. Thus, we assume that partitioning, technology mapping, and placement have already been performed, and our task consists of interconnecting each net in the design without exceeding the available FPGA routing resources (i.e., the routing solution must be *feasible*). The FPGA routing problem is therefore defined as follows:

The FPGA Routing Problem: Given an FPGA architecture along with a configuration of logic block pin assignments and a collection of nets over these pins, route all the nets without exceeding the total available FPGA routing resources.

In traditional VLSI routing regimes, wires can be placed in any available space within the routing region(s); therefore, the main considerations in traditional routing are mostly *geometrical* in nature (i.e., minimizing wirelength, avoiding collisions with other nets, compacting the overall layout, etc.). In contrast, an FPGA architecture offers a limited discrete set of *fixed* routing channels, and the utilization of a given channel edge in one route excludes its usage in the routing of other nets; thus, the FPGA routing problem is essentially *combinatorial* in nature (i.e., satisfying the allowable connectivity, maintaining feasibility, etc.). However, a purely topological approach to FPGA routing may fail to exploit certain available geometrical information, such as physical proximity of pins/blocks on the FPGA, distance from the FPGA "border", etc. On the other hand, a strictly geometrical approach to FPGA routing may fail to take into account the topological/combinatorial constraints induced by the discrete/limited nature of the available FPGA routing resources.

With this in mind, we propose a hybrid FPGA routing framework that combines both geometric and combinatorial techniques, and which simultaneously enjoys the attractive properties of each without undue sacrifices in either complexity or efficiency. Our method depends on and merges two well-studied routing techniques:

1. *geometry-based routing*: the constraints here are mainly geometrical, with the typical objective being to minimize overall wirelength; and
2. *graph-based routing*: the constraints here are mainly topological or combinatorial, with the typical objective being to find a feasible solution given the routing constraints.

Although our proposed framework can use any pair of known existing methods from the above two categories, for implementation purposes we chose two specific candidates, respectively: (a) for the geometry-based routing component, we selected the Iterated 1-Steiner routing method of [21], since it is known to have both excellent empirical performance [2] as well as an efficient implementation [1]; and (b) for the graph-based routing candidate, we chose to use the provably-good graph Steiner approximation scheme of [24], which can also be implemented efficiently [41]. Below we prove that our hybrid of these two methods inherits the best characteristics of its two component methods.

Given that an FPGA routing solution is already feasible, our second major goal is to be able to address secondary criteria, such as congestion reduction, jog minimization, delay optimization, etc., and perhaps additional, yet unconsidered optimizations. Ideally, one would hope to achieve a

smooth designer-controlled tradeoff among such mutually-competing optimization goals. A second major contribution of our work is the simultaneous optimization of such competing goals under a unified framework, where the designer may choose the relative priorities of these various goals. We achieve this by developing in Section 6 a theory of multi-weighted graphs, which is incorporated into our FPGA router. An important added benefit of our multi-weighted graph approach is that it can be used to model several important problems in optimization and operations research.

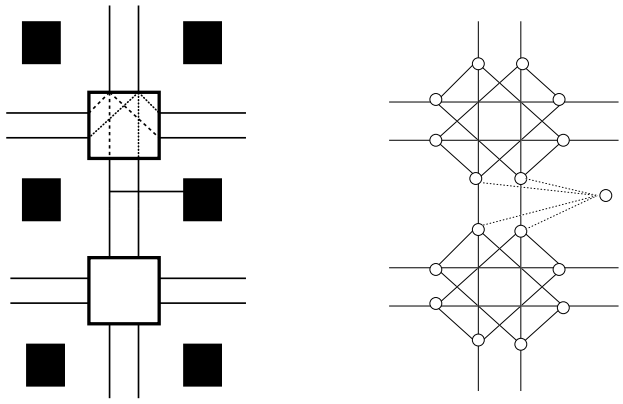
4 Modeling the FPGA as a Graph

Before we can apply graph-based techniques to FPGA routing, we must first model the FPGA as a graph, where the overall graph topology mirrors the complete FPGA architecture. Paths in this graph would correspond to feasible routes on the FPGA, and conversely. Each graph edge is assigned several distinct weights that represent the various optimization criteria. For example, one set of edge weights may represent wire lengths, a second set of edge weights can represent congestion information (measures of congestion are discussed in Section 5.6), still a third set of edge weights can model jog penalties, etc. Searches in the multi-weighted graph consider a weighted average of the values corresponding to the different competing criteria, relative to given designer-selected tradeoff parameters. This technique is very flexible in that new criteria are easily incorporated into the model by introducing additional weight sets into the graph, resulting in a smooth tradeoff and a graceful performance degradation with respect to the various criteria. Such a framework subsumes e.g., "alpha-beta" *routing*, which has been used for jog minimization in IC design [8] [16].

Figure 3 illustrates how the topology of the graph models the architecture of the FPGA. When a net is to be routed, nodes and edges representing the net pins and connection edges are added to the graph, as represented by the dotted lines in Figure 3(b). When the routing of a net is complete, these same nodes and edges are removed from the graph, reflecting the fact that they are no longer usable. Note that in general, the number of nodes corresponding to net pins is typically *much* smaller than the number of nodes corresponding to channel-edge and switchbox-edge endpoints.

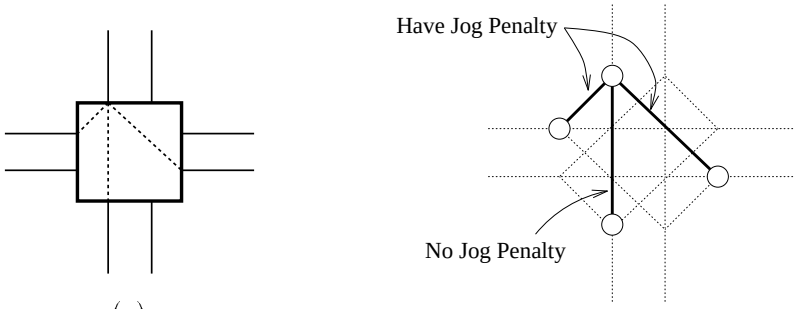
Figure 4 illustrates how jog-penalty weights in the graph reflect right-angle turns in the FPGA routing resources. Since *all* connections between connection edges and channel edges result in right-angles, jog penalties are unnecessary there. If a switchbox edge connects two channel edges that are at right angles to each other, a jog penalty is assigned to that switchbox edge; on the other hand, if

a switchbox edge connects channel edges on opposite sides of the switchbox, then no jog penalty is assigned. This scheme enables our router to prefer straight connections to right-angled ones, and thus promotes jog-minimization.



(a) (b)

Figure 3: Construction of a routing graph for modeling symmetrical-array FPGA.



(a) (b)

Figure 4: Portion of a symmetrical-array FPGA and corresponding routing graph, indicating when switch-box edges are assigned a jog penalty.

5 The FPGA Routing Algorithm

Recall from the discussion above that our FPGA routing method aims to combines geometric and combinatorial techniques. In particular, we would like to hybridize the geometrical Iterated 1-Steiner routing method of [21], with the graph Steiner approximation scheme of [24], referred to as the KMB algorithm. Therefore we first review each of these methods in turn.

5.1 Overview of the Iterated 1-Steiner Method

We begin with some terminology. For a given set P of n points in the plane, an *edge* (i.e., wire) between two points $x \in P$ and $y \in P$ is denoted by (x, y). The *cost* of an edge is the Rectilinear (i.e., Manhattan) distance between its endpoints. A *spanning tree* over P is a set T of $n-1$ edges with endpoints in P such that the induced graph is connected. The *cost* of a tree T, denoted $\overline{\mathrm{T}}$, is the sum of the costs of its edges. A *minimum spanning tree* (MST) is a spanning tree having least cost. Thus we denote the minimum spanning tree itself by "MST" and the *cost* of the minimum spanning tree by "$\overline{\mathrm{MST}}$". A Steiner tree is a spanning tree over the original pointset P and a (possibly empty) additional pointset S (i.e., the *Steiner* points). We are now ready to define the minimum rectilinear steiner tree problem:

The Minimum Rectilinear Steiner Tree (MRST) problem: Given a set P of n points in the Manhattan plane, find a set S of *Steiner points* such that the minimum spanning tree (MST) over $P \cup S$ has minimum cost.

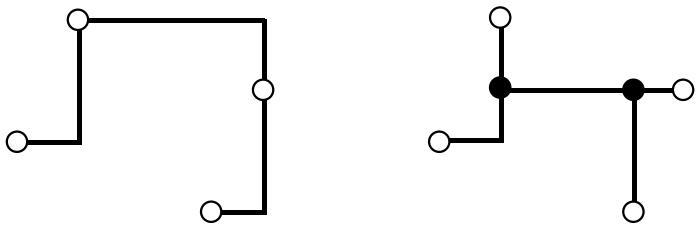


Figure 5: A minimum spanning tree (left) and MRST (right) for a fixed net.

Figure 5 shows an MST and an MRST for a fixed pointset. As with the MST, we denote the Steiner tree itself by MRST and the cost of this tree as $\overline{\mathrm{MRST}}$. Research on the MRST problem has been guided by several fundamental results. First, Hanan [13] has shown that there always exists an MRST

with Steiner points chosen from the intersection of all the horizontal and vertical lines passing through all the points in P (see Figure 6); indeed this result generalizes to all higher dimensions [36]). However, a second major result establishes that despite this restriction on the solution space, the MRST problem remains NP-complete [11], prompting a large number of heuristics, as surveyed in [19].

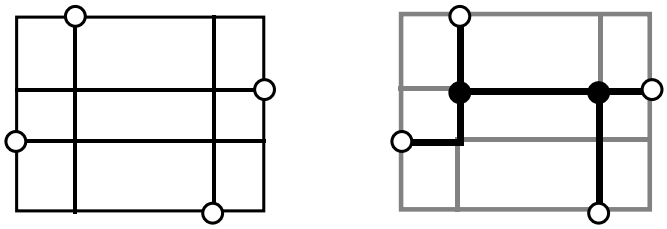


Figure 6: Hanan's theorem: there always exists an MRST with Steiner points chosen from the intersection of all the horizontal and vertical lines passing through all the points.

