3

Restriction Methods

3.1 Introduction

Restriction is one of the most basic techniques to design approximation algorithms. The idea is to generate a solution to a given problem \( P \) by providing an optimal or suboptimal solution to a subproblem of \( P \). A subproblem of a problem \( P \) means restricting the solution space for \( P \) by disallowing a subset of the feasible solutions. Restriction may be structural or algorithmic. In the case of structural restriction, the idea is to impose some structure on the set of feasible solutions (the solution space), which can be exploited by an efficient algorithm that solves the problem optimally or suboptimally. In the case of algorithmic restriction, an algorithm restricts the set of feasible solutions that may be found. The problem in the restricted solution spaces is referred as the subproblem. For this approach to be effective the subproblem must have the property that for every problem instance, its optimal or suboptimal solution should have an objective function value that is “close” to the optimal one for the original problem. The most common approach is to solve the subproblem once but there are algorithms where more than one subproblem is solved, and then the best of the solutions computed is the solution generated. In this chapter we discuss this methodology and show how to apply it to several problems. Approximation algorithms based on this approach are discussed in Volume 1, Chapters 31, 32, 36, and Volume 2, Chapters 6 and 15.

This approach is in a sense the opposite of “relaxation,” that is, augmenting the feasible solution space by including previously infeasible solutions. In this case, one needs to solve a superproblem of \( P \). An approximation algorithm for \( P \) solves the superproblem (optimally or suboptimally) and then transforms such solution to one that is feasible for \( P \). Approximation algorithms based on the linear programming methodology fall under this category. There are many different conversion techniques including rounding, randomized rounding, and so on. Chapters 4, 7, and 11 discuss this approach in detail. Approximation algorithms based on both restriction and relaxation exist (Vol. 2, Chap. 8). These algorithms first restrict the solution space and then relaxes it. The resulting solution space is different from the original one, as there are some infeasible solutions in the original problem that are now a part of the solution space, and some feasible solution in the original problem is no longer part of the solution space.

In this chapter we discuss several approximation algorithms based on restriction. When designing algorithms of this type the first question is: how to select one of the many subproblem(s) to provide a
good approximation for a given problem? One would like to select a subproblem that “works best.” But what do we mean by a subproblem that works best? The one that works best could be a subproblem that results in an approximation algorithm with smallest possible approximation ratio, or it could be a subproblem whose solution can be computed the fastest, or one may use some other criteria, for example, any of the ones discussed in Chapter 1. Perhaps “works best” should be with respect to a combination of different criteria. But even when using the approximation ratio as the only evaluation criteria for an algorithm, it is not at all clear how to select a subproblem that can be solved quickly and from which a best possible solution could be generated. These are the two most important properties when choosing a subproblem to solve. By studying several algorithms based on restriction, one learns why it works for these cases and then it becomes easier to find ways to approximate other problems.

The problems that we discuss in this chapter to illustrate “restriction” are Steiner tree, the traveling salesman, covering points by squares, rectangular partitions, and routing multiterminal nets. The Traveling Salesperson Problem (TSP), and Steiner tree problem are classical problems in combinatorial optimization. The algorithms that we discuss for the TSP are among the best known approximation algorithms for combinatorial problems.

A closely related approach to restriction is transformation-restriction. The idea is to transform the problem instance to a restricted instance of the same problem. The difference is that the restricted problem instance is not a subproblem of original problem instance as in the case of restriction, but it is a “simpler” problem of the same type. In Section 3.5 we present algorithms based on this approach for routing multiterminal nets and embedding hyperedges in a cycle. The fully polynomial time approximation scheme for the knapsack problem based on rounding discussed in Chapter 9 is based on transformation-restriction. In Section 3.8 we summarize the chapter and briefly discuss other algorithms based on restriction for path problems arising in computational geometry.

### 3.2 Steiner Tree

The Steiner tree problem is a classical problem in combinatorial optimization. Let’s define the Steiner tree problem over an edge weighted complete metric graph $G = (V, E, W)$, where $V$ is the set of $n$ vertices, $E$ is the set of $m = \binom{n}{2}$ edges, and $W : E \rightarrow R^+$ is the weight function for the edges. Since the graph is metric the set of weights satisfies the triangle inequality, that is, for every pair of vertices $i, j$, the weight $W(i, j)$ is less than or equal to the sum of the weight of the edges in any path from vertex $i$ to vertex $j$. The Steiner tree problem consists of a metric graph $G = (V, E, W)$ and a subset of vertices $T \subseteq V$. The problem is to find a tree that includes all the vertices in $T$ plus some other vertices in the graph such that the sum of the weight of the edges in the tree is least possible. The Steiner tree problem is an NP-hard problem.

When $T = V$ the problem is called the minimum weight (cost) spanning tree problem. By the 1960s there were several well-known polynomial time algorithms to construct a minimum weight spanning tree for edge weighted graphs [1]. These simple greedy algorithms have low-order polynomial time complexity bounds.

Given an instance of the metric graph Steiner tree problem $(G = (V, E, W), T)$ one may construct a minimum weight spanning tree for the subgraph $G' = (T, E', W')$, where $E'$ and $W'$ includes only the edges joining vertices in $T$. Clearly this minimum weight spanning tree is a restricted version of the Steiner tree problem and it seems a natural way to approximate the Steiner tree problem. This approach was analyzed in 1968 by E. F. Moore (see Reference 2) for the Steiner tree problem defined in metric space (e.g., 2D). The metric graph problem we just defined, includes only a subset of all the possible points in metric space. E. F. Moore presented an elegant proof of the fact that in metric space (and also for metric graphs) $L_M \leq L_T \leq 2L_S$, where $L_M$, $L_T$, and $L_S$ are the weight of a minimum weight spanning tree, a minimum weight tour (solution) for the TSP, and minimum weight Steiner tree for any set of points $P$, respectively. We defined the TSP in Chapter 2. Since every spanning tree for $T$ is a Steiner tree for $T$, the above bounds show that when using a minimum weight spanning tree to approximate the Steiner tree...
Restriction Methods

45

results in a solution whose weight is at most twice the weight of an optimal Steiner tree. In other words, any algorithm that generates a minimum weight spanning tree is a 2-approximation algorithm for the Steiner tree problem. Furthermore, this approximation algorithm takes the same time as an algorithm that constructs a minimum weight spanning trees for edge-weighted graphs [1], since such an algorithm can be used to construct an optimal spanning tree for a set of points in metric space. The above bound is established by defining a transformation from any minimum weight Steiner tree into a TSP tour in such a way that $L_T \leq 2L_S$ [2]. Then by observing that deleting an edge from an optimum tour to the TSP problem results in a spanning tree, shows that $L_M \leq L_T$. The proof is identical to the one given in the next section where we show this result, but starting from a minimum weight spanning tree. So where is restriction in this case? The algorithm finds the best possible solution that does not include Steiner points, that is, points in $V \setminus T$. In other words the restriction is to find the best tree without allowing Steiner points.

Chapter 36 also discusses several constant-ratio approximation algorithms for Steiner trees, including ones where the graph is not metric.

3.3 Traveling Salesperson

The TSP has been studied for many decades [3]. There are many variations of this problem. One of the simplest versions of the problem consists of an edge-weighted complete graph and the problem is to find a minimum weight tour that starts and ends at vertex one and visits every vertex exactly once. The weight of a tour is the sum of the weight of the edges in the tour. Sahni and Gonzalez [4] (Chapter 1) show that the constant-ratio approximation problem is NP-hard, that is, if for any constant $c$ there is a polynomial time algorithm with approximation ratio $c$, then $P = NP$. In this section we discuss approximation algorithms for the TSP defined over complete metric graphs. These algorithms are among the best known approximation algorithms for any problem. The “double minimum weight spanning tree” (DMWST) approximation algorithm that we discuss in this section is widely known, and it is based on the constructive proof for the approximation algorithm discussed in the previous section developed for the Steiner tree problem by E. F. Moore. Additional constant-ratio approximation algorithms for this version of the TSP problem were developed by Rosenkrantz et al. [5]. These algorithms as well the DMWST algorithm have an approximation ratio of $2 - 1/n$ and take $O(n^2)$ time. Since the graph is complete, the time complexity is linear with respect to the number of edges in the graph. After presenting this result we discuss the improved approximation algorithm by Christofides [6]. This algorithm has a smaller approximation ratio, but its time complexity grows faster than that of the previous algorithms.

In the literature you will find that the TSP is also defined with tours visiting each vertex at least once. We now show that the “at most once” version over metric graphs and the “exactly once” version over metric graphs are equivalent problems. Consider any optimal tour $T$ where some vertices are visited more than once. Let vertex $i$ be a vertex visited more than once. Let vertices $j$ and $k$ be visited just before and just after vertex $i$. Delete from the tour the edges $\{j, i\}$ and $\{i, k\}$ and add edge $\{j, k\}$. Because the graph is metric the tour weight will stay the same or decrease. If it decreases, then it contradicts the optimality of $R$. So the weight of the tour must be the same as before. After applying this transformation until it is no longer possible we obtain a tour $R'$ in which every vertex is visited exactly once and the weight of $R'$ is identical to that of $R$. Since every tour that visits every vertex exactly once also visits every vertex at least once, it follows that both versions of the problem for metric graphs have the same optimal tour weight, that is, both problems are equivalent. Since for the TSP defined over metric graphs both versions of the problem are equivalent, for convenience we use the definition of tours to visit each vertex at least once.