In solving intractable problems, we often seek provably good heuristics having bounded worst-case error from optimal. Thus, a third important result establishes that the rectilinear MST is a fairly good approximation to the MRST, with a worst-case performance ratio of $\overline{\text{MST}}/\overline{\text{MRST}} \leq \frac{3}{2}$ [17]. This implies that any MST-based strategy which improves upon an initial MST topology will also enjoy a performance ratio of at most $\frac{3}{2}$, which has prompted a large number of Steiner tree heuristics that resemble classic MST construction methods [14] [15] [18] [26] [27], all producing Steiner trees with average cost 7% to 9% smaller than MST cost [31] [40].

Unfortunately, all MST-based MRST constructions were recently shown to have a worst-case performance ratio of exactly $\frac{3}{2}$ [22]. This negative result has motivated research into alternate schemes for MRST approximation, with the best performing among these being the Iterated 1-Steiner (I1S) algorithm [21] [20]. I1S always performs strictly better than $\frac{3}{2}$ times optimal [32], and achieves almost 11% average improvement over MST cost. Efficient serial and parallel implementations of I1S were given in [1], and it was shown in [2] that for typical nets, I1S has average performance of less than 0.25% from optimal, and produce optimal solutions up to 90% of the time. For two pointsets P and S, define the MST savings of S with respect to P as:

$$\Delta\overline{\text{MST}}(P, S) = \overline{\text{MST}}(P) - \overline{\text{MST}}(P \cup S)$$

We use H(P) to denote the set of Hanan Steiner point candidates (i.e., the intersections of all

horizontal and vertical lines passing through points of P). For a pointset P, a 1-*Steiner point* $x \in H(P)$ maximizes $\Delta\overline{\text{MST}}(P, \{x\}) > 0$. The I1S method repeatedly finds 1-Steiner points and includes them into S. The cost of the MST over $P \cup S$ will decrease with each added point, and the construction terminates when there is no x with $\Delta\overline{\text{MST}}(P \cup S, \{x\}) > 0$. Although a Steiner tree may contain at most $n-2$ Steiner points [12], I1S may add more than $n-2$ Steiner points; therefore, at each step we eliminate any extraneous Steiner points having degree 2 or less in the MST. Figure 7 illustrates a sample execution of I1S, and Figure 8 describes the algorithm formally.

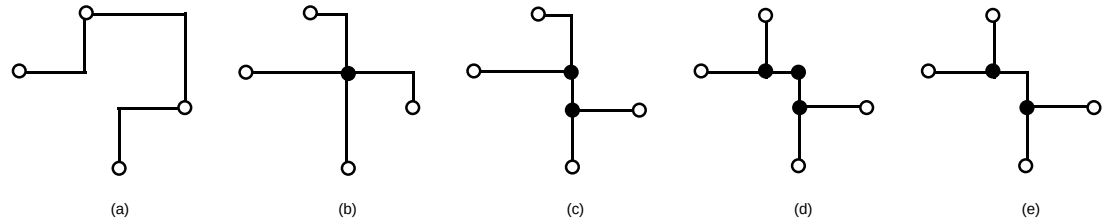


Figure 7: Execution of Iterated 1-Steiner (I1S) on a 4-point example. Note that in step (d) a degree-2 Steiner point is formed and is thus eliminated from the topology (e).

Iterated 1-Steiner (I1S) Algorithm
Input: A set P of n points **Output:** A rectilinear Steiner tree over P

$S = \emptyset$
While $T = \{x \in H(P) \mid \Delta\overline{\text{MST}}(P \cup S, \{x\}) > 0\} \neq \emptyset$ **Do**
  Find $x \in T$ with maximum $\Delta\overline{\text{MST}}(P \cup S, \{x\})$
  $S = S \cup \{x\}$
  Remove from S points with degree ≤ 2 in MST$(P \cup S)$
Output MST$(P \cup S)$

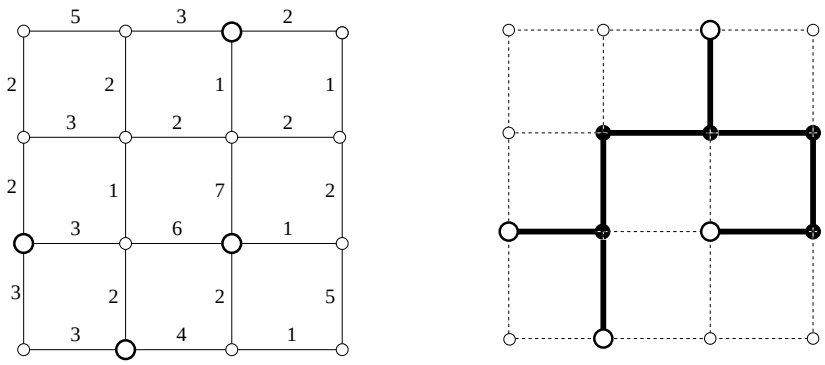
Figure 8: The Iterated 1-Steiner (I1S) algorithm.

5.2 Overview of the KMB Method

Often one encounters the graph version of the Steiner problem, where the problem is topological in nature and is embedded in a given graph $G = (V, E)$, where V is the node set, and $E \subseteq V \times V$ is a set of weighted edges. We are asked to span a subset of the nodes, while using the remaining nodes as Steiner points. Each graph edge e_{ij} has a weight w_{ij}, and edges that are "missing" from the graph are assumed to have an *infinite* weight. Our goal is to minimize the spanning tree total cost. Figure 9 shows a graph and an embedded Steiner minimum tree spanning the highlighted subset of the nodes.

The graph Steiner problem is thus stated as follows:

The Graph Steiner Minimum Tree (GSMT) problem: Given a weighted graph $G = (V, E)$, and a net of terminals $N \subseteq V$ to connect, find a tree $T = (V', E')$ with $N \subseteq V' \subseteq V$ and $E' \subseteq E$ such that $\sum_{e_{ij} \in E'} w_{ij}$ is minimized.



(a) (b)

Figure 9: A graph (left) and a Steiner minimum tree spanning the highlighted nodes.

It is easy to see that the GSMT problem is not any easier to solve than the geometrical MRST counterpart, since the latter is clearly a special case of the former (i.e., given an instance of the MRST problem, transform it into an instance of the GSMT problem by setting $N = P$, collecting all the points and the induced Hanan candidates $V = P \cup H(P)$, and creating the graph $G = (V, V \times V)$, with edge weights being Manhattan distances). It follows that the GSMT problem is NP-complete, and in order to remain computationally efficient, we are thus forced to resort to heuristic solutions to the GSMT problem.

As discussed above, in solving intractable problems, we ideally seek provably-good heuristics having bounded worst-case error from optimal. Indeed the algorithm of Kou, Markowsky and Berman [24] solves the GSMT problem in polynomial time, and is guaranteed to yield solutions never more than $2 \cdot (1 - \frac{1}{L})$ times optimal, where L is the minimum number of leaves in of any optimal solution. We refer to this as the KMB algorithm, and describe it as follows: first, construct the complete graph G' over N with the weight of each edge e_{ij} equal to $\text{dist}_G(N_i, N_j)$, the cost of the corresponding shortest path in G between N_i and N_j. Second, compute MST(G'), the minimum spanning tree of G', and expand each

edge e_{ij} of MST(G') into the corresponding shortest path, denoted path(N_i, N_j), yielding a subgraph G'' that spans N. Finally, compute the minimum spanning tree MST(G''), and delete pendant edges from MST(G'') until all leaves are members of N. The resulting tree is an approximation to the GSMT that has cost no worse than $2 \cdot (1 - \frac{1}{L})$ times optimal, where L is the minimum number of leaves in any optimal Steiner tree [24]. We denote the resulting tree as KMB and the cost of this tree as $\overline{\text{KMB}}$. A formal description of the KMB method is given in Figure 10.

The Kou, Markowsky and Berman (KMB) Algorithm
Input: A graph $G = (V, E)$ with edge weights w_{ij} and a net $N \subseteq V$ **Output:** A low-cost tree $T' = (V', E')$ spanning N (i.e. $N \subseteq V' \subseteq V$ and $E' \subseteq E$)
$G' = (N, N \times N)$, with edge weights w'_{ij} =dist$_G(N_i, N_j)$ **Compute** $T = (N, E'') = \text{MST}(G')$ **Let** the graph $G'' = \cup_{e_{ij} \in E''} \text{path}_G(N_i, N_j)$ **Compute** $T' = \text{MST}(G'')$ **Delete** from T' all leaf nodes that are not in N **Output** T'

Figure 10: The KMB heuristic for the GSMT problem.

Note that there exist more recent graph-based Steiner algorithms with even better performance bounds than that of the KMB method. For example, the Steiner heuristic of [43] has a worst-case performance ratio of $\frac{11}{6}$; however, the time complexity of this method, even though it is within polynomial time, is still rather excessive in practice. Therefore, we prefer to use the KMB method because of its efficiency and ease of implementation [41].

5.3 A New Hybrid FPGA Routing Algorithm

Our approach to FPGA routing is based on combining the geometric I1S heuristic with the KMB graph algorithm. This strategy allows us to model the FPGA architecture in a very natural and general way (i.e., as a graph), as well as to exploit geometrical information about the physical layout of the FPGA in order to yield improved routing solutions. Indeed the resulting hybrid method inherits the excellent average empirical behavior of the I1S algorithm, and it also enjoys the provably-good theoretical performance of the KMB algorithm. We refer to this hybrid method the *Graph Iterated 1-Steiner* (GI1S) algorithm. Note that the present discussion assumes that the graph edges have a single weight each. In Section 6 below, we explain how to extend our routing methodology to multi-weighted graphs.