Now suppose that you have an optimal tour $S$ for an instance $I$ of the “at least once” version of the TSP. Applying the above-mentioned transformation we obtain an optimal tour $S'$ in which every vertex is visited exactly once. Deleting an edge from the tour results in a spanning tree. Therefore, the weight of a minimum weight spanning tree is a lower bound for the weight of an optimal tour. The questions are: How good of a lower bound is it? How can one construct a tour from a spanning tree?
How can we find a tour from a spanning tree $T$? Just draw the spanning tree in the plane with a vertex as its root and construct a tour by visiting each edge in the tree $T$ twice as illustrated in Figure 3.1. A more formal approach is to construct an Euler circuit in the multigraph (graph with multiple edges between vertices) consisting of two copies of the edges in $T$. An Euler tour (or circuit) is a path that starts and ends at the same vertex and visits every edge in the multigraph once. An Euler tour always exists for the multigraphs we have defined because these multigraphs are connected and all their nodes are of even degree (the number of edges incident to each vertex is even). These multigraphs are called Eulerian, and an Euler tour can be constructed in linear time with respect to the number of nodes and edges in the multigraph [7].

The approximation algorithm, which we refer to as DMCST, constructs a minimum cost spanning tree, makes a copy of all the edges in the tree, and then generates a tour from this tree with weight at most equal to twice the cost of a minimum cost spanning tree. We established before that an optimal tour has weight greater than the weight of a minimum cost spanning tree, it then follows that the weight of the tour that the DMCST algorithm generates is at most twice the weight of an optimal tour for $G$. Therefore, algorithm DMCST generates 2-approximate solution. Actually the ratio is $2 - 1/n$, which can be established when the edge deleted for an optimal tour to obtain a spanning tree is the one with largest weight. The time complexity of the algorithm is bounded by the time complexity for generating a minimum cost spanning tree, since an Euler tour can be constructed in linear time with respect to the number of edges in the spanning tree. We formalize these results in the following theorem:

**Theorem 3.1** For the metric TSP problem algorithm DMCST generates a tour with weight at most $(2 - 1/n)$ times the weight of an optimal tour. The time complexity of the algorithm is $O(n^2)$ time, which is linear time with respect to the number of edges in the graph.

**Proof.** The proof for the approximation ratio follows from the above-mentioned discussion. As Fredman and Tarjan [8] point out, implementing Prim’s minimum cost spanning tree algorithm by using Fibonacci heaps results in a minimum cost spanning tree algorithm that takes $O(n \log n + m)$ time. Since the graph is complete, the time complexity is $O(n^2)$ which is linear with respect to the number of edges in the graph.

This constructive proof is based on Moore’s proof for the approximation algorithm for the Steiner tree problem. The difference is that he starts from an optimal Steiner tree.

So where is the restriction in the above-mentioned algorithms? We are actually restricting tours for the TSP to traverse the least possible number of different edges, though a tour may traverse some of these edges more than once. The minimum number of different edges needed from $G$ is $n - 1$ and they must form a spanning tree. It is therefore advantageous to select a spanning tree of least possible total weight.
That justifies the use of a minimum weight spanning tree. This is another way to think about the design of the DMWST algorithm.

Christofides [6] modified the previous approach so that the tours generated have total weight within 1.5 times the weight of an optimal tour. However, the currently fastest implementation of this procedure takes $O(n^3)$ time. His modification is very simple. First, observe that there are many different ways to transform a spanning tree into an Eulerian multigraph. All possible augmentations must include at least one edge incident to every odd degree vertex in the spanning tree. Let $N$ be the set of odd-degree vertices in the spanning tree. Christofides’ idea is to transform the spanning tree into an Eulerian multigraph by adding the least number of edges with the least possible total weight. He showed that such set of edges is a minimum cost complete matching on the graph $G_N$ induced by the set of vertices $N$ in $G$. A matching is a subset of the edges in a multigraph, no two of which are incident upon the same vertex. A matching is complete if every node has an edge in the matching incident to it, and the weight of a matching is the sum of the weight of the edges in it. A minimum cost complete matching can be constructed in polynomial time. The edges in the complete matching plus the ones in the spanning tree form an Eulerian multigraph, and Christofides’ algorithm generates as its solution an Euler tour of this multigraph.

To establish the 1.5 approximation ratio we observe that an optimal tour can be transformed without increasing its total weight into another tour that visits only the vertices in $N$ because the graph is metric. One can partition the edges in this reduced tour into two sets such that each set is a complete matching for the restricted graph $N$. One set contains the even numbered edges in the tour and the other set the odd numbered edges. Since a minimum cost complete matching for $G_N$ has total cost at most equal to either of the previous two matchings, it then follows that the minimum cost complete matching has total cost at most half of the weight of an optimal tour. Therefore, the edges in the the tour constructed by Christofides’ algorithm have weight at most 1.5 times the weight of an optimal tour. The time complexity for Christofides’ algorithm is $O(n^3)$ and it is dominated by the time required to construct a minimum cost complete matching [9,10]. We formalize this result in the following theorem whose proof is omitted as it follows the above discussion.

**Theorem 3.2 [6]** For the metric TSP, Christofides’ algorithm generates a tour with weight at most 1.5 times the weight of an optimal tour. The time complexity of the algorithm is $O(n^3)$.

This approach used by this algorithm is similar to the one employed by Edmonds and Johnson [11] for the Chinese postman problem. Given an edge weighted connected undirected graph, the Chinese postman problem is to construct a minimum weight cycle, which contains every edge in the graph possibly with repeated edges. The currently best algorithm to solve this problem takes $O(n^3)$ time, and it uses shortest paths and minimum weighted matching algorithms. There are asymptotically faster algorithms when graphs are sparse and weight of the edges are integers.

### 3.4 Covering Points by Squares

Given a set of $n$ points, $P = \{(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)\}$, in two-dimensional space and an integer $D$, the covering by squares in 2D ($CS_2$) problem is to find the least number of $D \times D$ squares to cover $P$. The $CS_2$ problem as well as the problem of covering by disks have been shown to be NP-hard [12]. Approximation algorithm for these problems as well as their generalizations to multidimensional space have been developed [13,14]. All these problems find applications in several areas [12,15,16]. The most popular application is to find the least number of emergency facilities such that every potential patient lives at a distance at most $D$ from at most one facility. This application corresponds to covering by the least number of disks with radius $D$.

We discuss in this section a simple approximation algorithm based on restriction for the $CS_2$ problem. Assume without loss of generality that $x_i \geq 0$ and $y_i \geq 0$ and that at least one of the points has $x$-coordinate value of zero. Define the function $I_x(P_i) = \lfloor x_i/D \rfloor$, used to identify vertical bands of points. For $l \geq 0$ and $k$ consists of all the points with $I_x(P_i) = k$. 

Restriction Methods
The restriction to the solution space is to only allow feasible solutions where each square covers points from only one band. Note that an optimal solution to the $CS_2$ problem does not necessarily satisfy this property. For example, the instance with $P_1 = (0.1, 1.0)$, $P_2 = (0.1, 2.0)$, $P_3 = (1.1, 0.9)$, $P_4 = (1.1, 2.1)$, and $D = 1$ has two squares in optimal cover. The first square covers points $P_1$ and $P_3$, and the second square covers $P_2$ and $P_4$. However, an optimal cover for the points in band 0 (i.e., $P_1$ and $P_2$) is 1 square and the one for the points in band 1 (i.e., $P_3$ and $P_4$) is 2 squares. So an optimal cover to the restricted problem has three squares, but an optimal cover for the $CS_2$ problem has two squares.

One reason for restricting the solution space in this way is that an optimal cover for any given band can be easily generated by a greedy procedure in $O(n \log n)$ time [14]. A greedy approach places a square as high as possible provided it includes the bottommost point in the band as well as all other points in the band at a vertical distance at most 1 from a bottommost point. All the points covered by this square are removed and the procedure is repeated until all the points have been covered. One can easily show that this is an optimal cover by transforming any optimal solution for the band, without increasing the number of squares, to the cover generated by the greedy algorithm. By using elaborate data structures Gonzalez [14] showed that the greedy algorithm can be implemented to take $O(n \log s)$ time, where $s$ is the number of squares in an optimal solution. Actually a method that uses considerable more space can be used to solve the problem in $O(n)$ time [14].

The solution generated by our algorithm for the whole problem is the union of the covers for each of the bands generated by the greedy method. Let $f = E + O$ be the total number of squares, where $E(O)$ is the number of square for the even (odd) numbered bands. We claim that an optimal solution to the $CS_2$ problem has at least $\max(E, O)$ squares. This follows from the fact that an optimal solution for the even (odd) numbered bands is $E(O)$ because it is not possible for a square to cover points from two different even (odd) numbered bands. Therefore, $\frac{f^*}{f} \leq 2$, where $f^*$ is the number of squares in an optimal solution for problem instance $I$. This result is formalized in the following theorem whose proof is omitted as it from the previous discussion.