Our overall GI1S method is basically an adaptation of I1S to graphs; but when we need to span a subset N of the nodes in a graph, the notion of "MST" is no longer well-defined. In essence, a "spanning tree" for N is now actually a "Steiner" tree, and can no longer be computed efficiently (since this is NP-complete and is what we are trying to compute in the first place). The key to this dilemma is to replace the "MST" construction with the KMB construction. In other words, instead of using an "MST" subroutine to determine the "savings" of a candidate Steiner point/node, we use the KMB algorithm for this purpose. Thus, given a graph $G = (V, E)$, the net $N \subseteq V$, and a set of potential Steiner points, we define the following:

$$\Delta\overline{\mathrm{KMB}}_G(N, S) = \overline{\mathrm{KMB}}_G(N) - \overline{\mathrm{KMB}}_G(N \cup S)$$

The GI1S algorithm starts by computing the KMB tree. Then, at each iteration the GI1S method repeatedly finds additional Steiner node candidates that reduce the overall KMB cost and includes them into the growing set of Steiner nodes S. The cost of the KMB tree over $N \cup S$ will decrease with each added node, and the construction terminates when there is no $x \in V$ with $\Delta\overline{\mathrm{KMB}}(N \cup S, \{x\}) > 0$. Naturally, there are numerous other details revolving around our method, mostly dealing with implementation and efficiency issues, and these will be described below. The overall GI1S method is formally described in Figure 11.

Graph Iterated 1-Steiner (GI1S) Algorithm
Input: A weighted graph $G = (V, E)$ and a net $N \subseteq V$ **Output:** A low-cost tree $T' = (V', E')$ spanning N (i.e. $N \subseteq V' \subseteq V$ and $E' \subseteq E$)

$S = \emptyset$
While $T = \{x \in V - N \mid \Delta\overline{\mathrm{KMB}}_G(N \cup S, \{x\}) > 0\} \neq \emptyset$ **Do**
  Find $x \in T$ with maximum $\Delta\overline{\mathrm{KMB}}_G(N \cup S, \{x\})$
  $S = S \cup \{x\}$
  Remove from S Steiner points with degree ≤ 2 in $\mathrm{KMB}(N \cup S)$
Return $\mathrm{KMB}_G(N \cup S)$

Figure 11: The Graph Iterated 1-Steiner algorithm (GI1S) using the KMB heuristic.

Since an optimal Steiner tree for a 3-pin net in a graph can have at most one Steiner point, our GI1S algorithm, which selects the best Steiner point candidate, is guaranteed to find the optimal solution for any 3-pin net. In general, given a weighted graph, and a single arbitrary net, we can show the following:

Theorem 5.1 *The GI1S algorithm will find a routing solution with cost less than* $2 \cdot (1 - \frac{1}{L})$ *times optimal, where* L *is the minimum number of leaves in of any optimal solution.*

Proof: Given a net, the GI1S algorithm constructs a KMB tree, and then attempts to reduce the resulting tree cost further by adding new Steiner nodes. This means that the final overall tree cost can not exceed that of the initial KMB tree, which is known to be no worse than $2 \cdot (1 - \frac{1}{L})$ times optimal, where L is the minimum number of leaves in any optimal Steiner tree [24]. □

Note that [25] have applied the Iterated 1-Steiner method to "escape graphs" in order to perform routing in the presence of obstacles; they also show a general performance bound of two times optimal, and note that 1-Steiner performs optimally for 3-pin nets in a graph.

5.4 Time and Space Complexity

We can upper-bound the time complexity of the GI1S method as follows. Assume that the FPGA is a rectangular array consisting of a total of B switchboxes. Furthermore, assume that each channel is S edges wide, and that each edge is connected inside a switchbox to a constant number other edges (i.e., the "connection flexibility"). The induced routing graph therefore contains $O(B \cdot S)$ vertices and $|G| = O(B \cdot S)$ edges. Thus $|G| = O(B \cdot S)$.

During the execution of GI1S over an n-pin net, shortest path trees are computed from each of the net pins. Each shortest path computation can be performed in time linear in the size of the graph using, e.g., Dijkstra's algorithm [9]. This brings the total time used for this step to $O(n \cdot |G|)$.

To compute the KMB savings of one Steiner candidate with respect to an n-pin net, we first form the complete graph over the n pins, with edge weights being shortest path lengths in the graph; then we can invoke any standard MST algorithm (e.g., Prim's algorithm [30]) which runs within time $O(n^2 \log n)$ or less. Expanding the resulting MST edges back into their original paths can take up to $O(|G|)$ time. Therefore, the entire KMB procedure (to compute the $\Delta\overline{\text{KMB}}$ of a single Steiner candidate) can be done within time $O(n^2 \log n + |G|)$.

Note that in order to select/add a single Steiner point, we may have to test the KMB savings of *most* of the nodes in the graph, bringing the computation time for a single Steiner point to $O(|G| \cdot [n^2 \log n + |G|])$. Finally, since a Steiner tree over n points can have up to $n - 2$ Steiner points [12], the total time to route a single net is bounded above by $O(n \cdot |G| \cdot [n^2 \log n + |G|])$.

The space used by the algorithm is dominated by storing the shortest path trees, one per net pin (and each Steiner node). The total space used by GI1S is therefore $O(n \cdot |G|)$ when routing a net of size n. Note however, that in practice we are able to reduce both the time and space complexity dramatically, by using several effective optimizations, as described in the next Subsection.

5.5 Practical Efficiency Considerations

In this section we present several techniques which improve the efficiency of the "pure" GI1S algorithm formally described in Figure 11. These techniques are complementary: first, we exploit the geometrical properties of the FPGA in order to examine only the 1-Steiner candidates that correspond to "Hanan" points; second, by bounding the search region while routing a net, we further reduce the time to compute shortest paths; third, we use a faster, *batched* version of the Iterated 1-Steiner algorithm to add a large set of Steiner points in a single *round*; forth, we speed up searches by caching shortest path computations with respect to the net pins, and modify GI1S/GB1S by substituting a much less costly computation for $\Delta\overline{\text{KMB}}_G$; and finally, we cut down on the time it takes to route large nets by using a divide-and-conquer technique.

1. We can reduces the number of Steiner candidates which are explored by exploiting geometrical information about the physical layout of the FPGA. Note in the description of GI1S in Figure 11 that Steiner candidates are chosen from the set $V - N$; this set includes, essentially, the nodes in *all* of the switchboxes, and is potentially very large. To reduce the size of this set, we use a second instance of the Iterated 1-Steiner algorithm to select a reduced set of *switchboxes*; only nodes inside this reduced set of switchboxes are explored as Steiner candidates. To compute the reduced set, a complete *switchbox* graph $G_{SB} = (V, E)$ is constructed which captures the physical location of switchboxes on the FPGA. Nodes in V correspond to the switchboxes which are closest to the net pins along a routing, and $E = V \times V$ with edge lengths equal to the physical rectilinear distance between the switchboxes. As explained in Section 5.1, a set of Hanan Steiner-point candidates (which in this case are switchboxes) will be induced in G_{SB}. If $\Delta\overline{\text{MST}}$ in this graph is positive for a Hanan Steiner-point candidate, then the switchbox corresponding to this candidate is included in the reduced set, otherwise it is not. Clearly the switchboxes in the reduced set are those switchboxes which are likely to be "branching" points in the net routing. Figure 12 depicts this process for a 3-pin net; here the set of switchboxes whose nodes are explored

as Steiner candidates is the single darkly shaded switchbox in Figure 12 (b) and (c), rather than all sixteen possible switchboxes.

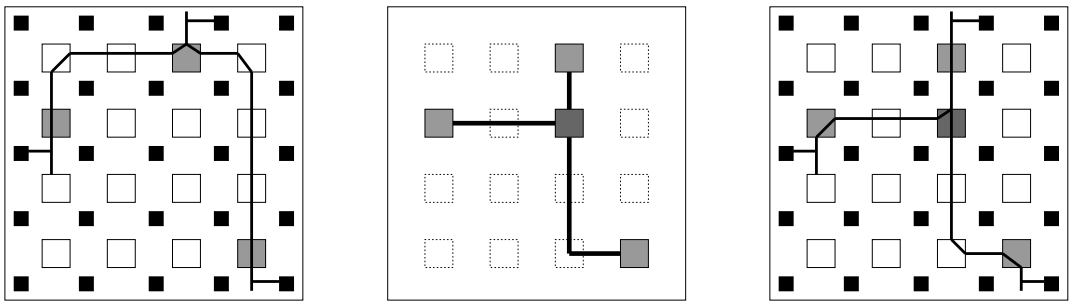


Figure 12: Computing the reduced set of Steiner switchboxes; (a) determining the switchboxes (lightly shaded) used to build G_{SB}; (b) MRST of G_{SB} showing Steiner node (filled circle) corresponding to the shaded switchbox in (c); (c) Steiner switchbox (darkly shaded) used to produce optimal routing.

2. Another time-saving technique is based on the observation that the logic blocks which a net connects are often not spread across the entire FPGA, but are rather located relatively close to each other. We can exploit this situation in order to decrease the run time of our algorithm by limiting the search space to a designer-specified bounding box, i.e., a rectangular subset of the FPGA which encloses all of the logic blocks that a net connects (Figure 13). This allows the designer to control the search area for feasible routes, from as little as the area immediately surrounding the net, to as much as the entire FPGA. Recall from Section 5.2 that the KMB algorithm requires shortest path trees be computed for all net-pin nodes. Since only edges inside the bounding box are considered when, e.g., shortest paths are computed, the execution time is often dramatically reduced. A second benefit of using bounding boxes is the coarse-grain parallelism which they expose: nets whose bounding boxes do not overlap can be routed independently of each other (or indeed in parallel), since the effect of routing a net is usually confined to the nodes and edges interior to the net's bounding box. In case that we find no feasible routing solution within the bounding box, we look again for a routing solution by searching the entire graph.

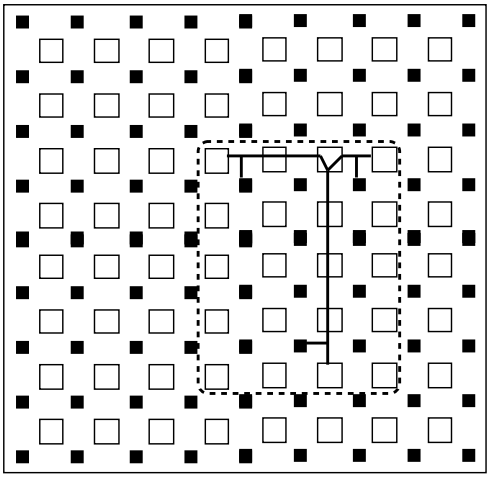


Figure 13: Example of bounding box (dotted line) which encloses the three-pin net.