**Theorem 3.3** For the $CS_2$ problem the above-mentioned procedure generates a cover such that $\frac{f^*}{f} \leq 2$ in $O(n \log s)$ time, where $s$ is the number of squares in an optimal solution.

A polynomial time approximation scheme for the generalization of the $CS_2$ to $d$ dimensions (the $CS_d$ problem) is discussed in Volume 1, Chapter 8 and Volume 2, Chapter 5. The idea is to generate a set of solutions by shifting the bands by different amounts and then selecting as the solution the best cover computed by the algorithm. This approach is called shifting and it was introduced by Hochbaum and Maass [13] and Baker [17].

### 3.5 Optimal Rectangular Partitions

The minimum edge-length rectangular partition ($RG_P$) problem has applications in the area of computer aided-design of integrated circuits and systems. Given a rectangle $R$ with interior points $P$, the $RG_P$ problem is to introduce a set of interior lines segments with least total length such that every point in $P$ is in at least one of the partitioning line segments, and $R$ is partitioned into rectangles. Figure 3.2a shows a problem instance $I$ and Figure 3.2b shows an optimal rectangular partition for the problem instance $I$.

A rectangular partition $E$ is said to have a guillotine cut if one of the vertical or horizontal line segments partitions the rectangle into two rectangles. A rectangular partition $E$ is said to be a guillotine partition if either $E$ is empty, or $E$ has a guillotine cut and each of the two resulting rectangular partitions is a guillotine partition.

Finding an optimal rectangular partition is an NP-hard problem [18]. However, an optimal guillotine partition can be constructed in polynomial time. Therefore, it is natural to restrict the solution space to guillotine partitions when approximating rectangular partitions.
In Volume 2, Chapter 6 we prove that an optimal guillotine partition has total edge length, that is at most twice the length of an optimal rectangular partition. Gonzalez and Zheng [19] presented a complex proof that shows that bound is just 1.75. In Volume 2, Chapter 6 we also explain the basic ideas behind the proof of the approximation ratio of 1.75. This approach has been extended to the multidimensional version of this problem by Gonzalez et al. [20].

An optimal guillotine partition can be constructed in $O(n^5)$ time via dynamic programming. When $n$ is large this approach is not practical. Gonzalez et al. [21] showed that suboptimal guillotine partitions that can be constructed in $O(n \log n)$ time generate solutions with total edge length at most four times the length of an optimal rectangular partition. As in the case of optimal guillotine partitions, this result has been extended to the multidimensional version of the problem [21]. Clearly, none of the methods dominates the other when considering both the approximation ratio and the time complexity bound.

Chapter 36 discusses how more general guillotine cuts can be used to develop a PTAS for the TSP in two-dimensional space. Volume 2, Chapter 2 discusses this approach for the TSP and Steiner tree problems.

### 3.6 Routing Multiterminal Nets

Let $R$ be a rectangle whose sides lie on the two-dimensional integer grid. A subset of grid points on the boundary of $R$ that do not include the corners of $R$ is denoted by $S$, and its grid points are called terminal points. Let $n$ be the number of terminal points, that is, the cardinality of set $S$, and let $N_1, N_2, \ldots, N_m$ be a partition of $S$ such that each set $N_i$ includes at least two terminal points. Each set $N_i$ is called a net and the problem is to make all the terminal points electrically common by introducing a set wire segments in two layers. Terminal points from different nets should not be made electrically common. The wire segment must be along the grid lines outside $R$ with at most one wire segment assigned to each grid edge. All vertical wires are in one layer and all the horizontal wires are in another layer. A via (or hole) is used to connect a vertical wire to a horizontal wire that have a common point. Note that each grid point may only belong to one net. In other words, when viewing the two layers from the top, dog-leg on one layer (wires from two nets bending at a grid point) are not allowed. The main reasons are that dog-legs would complicate the layer assignment without improving the layout area.

The Multiterminal net routing Around a Rectangle (MAR) problem is given a rectangle $R$ and a set of nets, find a layout, subject to the constraints defined earlier, that fits inside a rectangle with least possible area. Constructing a layout in this case reduces to just finding the wire segments for each net along the grid lines (without dog-legs) outside $R$, since the layer assignment is straight forward.

Developing a constant-ratio approximation algorithm for this problem is complex because the objective function depends on the product of two values, rather than just one value as in most other problems. Gonzalez and Lee [22] developed a linear time algorithm for the MAR problem when every net consists of two terminal points. It is conjectured that the problem becomes NP-hard when the nets have three terminal points each. Gonzalez and Lee [22,23] developed constant-ratio approximation algorithms for the MAR problem [23,24]. The approximation ratios for these algorithms are 1.69 [23] and 1.6 [24].
The approach is to partition the set of nets into groups and then route each group of nets independently of each other. Some of the groups are routed optimally. Since the analysis of the approximation ratio for these algorithms is complex, in this section we only analyze the case when the nets contain one terminal point on the top side of $R$, and one or more terminal points on the bottom side of $R$. The set of these nets is called $N_{TB}$. The algorithm to route the $N_{TB}$ nets is based on restriction and it is interesting. Readers interested in additional details are referred to papers [23,24].

Let $n_{TB}$ be the number of $N_{TB}$ nets. Let $E$ be an optimal area layout for all the nets and let $D$ be $E$ except that the set of nets in $N_{TB}$ are all connected by a path that crosses the left side of $R$. In this case the layout for the nets $N_{TB}$ is restricted (only paths that cross the left side of $R$ are allowed). We use $H_E(TB)$ ($H_D(TB)$) to denote the height of the layout $E(D)$ on the top plus the corresponding height on the bottom side of $R$.

To simplify the analysis, lets assume that every net in $N_{TB}$ is connected in $E$ by a path that either crosses the left or right (but not both) sides of $R$. Gonzalez and Lee [24] explain how to modify the analysis when some of these nets are connected by paths that cross both the left and right sides of $R$.

By reversing the connecting path for a net in $N_{TB}$ we mean to connect the net by a path that crosses the opposite side of $R$, that is, if it crossed the left side of $R$ it will now cross the right side, or vice versa. When we reverse the connecting path for a net the height on the top side plus the bottom side of $R$ increases by at most two. We say that connecting paths for two $N_{TB}$ nets cross on the top side of $R$ when their contribution to the total height of the assignment is two for at least one point in between the two terminal points. When we interchange the connecting paths for two $N_{TB}$ nets that cross on the top side of $R$ we mean reversing both connecting paths. An interchange increases by at most two the height on the top side plus the bottom side of $R$.

We transform $E$ to $D$ by reversals in order to quantify the difference in heights between $E$ and $D$. The largest increase in height is when all the $N_{TB}$ nets are connected in $E$ by paths that cross the right side of $R$. In this case we need to reverse all the connecting paths for the $N_{TB}$ nets, so $H_D(TB) \leq H_E(TB) + 2n_{TB}$. When one plugs this in the analysis for the whole problem it results in an algorithm with an approximation ratio greater than 2.

A better approach is to use the following restriction: All the connecting paths for the $N_{TB}$ nets are identical, and either they cross the left or the right side of $R$. In this case we construct two different layouts. Let layout $D_1$ ($D_r$) be $E$ except that all the nets in $N_{TB}$ are connected by a path crossing the left (right) side of $R$. Let $M$ be a minimum area layout between $D_1$ and $D_r$. In $E$ let $l$ ($r$) be the number of $N_{TB}$ nets connected by a path crossing the left (right) side of $R$. By reversing the minimum of $\{l, r\}$ paths it is possible to transform $E$ to $D_1$ or $D_r$. Therefore, $H_M(TB) \leq H_E(TB) + n_{TB}$, which is better by 50% than for the assignment $D$ defined before.

By trying more alternatives one can obtain better solutions. Let us partition the set of nets $N_{TB}$ into two groups, $S_l$ and $S_r$. The set $S_l$ contains the $\frac{n_{TB}}{2}$ nets in $N_{TB}$ whose terminal point on the top side of $R$ is closest to the left side of $R$, and set $S_r$ contains the remaining ones. For $i, j \in \{l, r\}$ let $D_{ij}$ be $E$ except that all the nets in $S_i$ are connected by paths that cross the “$i$” side of $R$ and all the nets in $S_r$ are connected by paths that cross the “$j$” side of $R$. Let $P$ be a minimum area layout among $D_{ll}, D_{lr}, D_{rl}, D_{rr}$. Let $l_1(l_2)$ be the number of nets in $S_l$ connected by a path that crosses the left side of $R$. We define $l_2$ and $r_2$ similarly, but using the set $S_r$. We establish in the following lemma that $H_P(TB) \leq H_E(TB) + \frac{3}{4}n_{TB}$.

**Lemma 3.1** Let $P$ and $E$ be the assignments defined previously. Then $H_P(TB) \leq H_E(TB) + \frac{3}{4}n_{TB}$.

**Proof.** The proof is by contradiction. Suppose that $H_P(TB) > H_E(TB) + \frac{3}{4}n_{TB}$. There are two cases depending on the values of $r_1$ and $l_2$.