3. We can speed up the Iterated 1-Steiner computation by efficiently adding an entire set of "independent" Steiner points in a single "batch" or "round" [1] [21] [20], as follows: given a pointset P in the Manhattan plane and a set of Steiner points S which was already added to P, two new candidate Steiner points x and y are *independent* with respect to each other only if:

$$\Delta\overline{\text{MST}}(P \cup S, x) + \Delta\overline{\text{MST}}(P \cup S, y) \leq \Delta\overline{\text{MST}}(P \cup S, \{x, y\})$$

Thus, such Steiner candidates with independent MST savings can be added into the pointset in a single *round*; we call this version of I1S the *Batched* 1-*Steiner* algorithm, and the graph-based adaptation of this *Graph Batched* 1-*Steiner* (GB1S). We use the GB1S variant in our implementation, and describe it formally in Figure 14.

4. Recall from Section 5.2 that in order to build the complete graph G' used to compute $KMB_G(N)$, shortest-path trees in G are computed for all points in N. Note, however, that once these trees are built, if $|S| = 1$, computing $\Delta\overline{\text{MST}}_{G'}(N \cup S)$ can be done very efficiently: since shortest-paths is a reflexive relationship no new shortest-path tree is needed to build the complete graph G' over $N \cup S$. This key observation enables the Iterated 1-Steiner algorithm to quickly explore a large number of Steiner candidates.

   We note that computing $\Delta\overline{\text{KMB}}_G$ will generally take much longer than computing $\Delta\overline{\text{MST}}_{G'}$, as the former requires several additional steps which manipulate the underlying (much larger)

Graph Batched 1-Steiner (GB1S) Algorithm
Input: A graph $G = (V, E)$ with edge weights w_{ij} and a net $N \subseteq V$ **Output:** A low-cost tree $T' = (V', E')$ spanning N (i.e. $N \subseteq V' \subseteq V$ and $E' \subseteq E$)

While $T = \{x \in V - N \mid \Delta\overline{\text{KMB}}_G(N \cup S, \{x\}) > 0\} \neq \emptyset$ **Do**
$S = \emptyset$
For $x \in \{T$ in order of non-increasing ΔMST_G $\}$ **Do**
If $\Delta\overline{\text{KMB}}_G(N \cup S, \{x\}) \geq \Delta\overline{\text{KMB}}_G(N, \{x\})$ **Then** $S = S \cup \{x\}$
$N = N \cup S$
Remove from S Steiner points with degree ≤ 2 in $KMB(N \cup S)$
Return $KMB_G(N \cup S)$

Figure 14: The Graph Batched 1-Steiner algorithm using KMB heuristic.

graph G. Also, it seems intuitive that $\Delta\overline{\text{MST}}_{G'}$ will be a good predictor of $\Delta\overline{\text{KMB}}_G$. Thus for efficiency reasons we replace the **If** statement in Figure 14 with the following one:

> **If** $\Delta\overline{\text{MST}}_{G'}(N \cup S, \{x\}) \geq \Delta\overline{\text{MST}}_{G'}(N, \{x\})$ **Then**
> **If** $\Delta\overline{\text{KMB}}_G(N \cup S, \{x\}) \geq \Delta\overline{\text{KMB}}_G(N, \{x\})$ **Then** $S = S \cup \{x\}$

Clearly the $\Delta\overline{\text{MST}}_{G'}$ computations act as a filter, only invoking the more costly $\Delta\overline{\text{KMB}}_G$ computation for promising Steiner candidates.

5. Finally, we note an additional speedup technique which attempts to reduce the time it takes to route large nets (i.e., which indeed dominates *most* of the execution time). Since the time to route a net is superlinear in the size of the net (indeed, superquadratic even), a simple divide-and-conquer technique is highly effective here: (1) divide the net pins into several smaller nets according to geometric proximity; (2) route each of these smaller nets separately; and (3) connect the resulting separate routings together into a final routing for the original net. This technique would allow large nets to be routed in time comparable to that of much smaller nets.

   The latter practical optimization of breaking a net into m more or less equal-sized fragments and routing each fragment separately, will also reduce the space requirement of the GI1S/GB1S algorithms (by a factor of m). This is true because now the same space can be reused to route each net fragment in turn. Alternatively, since the processing of large nets dominates the execution time, we may choose to route several large nets first, and then "fix" their routing in the graph. When different net routing orders are later tried, these "fixed" large nets need not be routed from scratch again, since their previously-computed routing is already "embedded" in the graph.

Recall from Section 5.4 above that the total time to route a single n-pin net is bounded by $O(n \cdot |G| \cdot [n^2 \log n + |G|])$, where $|G|$ is the size of the FPGA routing graph. However, using the practical efficiency enhancements listed above, we can reduce this time complexity substantially. For example, by not expanding *all* of the MSTs into their corresponding paths (only the ones which seemed superior to the rest), we are able to reduce the time complexity to $O(n \cdot |G| \cdot n^2 \log n)$. Another practical reduction in the time complexity follows from the observation that the number of "rounds" in B1S is in practice a small constant (less than 3) [21]; this eliminates an additional factor of n off the time complexity. Finally, by exploiting the underlying geometry of the FPGA in order to only test potential Steiner points that are near switchboxes corresponding to "Hanan" candidates (i.e., $n^2 << |G|$), we can reduce the time complexity of routing a single net to $O(n \cdot |G| + n^4 \log n)$ (since the shortest paths computations still require $O(n \cdot |G|)$ time).

5.6 Routing Multiple Nets

We use the notion of *congestion* as a measure of resource utilization within a group of routing edges. Clearly as congestion within a group increases, that group becomes much less "usable" with respect to the routing of future nets, with routing through a group becoming altogether impossible when its congestion reaches a high enough value. Therefore, rather than risk concentrated congestion forming absolute routing blockages, we prefer to "spread" the congestion around different groups, so as to retain the future feasibility of as many routing paths as possible.

With this in mind, we model congestion in our routing graph as follows. After a net has been successfully routed, the graph is updated in two ways. First, edges used to route the net are no longer available to route subsequent nets and must be removed from the graph. In our implementation, rather than actually removing these edges from the graph, they are marked "unavailable." Our shortest-path algorithm treats edges that are "unavailable" as if they were not present in the graph, so such edges are never explored or used in subsequent routing. Second, the graph is updated to reflect the additional congestion resulting from the placement of this net on the FPGA.

Recall that every edge belongs to a *group* consisting of those edges which are related by physical proximity. Every edge used to route the net updates the graph weights as follows: the congestion weight for every edge in this edge's group is increased. This is done to reflect the fact that fewer of this group's resources are now available (since one edge was just used). The congestion value in our

model is the number of edges in the group which have been taken by previous routings (the number "unavailable"). Thus when the graph has been updated to reflect the successful routing of a net, the edges used by this routing are no longer available, and the edges "nearby" this routing contain higher congestion values than before.

Under this strategy, when routing nets, the congestion in the graph will tend to *spread* over the FPGA the resources used by the net, increasing the balance of resource utilization and hence the feasibility of routing subsequent nets. The congestion weights are updated as routing proceeds, to reflect the current resource usage. We use a move-to-front heuristic when infeasibility is encountered: after an attempt has been made to route all nets, those nets which could not be routed are given a higher priority by moving them to the front of the net-routing order. The routing graph is re-initialized to its initial state and the new routing order is tried.

6 Simultaneous Optimization of Multiple Objectives

Recall that ideally we wish to simultaneously optimize k multiple (possibly competing) objectives (i.e., wirelength, jogs, congestion, etc.). We now show how to accomplish this, by generalizing the GB1S heuristic to multi-weighted graphs, where each optimization criterion has a separate set of graph edge weights. The simultaneous optimization is accomplished by transforming these multiple edge weights into a single weighted average, which is then used in computation by GB1S in the normal way. The relative magnitudes of the weighing factors (i.e., tradeoff parameters) are completely controlled by the circuit designer, and generally consist of k real numbers $d_1, d_2, \cdots, d_k$ where $\sum_{i=1}^{k} d_i = 1$. This scheme enables a unified smooth tradeoff among the various competing objectives. We now formalize and develop a framework of multi-weighted graphs.

6.1 A Theory of Multi-Weighted Graphs

Let $V = \{v_1, v_2, \cdots, v_n\}$ be a set of $|V| = n$ vertices, and let $E \subseteq V \times V$ be a set of $|E| = m$ edges. We define a k-weighted graph $G = (V, E)$ to be a weighted graph with a vector-valued weight function $w : E \rightarrow \Re^k$. In other words, associated with each edge $e_{ij} \in E$ is a *vector* of k real-valued weights $\vec{w}_{ij} = (w_{ij\,1}, w_{ij\,2}, \cdots, w_{ij\,k})$. Note that ordinary weighted graph are a special case of k-weighted graphs, with $k = 1$.

Let $\vec{d} = (d_1, d_2, \cdots, d_k)$ be a vector of k real-valued *tradeoff parameters*, where $0 \leq d_i \leq 1$ for $0 \leq i \leq k$, and $\sum_{i=1}^{k} d_i = 1$. ¿From the k-weighted graph $G = (V, E)$ and the tradeoff parameters $\vec{d}$ we construct a new weighted *tradeoff* graph $\widehat{G}(\vec{d}) = (V, E)$ with weight function $w'_{ij} = \vec{d} \cdot \vec{w}_{ij} = \sum_{m=1}^{k} d_m \cdot w_{ij}$. Clearly the tradeoff graph $\widehat{G}$ is an ordinary weighted graph having the same topology as G, but whose single edge weights represent the weighted averages of the multi-weights of G, with respect to the tradeoff parameters $\vec{d}$.

Let $\vec{u} = (1, 1, \ldots, 1)$ be the unit vector, and let $\vec{v}_i = (0, 0, \cdots, 0, v_i, 0, 0, \cdots, 0)$ denote the vector obtained from the vector $\vec{v}$ by using v_i in the i-th place, and the rest of the places being set to zero. Thus, $\vec{u}_i$ denotes the vector consisting of zeros everywhere except the ith place, which will contain a "1". Now a k-weighted graph G can induce k distinct graphs, each with an identical topology but with edge weights restricted to only one of the k components of vector-valued weight function $\vec{w}$; these k induced graphs are denoted by $G_i = \widehat{G}(\vec{u}_i)$.