**Case 1:** $r_1 < l_2$.

To transform assignment $E$ to $D_{lr}$ we need to interchange $l_2$ connecting paths that cross on the top side of $R$ and reverse $r_1 - l_2$ connecting paths. Therefore, $H_{D_{lr}}(TB) \leq H_E(TB) + 2r_1$. Since $H_{D_{lr}}(TB) \geq H_P(TB) > H_E(TB) + \frac{3}{4}n_{TB}$, we know that $2r_1 > \frac{3}{8}n_{TB}$ which is equivalent to $r_1 > \frac{3}{8}n_{TB}$. Since $r_1 + l_1 = \frac{1}{2}n_{TB}$, we know that $l_1 < \frac{5}{8}n_{TB}$.

Downloaded By: 10.3.98.93 At: 08:24 09 Nov 2019; For: 9781351236423, chapter3, 10.1201/9781351236423-3
Restriction Methods

To transform assignment $E$ to $D_{rr}$ we need to reverse the connecting paths of $l_1 + l_2$ connecting paths. Therefore, $H_{D_{rr}}(TB) \leq H_E(TB) + 2l_1 + 2l_2$. Since $H_{D_{rr}}(TB) \geq H_P(TB) > H_E(TB) + \frac{3}{8}n_{TB}$, we know that $l_1 + l_2 > \frac{3}{8}n_{TB}$. Applying the same argument to assignment $D_{ll}$, we know $l_1 + r_2 > \frac{3}{8}n_{TB}$. Adding these two last inequalities and substituting the fact that $l_2 + r_2 = \frac{1}{2}n_{TB}$, we know that $l_1 > \frac{1}{8}n_{TB}$. Which contradicts our previous finding that $l_1 < \frac{1}{8}n_{TB}$.

Case 2: $r_1 \geq l_2$.

A contradiction in this case can be obtained applying similar arguments to assignments $D_{ll}$ and $D_{gr}$.

For three groups, rather than two, Gonzalez and Lee [23] showed that $H_P(TB) \leq H_E(TB) + \frac{3}{8}n_{TB}$, where $P$ is the best of the eight assignments. This is enough to prove the approximation ratio of 1.69 for the MAR problem. If instead of three groups one uses six, one can prove $H_P(TB) \leq H_E(TB) + 0.6n_{TB}$, where $P$ is the best of the 64 assignments generated. In this case, approximation ratio for the whole MAR problem is 1.6. Interestingly, partitioning into more groups results in smaller bounds for this group, but does not reduce the approximation ratio for the MAR problem because the routing of other nets becomes the bottleneck. We state Gonzalez and Lee’s theorem without a proof. Readers interested in the proof are referred to Reference 24.

**Theorem 3.4** For the MAR problem the procedure given in Reference 24 generates a layout with area at most 1.6 times the area of an optimal layout in $O(nm)$ time.

An interesting observation is that the proof that the bound $H_P(TB) \leq H_E(TB) + (1.6)n_{TB}$ holds can be carried out automatically by solving a set of linear programming problems. The linear programming problems find the ratios for $l_i$ and $r_i$ such that the minimum increase from $E$ to one of the layouts is maximized. Note that some of the “natural” constraints of the problem are in terms $\max\{r_1, l_2\}$ which makes the solution space nonconvex. However by replacing it with inequalities of the form $r_1 \leq l_2$ and $r_1 > l_2$ we partition the optimization region into several convex regions. By solving a set of linear programming problems (one for each convex region) the maximum possible increase can be computed.

### 3.7 Variations on Restriction

A closely related approach to restriction is to generate a solution by solving a restricted problem instance constructed from the original instance. We call this approach transformation-restriction. For example, consider the routing multiterminal nets around a rectangle discussed in Section 3.6. Remember that there are $n$ terminal points and $m$ nets. Suppose that we break every net $i$ with $k_i$ points into $k_i$ nets with two terminal points each. The $k$ nets consist of adjacent terminal points of the net. In order for these $k_i$ nets to have different terminal points we make a copy of each terminal point at half integer points next to the old ones. Note that a new grid needs to be redefined to include the half integer points without introducing more horizontal (vertical) routing tracks above or below (to the left or right) of $R$. Figure 3.3b shows the details. The resulting two terminal net problem can be solved in linear time using the optimal algorithm developed by Gonzalez and Lee [22]. A solution to this problem can be easily transformed into a solution to the original problem after deleting the added terminal points as well as some superfluous connections. This algorithm generates a layout whose total area is at most 4 times the area of an optimal layout. Furthermore, the layout can be constructed in $O(n)$ time. With respect to the approximation ratio Gonzalez and Lee’s algorithms [23,24] are better, but these algorithms take $O(nm)$ time, whereas the simple algorithm just described takes linear time.