We define the minimum spanning tree (MST) for a multi-weighted graph G with respect to the tradeoff parameters $\vec{d}$ as the "normal" MST over the tradeoff graph $\widehat{G}(\vec{d})$, and denote it by MST($\widehat{G}(\vec{d})$). Similarly, we can compute the MST on each of the k induced graphs G_i, and we denote these MST(G_i). Again, MST denotes the resulting tree and $\overline{\text{MST}}$ denotes the cost of this tree.

To give a concrete application of multi-weighted graphs, let $k = 2$ and construct a 2-weighted graph G over n sites, where w_{ij1} represents the *cost* of laying down a fiber optic cable between site i and site j, and where w_{ij2} represents the *time* it would take a team to accomplish that job (see Figure 15). Our goal is to utilize a single construction team to build a fiber optic network that spans the n sites. Clearly, MST($\widehat{G}((1, 0))$) denotes the cheapest possible network, while MST($\widehat{G}((0, 1))$) denotes the network that can be built the fastest (in terms of total worker-time). Finally, MST($\widehat{G}((\frac{1}{2}, \frac{1}{2}))$) represents the network that simultaneously optimizes both the construction time as well as the building cost, with both objectives being equally important (i.e., $d_1 = d_2 = \frac{1}{2}$).

Given a k-weighted graph G and a parameter vector $\vec{d}$, it would be of interest to bound $\overline{\text{MST}}(\widehat{G}(\vec{d}))$ both from above and below in terms of $\overline{\text{MST}}(G_1)$ through $\overline{\text{MST}}(G_k)$, $\vec{d}$, and n. Below we show both upper and a lower bounds for metric graphs (i.e., graphs with weight functions satisfying the triangle inequality dist(a, b)+ dist$(b, c) \leq$ dist(a, c), $\forall a, b, c \in V$):

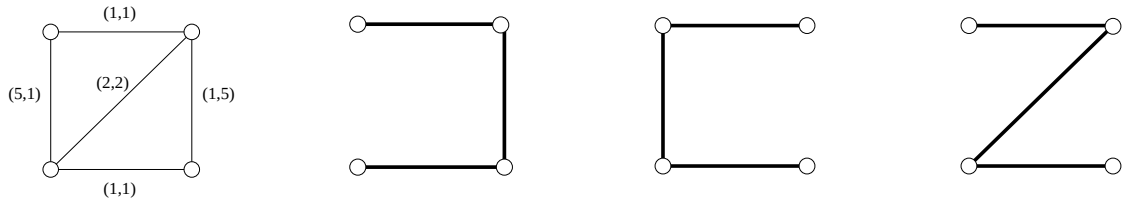


Figure 15: (a) A 2-weighted graph G, and its two induced graphs (b) ($\widehat{G}((1,0))$) with cost 3, and (c) ($\widehat{G}((0,1))$) with cost 3; (d) shows the MST over the tradeoff graph: $\mathrm{MST}(\widehat{G}((\frac{1}{2},\frac{1}{2})))$ with cost 4.

$$\sum_{i=1}^{k} d_i \cdot \overline{\mathrm{MST}}(G_i) \;\leq\; \overline{\mathrm{MST}}(\widehat{G}(\vec{d})) \;\leq\; (n-1) \cdot \sum_{i=1}^{k} d_i \cdot \overline{\mathrm{MST}}(G_i)$$

We also show that while the above lower bound holds in general, for arbitrary (non-metric) weighted graphs there exists no upper bound strictly in terms of the $\overline{\mathrm{MST}}(G_i)$'s, $\vec{d}$, and n. We will also show that even for metric graphs, the above lower bound is tight, while the upper bound is not; for example, we will show a tighter upper bound of $\overline{\mathrm{MST}}(\widehat{G}(\vec{d})) \leq \frac{3}{2} \cdot \sum_{i=1}^{k} d_i \cdot \overline{\mathrm{MST}}(G_i)$ for metrical graphs over three nodes, which is of significance since most nets in typical VLSI designs contain 3 or less pins [25].

We start by showing a general lower bound for the cost of $\overline{\mathrm{MST}}(\widehat{G}(\vec{d}))$ in terms of $\overline{\mathrm{MST}}(G_i)$'s, $\vec{d}$, and n:

Theorem 6.1 *For any k-weighted graph G over n vertices, and tradeoff parameters $\vec{d}$, $\sum_{i=1}^{k} d_i \cdot \overline{\mathrm{MST}}(G_i) \leq \overline{\mathrm{MST}}(\widehat{G}(\vec{d}))$.*

Proof: Consider an arbitrary edge e_{ij} in $\mathrm{MST}(\widehat{G}(\vec{d}))$ having cost of $\sum_{m=1}^{k} d_m \cdot w_{ijm}$. If *every* $\mathrm{MST}(G_m)$, $1 \leq m \leq k$, also contains edge e_{ij}, then clearly the cost of edge e_{ij} in all k trees is $\sum_{m=1}^{k} w_{ijm}$, and the cost of this edge scaled by the tradeoff parameters $\vec{d}$ is $\sum_{m=1}^{k} d_m \cdot w_{ijm}$, which is equal to the cost of this edge in $\mathrm{MST}(\widehat{G}(\vec{d}))$. Clearly, if *all* of the k $\mathrm{MST}(G_m)$, $1 \leq m \leq k$, chose the same edges as $\mathrm{MST}(\widehat{G}(\vec{d}))$,

then equality holds and the theorem is true. On the other hand, if $\mathrm{MST}(\widehat{G}(\vec{d}))$ contains an edge that is not in $\mathrm{MST}(G_m)$, $1 \leq m \leq k$, then the cost of $\mathrm{MST}(\widehat{G}(\vec{d}))$ relative to $\sum_{m=1}^{k} d_m \cdot \overline{\mathrm{MST}}(G_m)$ can only increase. □

Next, we show the non-existence of general upper bounds. Ideally, we would like to bound the MST cost of arbitrary multi-weighted graphs in terms of only the costs of the $\mathrm{MST}(G_i)$'s, $\vec{d}$, and n. Unfortunately, this property does not hold in general:

Theorem 6.2 *For any k-weighted graph G over n vertices, and tradeoff parameters $\vec{d}$, the tradeoff graph cost $\overline{\mathrm{MST}}(\widehat{G}(\vec{d}))$ can not be bounded from above by any function of only $\overline{\mathrm{MST}}(G_i)$'s, $\vec{d}$, n, and k.*

Proof: Consider the doubly-weighted graph $G = (V, E)$ over $n = 3$ nodes, where $k = 2$. Fix $\vec{d}$ by setting $0 < d_1, d_2 < 1$. Let M be some very large number, $V = \{a, b, c\}$, and $E = V \times V$. Let $w_{ab1} = 0$, $w_{bc1} = 0$, $w_{ac1} = M$ and $w_{ab2} = M$, $w_{bc2} = 0$, $w_{ac2} = 0$ (see Figure 16). Observe that $\overline{\mathrm{MST}}(G_1) = \overline{\mathrm{MST}}(G_2) = 0$, $k = 2$, $n = 3$, d_1, and d_2 are all constants. On the other hand, $\overline{\mathrm{MST}}(\widehat{G}) = min(d_1 \cdot M, d_2 \cdot M)$, which can be made arbitrarily large for any fixed $\vec{d}$ by making M large enough. Clearly, any expression in terms of a set of constants must also be constant; therefore $\overline{\mathrm{MST}}(\widehat{G})$ can not be bounded from above by any function purely in terms of only $\overline{\mathrm{MST}}(G_1)$, $\overline{\mathrm{MST}}(G_2)$, k, n, and $\vec{d}$. □

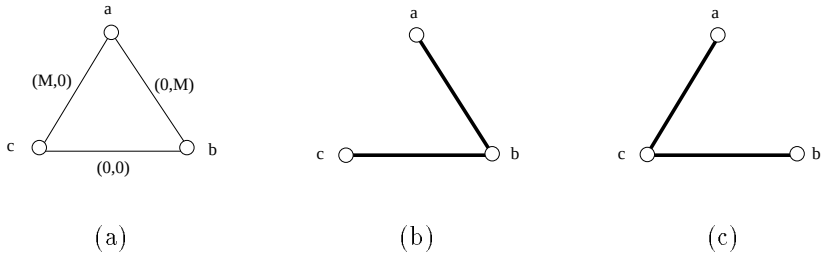


Figure 16: An example showing that $\overline{\mathrm{MST}}(\widehat{G}(\vec{d}))$ can not be bounded from above by any function strictly in terms of $\overline{\mathrm{MST}}(G_i)$'s, $\vec{d}$, n, and k: (a) The 2-weighted graph G; (b) $\mathrm{MST}(G((1,0)))$ has cost 0; (c) $\mathrm{MST}(G((0,1)))$ has cost 0. On the other hand, $\mathrm{MST}(G((\frac{1}{2},\frac{1}{2})))$ has cost $\frac{M}{2}$, which can be arbitrarily large.

The negative result of Theorem 6.2 only applies to non-metric graphs. We now give a general upper bound for metric graphs:

Theorem 6.3 *For any* metric *k-weighted graph G over n vertices, and tradeoff parameters $\vec{d}$,* $\overline{\text{MST}}(\widehat{G}(\vec{d}))$ *is bounded by:* $\overline{\text{MST}}(\widehat{G}(\vec{d})) \leq (n-1) \cdot \sum_{i=1}^{k} d_i \cdot \overline{\text{MST}}(G_i)$

Proof: Consider an arbitrary edge e_{ij} in $\text{MST}(\widehat{G}(\vec{d}))$ and its cost, $\sum_{m=1}^{k} d_m \cdot w_{ijk}$. Consider element m' in this summation, and the corresponding MST of $G_{m'}$. $\text{MST}(G_{m'})$ spans vertices v_i and v_j, but does not necessarily contain the edge e_{ij}. Rather, a path must exist in $\text{MST}(G_{m'})$ from v_i to v_j, denoted $\text{path}_{MST(G_{m'})}(i,j)$. By metricity, $w_{ijm'} \leq \text{path}_{MST(G_{m'})}(i,j)$. Therefore:

$$\begin{aligned}
\text{cost of edge } e_{ij} \text{ in } \text{MST}(\widehat{G}(\vec{d})) &= \sum_{m=1}^{k} d_m \cdot w_{ijm} \\
&\leq \sum_{l=1}^{k} d_m \cdot \text{path}_{MST(G_{m'})}(i,j) \qquad \text{(by metricity)} \\
&\leq \sum_{l=1}^{k} d_m \cdot \overline{\text{MST}}(G'_m)
\end{aligned}$$

Since e_{ij} is an arbitrary edge of $\text{MST}(\widehat{G}(\vec{d}))$, this holds for *all* $n-1$ edges in $\text{MST}(\widehat{G}(\vec{d}))$. Thus, $\overline{\text{MST}}(\widehat{G}(\vec{d}))) \leq (n-1)\sum_{m=1}^{k} d_m \cdot \overline{\text{MST}}(G_m)$, and the theorem holds. □

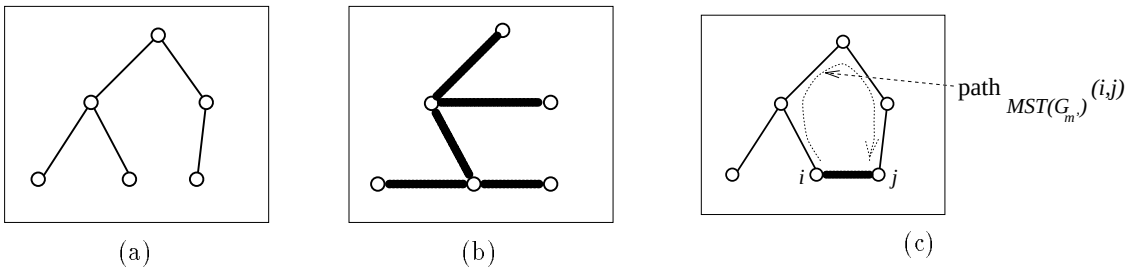


Figure 17: A general upper bound in the metric case: for $\overline{\text{MST}}(\widehat{G}(\vec{d}))$ in terms of $\overline{\text{MST}}(G_i)$'s, $\vec{d}$, n, and k: (a) depicts $\text{MST}(G_m)$; (b) depicts $\text{MST}(\widehat{G}(\vec{d}))$; and (c) shows how the cost of the m^{th} weight component of each e_{ij} can be bounded by $d_m \cdot \overline{\text{MST}}(G_m)$.

Since many nets in VLSI design contain three pins or less [25], we now derive a tighter upper bound for three pin nets where metricity holds. But first, we note the following Lemma:

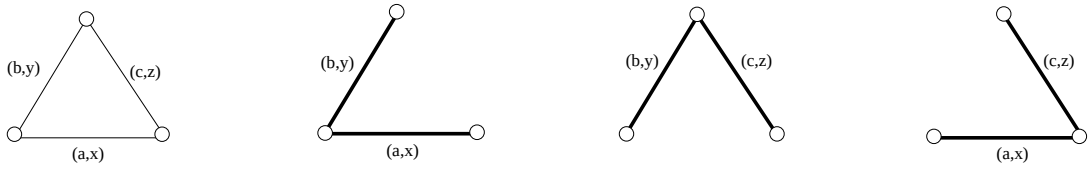
Lemma 6.4 *If $y \leq z$, then $y \leq \frac{y+z}{2}$.*

Proof: Adding y to both sides of $y \leq z$ yields $2y \leq y + z$, and dividing both sides of the latter by 2 yields $y \leq \frac{y+z}{2}$. □

Theorem 6.5 *For 2-weighted metric graphs with three nodes, and any scaling vector $\vec{d} = (d_1, d_2)$, the following holds:*

$$d_1 \cdot \overline{\text{MST}}(G_1) + d_2 \cdot \overline{\text{MST}}(G_2) \leq \overline{\text{MST}}(\widehat{G}(\vec{d})) \leq \frac{3}{2} \cdot [d_1 \cdot \overline{\text{MST}}(G_1) + d_2 \cdot \overline{\text{MST}}(G_2)]$$

Proof: Let $G = (V, E)$ be a 3-node 2-weighted graph, with edge weights (a, x), (b, y), and (c, z). Let $\vec{d} = (d_1, d_2)$ be an arbitrary constant vector, such that $0 \leq d_1, d_2 \leq 1$, and $d_1 + d_2 = 1$ (see Figure 18(a)).



(a) (b) (c) (d)

Figure 18: A tighter upper bound for 3-pin nets (a); we assume without loss of generality that $a \leq b \leq c$, and each case shows $\text{MST}(G_2)$ for a different relation between x, y, and z: (b) $x \leq y \leq z$; (c) $y \leq z \leq x$; and (d) $z \leq x \leq y$.

Clearly, the lower bound $d_1 \cdot \overline{\text{MST}}(G_1) + d_2 \cdot \overline{\text{MST}}(G_2) \leq \overline{\text{MST}}(\widehat{G}(\vec{d}))$ holds by Theorem 6.1. Assume without loss of generality that $a \leq b \leq c$, which implies that $\overline{\text{MST}}(G_1) = a + b$. The following three cases must be considered:

1. Assume $x \leq y \leq z$, which implies that $\overline{\text{MST}}(G_2) = x + y$ (see Figure 18(b)). Thus,

$$\begin{aligned} d_1 \cdot a + d_2 \cdot x &\leq d_1 \cdot c + d_2 \cdot z \quad \text{and} \\ d_1 \cdot b + d_2 \cdot y &\leq d_1 \cdot c + d_2 \cdot z \end{aligned}$$

$$\begin{aligned} \text{Now} \quad \overline{\text{MST}}(\widehat{G}(\vec{d})) &= d_1 \cdot a + d_2 \cdot x + d_1 \cdot b + d_2 \cdot y \\ &= d_1(\cdot a + b) + d_2 \cdot (x + y) \\ &= d_1 \cdot \overline{\text{MST}}(G_1) + d_2 \cdot \overline{\text{MST}}(G_2) \end{aligned}$$

and the theorem holds.

2. Assume $y \le z \le x$, which implies that $\overline{\mathrm{MST}}(G_2) = y + z$. (see Figure 18(c)). Note that $b \le c$ and $y \le z$, so $d_1 \cdot b + d_2 \cdot y \le d_1 c + d_2 z$; thus $\mathrm{MST}(\widehat{G}(\vec{d}))$ must contain the "b/y" edge. Now, the other edge of $\mathrm{MST}(\widehat{G}(\vec{d}))$ must be either the "a/x" edge e_{de} or the "c/z" edge; we analyze each of these cases in turn.

   (a) Assume $\mathrm{MST}(\widehat{G}(\vec{d}))$ contains the "a/x" edge.

$$\begin{aligned} \text{Then} \quad \overline{\mathrm{MST}}(\widehat{G}(\vec{d})) &= d_1 \cdot b + d_2 \cdot y + d_1 \cdot a + d_2 \cdot x \\ &= d_1 \cdot (a+b) + d_2 \cdot (x+y) \\ &\ge d_1 \cdot (a+b) + d_2 \cdot (z+y) \qquad (\text{since} \quad x \ge z) \\ &= d_1 \cdot \overline{\mathrm{MST}}(G_1) + d_2 \cdot \overline{\mathrm{MST}}(G_2) \end{aligned}$$

$$\begin{aligned} \text{Also} \quad \overline{\mathrm{MST}}(\widehat{G}(\vec{d})) &= d_1 \cdot (a+b) + d_2 \cdot (x+y) \\ &\le d_1 \cdot (a+b) + d_2 \cdot ((y+z)+y) \qquad (\text{by metricity}) \\ &= d_1 \cdot \overline{\mathrm{MST}}(G_1) + d_2 \cdot \overline{\mathrm{MST}}(G_2) + d_2 \cdot y \\ &\le d_1 \cdot \overline{\mathrm{MST}}(G_1) + d_2 \cdot \overline{\mathrm{MST}}(G_2) + \frac{d_2}{2} \cdot \overline{\mathrm{MST}}(G_2) \qquad (\text{by Lemma 6.4}) \\ &= d_1 \cdot \overline{\mathrm{MST}}(G_1) + \frac{3}{2} \cdot d_2 \cdot \overline{\mathrm{MST}}(G_2) \end{aligned}$$

   (b) Assume $\mathrm{MST}(\widehat{G}(\vec{d}))$ contains the "c/z" edge.

$$\begin{aligned} \text{Then} \quad \overline{\mathrm{MST}}(\widehat{G}(\vec{d})) &= d_1 \cdot b + d_2 \cdot y + d_1 \cdot c + d_2 \cdot z \\ &= d_1 \cdot (b+c) + d_2 \cdot (y+z) \\ &\ge d_1 \cdot (b+a) + d_2 \cdot (z+y) \quad \text{since} \quad c \ge a \\ &= d_1 \cdot \overline{\mathrm{MST}}(G_1) + d_2 \cdot \overline{\mathrm{MST}}(G_2) \end{aligned}$$

$$\begin{aligned} \text{Also} \quad \overline{\mathrm{MST}}(\widehat{G}(\vec{d})) &= d_1 \cdot (b+c) + d_2 \cdot (y+z) \\ &\le d_1 \cdot (b+(a+b)) + d_2 \cdot (y+z) \qquad (\text{by metricity}) \\ &= d_1 \cdot b + d_1 \cdot \overline{\mathrm{MST}}(G_1) + d_2 \cdot \overline{\mathrm{MST}}(G_2) \\ &\le \frac{d_1}{2} \cdot \overline{\mathrm{MST}}(G_1) + d_1 \cdot \overline{\mathrm{MST}}(G_1) + d_2 \cdot \overline{\mathrm{MST}}(G_2) \qquad (\text{by Lemma 6.4}) \\ &= \frac{3}{2} \cdot d_1 \cdot \overline{\mathrm{MST}}(G_1) + d_2 \cdot \overline{\mathrm{MST}}(G_2) \end{aligned}$$