#### 3.7.1 Embedding Hyperedges in a Cycle

In this subsection we present an approximation algorithm for embedding hyperedges in a cycle so as to minimize the congestion (EHCME). This problem has applications in the area of design automation and
parallel computing. As input we are given a hypergraph \( G = (V, H) \), where \( V = \{v_1, v_2, \ldots, v_n\} \) is the set vertices and \( H = \{h_1, h_2, \ldots, h_m\} \) is the set of hyperedges (or subsets with at least two elements of the set \( V \)). Traversing the vertices \( v_1, v_2, \ldots, v_n \) the clockwise direction forms a cycle which we call \( C \). Let \( v_t \) and \( v_s \) be two vertices in \( h_i \) such that \( v_s \) is the next vertex in \( h_i \) in clockwise direction from \( v_t \). Then the pair \( (v_s, v_t) \) for hyperedge \( h_i \) defines the connecting path for \( h_i \) that begins at vertex \( v_s \) then proceeds in the clockwise direction along the cycle until reaching vertex \( v_t \). Every edge \( e \) in the cycle that is visited by the connecting path formed by pair \( (v_s, v_t) \) is said to be covered by the connecting path. The EHCMC problem consists of finding a connecting path \( c_i \) for every hyperedge \( h_i \) such that the maximum congestion of an edge in \( C \) is least possible, where the congestion of an edge \( e \) in cycle \( C \) is the number of connecting paths that include edge \( e \).

Ganley and Cohoon [25] showed that when the maximum congestion is bounded by a fixed constant \( k \), the EHCMC problem is solvable in polynomial time. But, the problem is NP-hard when there is no constant bound for \( k \). Frank et al. [26] showed that when the hypergraph is a graph the EHCMC problem can be solved in polynomial time. We call this problem the embedding edges in a cycle to minimize congestion (EECMC). In this section we present the simple linear time algorithm with an approximation ratio of two for the EHCMC problem developed by Gonzalez [27].

The algorithm based on transformation-restriction for this problem is simple and uses the same approach as in the previous subsection. This general approach also works for other routing problems. A hyperedge with \( k \) vertices \( x_1, x_2, \ldots, x_k \), appearing in that order around the cycle \( C \) is decomposed into the following \( k \) edges \( \{x_1, x_2\}, \{x_2, x_3\} \ldots \{x_{k-1}, x_k\}, \{x_k, x_1\} \). Note that in this case we do not need to add additional vertices as in the previous subsection because a vertex may be part of several hyperedges. The decomposition transforms the problem into an instance of the EECMC problem, which can be solved by the algorithm given in Reference 26. From this embedding we can construct an embedding to the original problem instance after deleting some superfluous edges in the embedding. The resulting embedding can be easily shown to have congestion of at most twice the one in an optimal solution \( X \). This is because there is a solution \( S \) to the EECMC problem instance in which every connecting path \( Y \) in \( X \) can be mapped to a set of connecting paths in \( S \) with the property that if the connecting path \( Y \) contributes one unit to the congestion of an edge \( e \), then the set of connecting paths in \( S \) contribute two units to the congestion of edge \( e \). Furthermore, each connecting path in \( S \) appears in one mapping. The time complexity of the algorithm is \( O(n) \).

3.8 Concluding Remarks

We have seen several approximation algorithms based on restriction. As we have seen the restricted problem may be solved optimally or suboptimally as in Section 3.5. One generates solutions closer to optimal, whereas the other generates the solution faster. These are many more algorithms based on this technique. For example, some computational geometry problems where the objective function is in terms of distance
have been approximated via restriction [28–31]. These type of problems allow feasible solutions to be any set of points. A restricted problem allows a set of points (called artificial points) to be part of the feasible solution space. The more artificial points, the smaller the approximation ratio of the algorithm; however, the algorithm takes longer to generate a solution.

There are problems for which it is not know whether or not there is a constant-ratio approximation algorithm. However, heuristics based on restriction are used to generate good solutions in practice. One such problems is discussed in Volume 2, Chapter 25.

A closely related approach to restriction is transformation-restriction. The idea is to transform the problem instance to a restricted instance of the same problem. The difference is that the restricted problem instance is not a subproblem of original problem instance as in the case of restriction. We showed how this approached is applied to a couple of problems.

Approximation algorithms that are based on restriction and relaxation exist. These algorithms first restrict the solution space and then relaxes it resulting in a solution space that is different from the original one. Gonzalez and Gonzalez [32] have applied this approach successfully to the minimum edge length corridor problem (Vol. 2, Chap. 8).

References