In both cases 2a and 2b we have $\overline{\text{MST}}(\widehat{G}(\vec{d})) \geq d_1 \cdot \overline{\text{MST}}(G_1) + d_2 \cdot \overline{\text{MST}}(G_2)$. Also, combining

$$\begin{aligned}
\overline{\text{MST}}(\widehat{G}(\vec{d})) &\leq d_1 \cdot \overline{\text{MST}}(G_1) + \frac{3}{2} \cdot d_2 \cdot \overline{\text{MST}}(G_2) \quad \text{and} \\
\overline{\text{MST}}(\widehat{G}(\vec{d})) &\leq \frac{3}{2} \cdot d_1 \cdot \overline{\text{MST}}(G_1) + d_2 \cdot \overline{\text{MST}}(G_2) \\
\text{yields} \quad \overline{\text{MST}}(\widehat{G}(\vec{d})) &\leq \max[d_1 \cdot \overline{\text{MST}}(G_1) + \frac{3}{2} \cdot d_2 \cdot \overline{\text{MST}}(G_2), \\
&\qquad \frac{3}{2} \cdot d_1 \cdot \overline{\text{MST}}(G_1) + d_2 \cdot \overline{\text{MST}}(G_2)] \\
\overline{\text{MST}}(\widehat{G}(\vec{d})) &\leq \frac{3}{2} \cdot [d_1 \cdot \overline{\text{MST}}(G_1) + d_2 \cdot \overline{\text{MST}}(G_2)]
\end{aligned}$$

3. Assume $z \leq x \leq y$, which implies that $\overline{\text{MST}}(G_2) = z + x$. (see Figure 18(d)) This is analogous to case 2; the proof parallels the one for case 2 above.

Clearly the bound $\overline{\text{MST}}(\widehat{G}(\vec{d})) \leq \frac{3}{2} \cdot [d_1 \cdot \overline{\text{MST}}(G_1) + d_2 \cdot \overline{\text{MST}}(G_2)]$ holds in each one of the three possible cases 1, 2 and 3. □

Although Theorem 6.5 above shows that the upper bound has a constant factor of $\frac{3}{2}$ for 3-pin nets, we have been unable to find an example of a 3-pin net with metric weights where the cost of the tradeoff MST exceeds the lower bound by more than a factor of $\frac{4}{3}$. Similarly, for 4-pin nets the general upper bound of Theorem 6.3 implies that this factor is $n - 1 = 3$, but despite an extensive computer-aided search, we have been unable to find an example of a 4-pin net with metric weights where the cost of the tradeoff MST exceeds the lower bound by more than a factor of $\frac{3}{2}$. Therefore, we conjecture that our proven bounds can be made considerably tighter, and leave this as an open problem.

These theoretical results have direct implications for the performance of our router, since we can bound the cost of the solutions that our router will produce in terms of the optimal routing solutions over the individual edge weights. In particular, when optimizing multiple metric objectives functions in the routing of 3-pin nets, our router will produce solutions that are guaranteed to be no worse than 50% more than optimal, etc.

7 Experimental Results

We have implemented a complete FPGA router, based on the GB1S algorithm, using C++ in the SUN IPC workstation environment. The code is available from the authors upon request. We have benchmarked our router on several industrial circuits, including the ones used in [5]. A common criteria

used to evaluate the quality of FPGA routing solutions is the maximum width of the channels (i.e., how many edges wide are they) required to successfully route all nets of a circuit [5]; this measure captures the amount of FPGA routing resources needed to route a design. We have therefore compared the maximum channel width our router required to that required by CGE [5]

Following the example of [5], we use a "connection flexibility" of 0.6 times the number of edges per channel (rounding this number to the nearest integer), and a switchbox flexibility of 6; these parameters are typical for current symmetrical-array FPGAs [42]. The switchbox interconnection options and connection-edge to logic block connections which we use are identical to those used by CGE [5]. Note that it is crucial to use identical FPGA interconnection specifications when comparing FPGA routers, since switchbox and connection flexibility dramatically affects routability [5].

We have tested out router on the five industrial benchmarks used in [5] (see Figure 19). Table 20 gives the comparison results of both CGE and our router for these benchmarks. Note that in three of the five cases, our router is able to successfully route all nets using *fewer* edges per channel. This indicates that our router is more thrifty in its utilization of the available resources than other routers, and can produce feasible routings where others cannot. FPGAs are available in several standard sizes, each with a fixed number of edges per channel [42]; clearly being able to successfully route system designs using fewer resources will allow smaller (and cheaper) standard-sized parts to be used, or may allow a system designs to be implemented using fewer fixed-size FPGAs when the design cannot fit on a single FPGA chip. Figure 21 shows the solution our router produced for the BUSC circuit with 8 edges per channel, switchbox flexibility of 6, and connection flexibility of 5.

We have noted that our GB1S algorithm produces optimal routings for three-pin nets; Figure 22 shows a simple example of how our router finds the optimal solution for a three-pin net where others do not. Figure 22(a) shows the routing produced by CGE [5] which uses nearly twice the number of resources as the optimal routing produced by our GB1S router, shown in (b).

8 Conclusion

We proposed a unified general framework for FPGA routing, allowing the simultaneous optimization of multiple competing objectives under a smooth designer-controlled tradeoff. Our approach is based on a new and general multi-weighted graph formulation, which enables a good theoretical characterization

Breakdown of nets by number of pins							
Circuit	#nets	#2-pin	#3-pin	#4-10	#11-50	#over 50	FPGA size
BUSC	151	98	17	28	8	0	12 × 13
DMA	213	105	34	52	22	0	16 × 18
BNRE	352	169	86	70	25	2	21 × 22
DFSM	420	354	7	26	28	5	22 × 23
Z03	608	321	77	176	29	5	26 × 27

Figure 19: This table shows a breakdown of the nets by size (#2-pin nets, etc.). We observe that he majority of nets contain 3 pins or less, for which the GB1S router produces the optimal solution. Also shown are the total number of nets and total number of pins (summed over all nets) for each circuit, as well as the size of the FPGA used to route each circuit (the number of rows and columns of FPGA logic blocks).

Maximum channel width required to route all nets		
Circuit	CGE	GB1S
BUSC	10	**8**
DMA	10	**9**
BNRE	12	**11**
DFSM	10	**11**
Z03	13	**13**

Figure 20: This table shows the maximum channel width required by our GB1S router to successfully route all nets in each of five industry benchmark circuits; given the maximum channel widths shown, *all* nets in each benchmark have been successfully routed by GB1S. For comparison we also give the analogous maximum channel widths required by the CGE detailed router.

of our method's performance. Moreover, our approach is architecture-independent, computationally efficient, and parallelizable. Finally, our general multi-weighted graph approach may be directly applied to other areas of CAD, as well as to many other classic combinatorial optimization problems (e.g., traveling salesman, matching, partitioning, spanning and Steiner trees, etc.)

Remaining open problems include:

- Improved exact upper bounds for multi-weighted metric graphs over 3- and 4-pin nets;
- Improved asymptotic upper bounds for multi-weighted metric graphs; and
- Application of multi-weighted graphs to other optimization problems.
- Find computationally effective methods of combining the FPGA placement phase with the routing

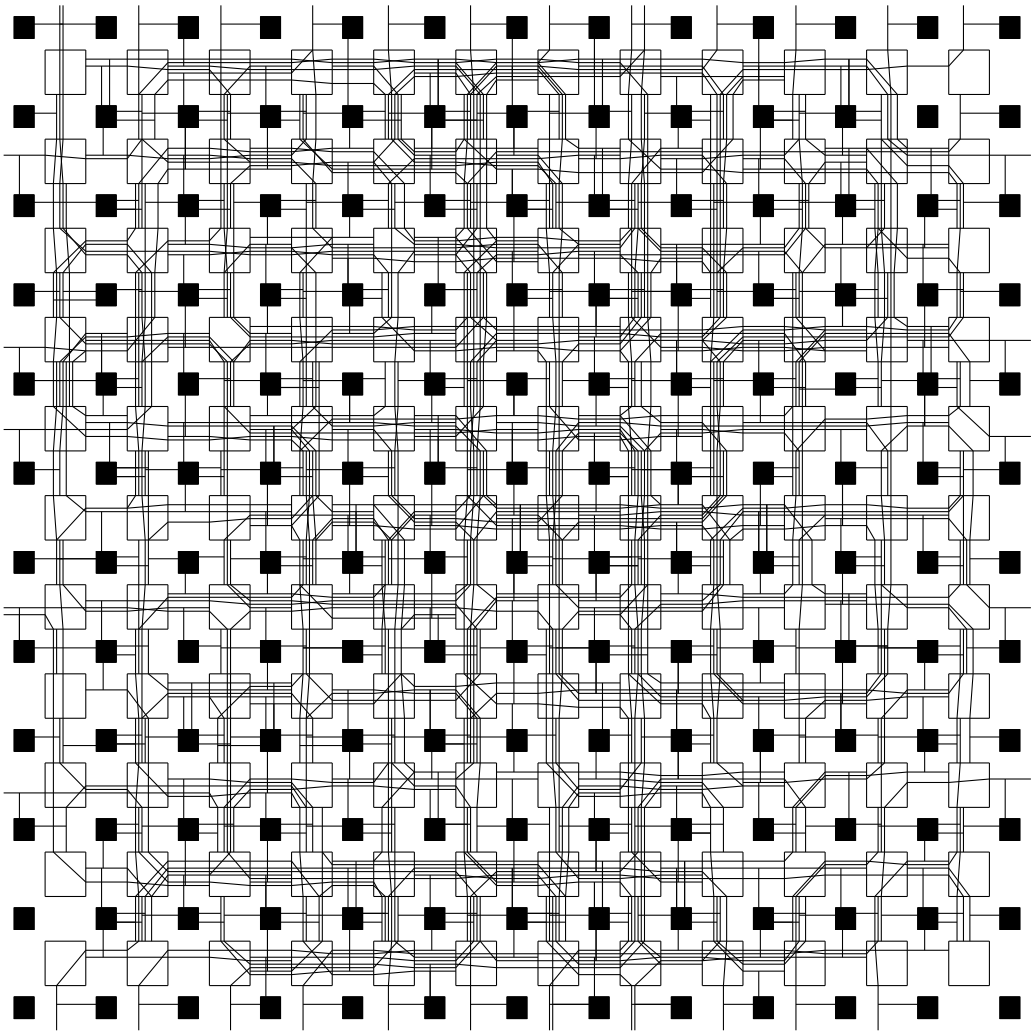
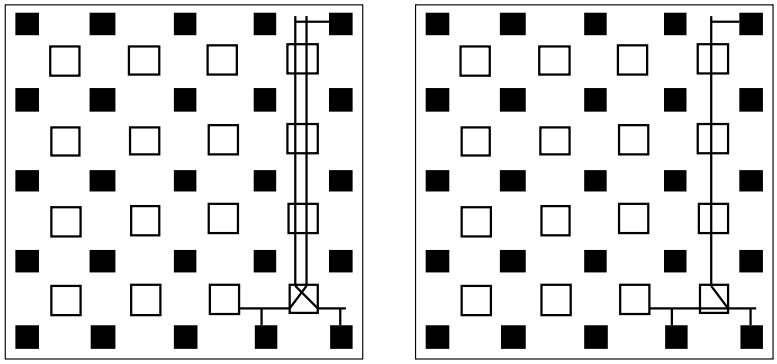


Figure 21: Solution produced by the GB1S router for the BUSC circuit with 8 edges per channel, switchbox flexibility of 6, and connection flexibility of 5.

phase into a unified single place/route step.

9 Acknowledgments

The authors would like to thank Jonathan Rose and Stephen Brown for their help and patience in answering questions on many occasions.



(a) Routing which is twice as bad as optimal. (b) Optimal routing.

Figure 22: Example of a "naive" routing for three-pin "T" net which is twice as bad as optimal (a), and the optimal routing produced by GB1S (b).

References

[1] T. Barrera, J. Griffith, S. A. McKee, G. Robins, and T. Zhang, *Toward a Steiner Engine: Enhanced Serial and Parallel Implementations of the Iterated 1-Steiner Algorithm*, in Proc. Great Lakes Symp. VLSI, Kalamazoo, MI, March 1993, pp. 90–94.

[2] T. Barrera, J. Griffith, G. Robins, and T. Zhang, *Narrowing the Gap: Near-Optimal Steiner Trees in Polynomial Time*, in Proc. IEEE Intl. ASIC Conf., Rochester, NY, September 1993, pp. 87–90.

[3] N. B. Bhat and D. D. Hill, *Routable Technology Mapping for LUT FPGAs*, in Proc. IEEE Intl. Conf. Computer-Aided Design, 1992, pp. 95–98.

[4] S. Brown, J. Rose, and Z. G. Vranesic, *A Detailed Router for Field-Programmable Gate Arrays*, IEEE Trans. Computer-Aided Design, 11 (1992), pp. 620–628.

[5] S. D. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic, *Field-Programmable Gate Arrays*, Kluwer Academic Publishers, Boston, 1992.

[6] P. K. Chan, M. D. F. Schlag, and J. Y. Zien, *On Routability Prediction for Field-Programmable Gate Arrays*, in Proc. ACM/IEEE Design Automation Conf., 1993, pp. 326–330.

[7] K. C. Chen, J. Cong, Y. Ding, A. B. Kahng, and P. Trajmar, *DAG-Map: Graph-Based FPGA Technology Mapping for Delay Optimization*, IEEE Design & Test of Computers, 9 (1992), pp. 7–20.

[8] J. P. Cohoon and D. S. Richards, *Optimal Two-Terminal alpha-beta Wire Routing*, INTEGRATION: the VLSI Journal, 6 (1988), pp. 35–57.

[9] E. W. Dijkstra, *A Note on Two Problems in Connection With Graphs*, Numerische Mathematik, 1 (1959), pp. 269–271.

[10] T. Gao, K. C. Chen, J. Cong, Y. Ding, and C. L. Liu, *Placement and Placement Driven Technology Mapping for FPGA Synthesis*, in Proc. IEEE Intl. ASIC Conf., Rochester, NY, September 1993, pp. 87–91.

[11] M. Garey and D. S. Johnson, *The Rectilinear Steiner Problem is NP-Complete*, SIAM J. Applied Math., 32 (1977), pp. 826–834.

[12] E. N. Gilbert and H. O. Pollak, *Steiner Minimal Trees*, SIAM J. Applied Math., 16 (1968), pp. 1–29.

[13] M. Hanan, *On Steiner's Problem With Rectilinear Distance*, SIAM J. Applied Math., 14 (1966), pp. 255–265.

[14] N. Hasan, G. Vijayan, and C. K. Wong, *A Neighborhood Improvement Algorithm for Rectilinear Steiner Trees*, in Proc. IEEE Intl. Symp. Circuits and Systems, New Orleans, LA, 1990.

[15] J.-M. Ho, G. Vijayan, and C. K. Wong, *New Algorithms for the Rectilinear Steiner Tree Problem*, IEEE Trans. Computer-Aided Design, 9 (1990), pp. 185–193.

[16] T. C. Hu and T. Shing, *The α-β Routing*, in VLSI Circuit Layout: Theory and Design, New York, 1985, IEEE Press, pp. 139–143.

[17] F. K. Hwang, *On Steiner Minimal Trees with Rectilinear Distance*, SIAM J. Applied Math., 30 (1976), pp. 104–114.

[18] ——, *An O(n log n) Algorithm for Rectilinear Minimal Spanning Trees*, J. ACM, 26 (1979), pp. 177–182.

[19] F. K. Hwang, D. S. Richards, and P. Winter, *The Steiner Tree Problem*, North-Holland, 1992.

[20] A. B. Kahng and G. Robins, *A New Family of Steiner Tree Heuristics With Good Performance: The Iterated 1-Steiner Approach*, in Proc. IEEE Intl. Conf. Computer-Aided Design, Santa Clara, CA, November 1990, pp. 428–431.

[21] ——, *A New Class of Iterative Steiner Tree Heuristics With Good Performance*, IEEE Trans. Computer-Aided Design, 11 (1992), pp. 893–902.

[22] ——, *On Performance Bounds for a Class of Rectilinear Steiner Tree Heuristics in Arbitrary Dimension*, IEEE Trans. Computer-Aided Design, 11 (1992), pp. 1462–1465.

[23] K. Karplus, *Xmap: a Technology Mapper for Table-lookup Field-Programmable Gate Arrays*, in Proc. ACM/IEEE Design Automation Conf., 1991, pp. 240–243.

[24] L. Kou, G. Markowsky, and L. Berman, *A Fast Algorithm for Steiner Trees*, Acta Informatica, 15 (1981), pp. 141–145.

[25] J. W. Lavinus and J. P. Cohoon, *Routing a Multi-Terminal Critical Net: Steiner Tree Construction in the Presence of Obstacles*, Tech. Rep. CS-93-19, Department of Computer Science, University of Virginia, April 1993.

[26] J. H. Lee, N. K. Bose, and F. K. Hwang, *Use of Steiner's Problem in Sub-Optimal Routing in Rectilinear Metric*, IEEE Trans. Circuits and Systems, 23 (1976), pp. 470–476.

[27] K. W. Lee and C. Sechen, *A New Global Router for Row-Based Layout*, in Proc. IEEE Intl. Conf. Computer-Aided Design, Santa Clara, CA, November 1990, pp. 180–183.

[28] G. G. Lemieux and S. D. Brown, *A Detailed Routing Algorithm for Allocating Wire Segments in Field-Programmable Gate Arrays*, in Proc. ACM/SIGDA Physical Design Workshop, Lake Arrowhead, CA, April 1993.

[29] F. D. Lewis and W. C. Pong, *A Negative Reinforcement Method of PGA Routing*, in Proc. ACM/IEEE Design Automation Conf., 1993, pp. 601–605.

[30] A. Prim, *Shortest Connecting Networks and Some Generalizations*, Bell Syst. Tech J., 36 (1957), pp. 1389–1401.

[31] D. Richards, *Fast Heuristic Algorithms for Rectilinear Steiner Trees*, Algorithmica, 4 (1989), pp. 191–207.

[32] G. Robins, *On Optimal Interconnections*, Ph.D. Dissertation, CSD-TR-920024, Department of Computer Science, UCLA, 1992.

[33] J. Rose, *Parallel Global Routing for Standard Cells*, IEEE Trans. Computer-Aided Design, 9 (1990), pp. 1085–1095.

[34] K. Roy, B. Guan, and C. Sechen, *FPGA MCM Partitioning and Placement*, in Proc. ACM/SIGDA Physical Design Workshop, Lake Arrowhead, CA, April 1993, pp. 211–212.

[35] M. Schlag, J. Kong, and P. K. Chan, *Routability-Driven Technology Mapping for LookUp Table-Based FPGAs*, in Proc. IEEE Intl. Conf. Computer-Aided Design, 1992, pp. 86–90.

[36] T. L. Snyder, *On the Exact Location of Steiner Points in General Dimension*, SIAM J. Comput., 21 (1992), pp. 163–180.

[37] S. Trimberger, *Field-Programmable Gate Arrays*, IEEE Design & Test of Computers, 9 (1992), pp. 3–5.

[38] S. Trimberger and M. R. Chene, *Placement-Based Partitioning for Lookup-Table-Based FPGAs*, in Proc. IEEE Intl. Conf. Computer-Aided Design, 1992, pp. 91–94.

[39] B. Tseng, J. Rose, and S. Brown, *Improving FPGA Routing Architectures Using Architecture and CAD Interactions*, in Proc. IEEE Intl. Conf. Computer-Aided Design, 1992, pp. 99–104.

[40] P. Winter, *Steiner Problem in Networks: A Survey*, Networks, 17 (1987), pp. 129–167.

[41] Y. F. Wu, P. Widmayer, and C. K. Wong, *A Faster Approximation Algorithm for the Steiner Problem in Graphs*, Acta Informatica, 23 (1986), pp. 223–229.

[42] Xilinx, *The Programmable Gate Array Data Book*, Xilinx, Inc., San Jose, California, 1991.

[43] A. Z. Zelikovsky, *The 11/6 Approximation Algorithm for the Steiner Problem on Networks*, Information and Computation (to appear), (1992).