Rectilinear Steiner Tree Construction using Answer Set Programming

Esra Erdem[1] and Martin D. F. Wong[2]

[1] Institut für Informationssysteme 184/3,
Technische Universität Wien, 1040 Wien, Austria
esra@kr.tuwien.ac.at
[2] Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
mdfwong@uiuc.edu

Abstract. We introduce a new method for Rectilinear Steiner Tree (RST) construction in a graph, using answer set programming. This method provides a formal representation of the problem as a logic program whose answer sets correspond to solutions. The answer sets for a logic program can be computed by special systems called answer set solvers. We describe the method for RST construction in the context of VLSI routing where multiple pins in a given placement of a chip are connected by an RST. Our method is different from the existing methods mainly in three ways. First, it always correctly determines whether a given RST routing problem is solvable, and it always produces a solution if one exists. Second, some enhancements of the basic problem, in which lengths of wires connecting the source pin to sink pins are restricted, can be easily represented by adding some rules. Our method guarantees to find a tree if one exists, even when the total wire length is not minimum. Third, routing problems with the presence of obstacles can be solved. With this approach, we have computed solutions to some RST routing problems using the answer set solver CMODELS.

1 Introduction

The Steiner tree problem is a combinatorial search problem that asks for a connected graph spanning a given set of points such that the total "length" of edges is minimum. In this paper, we consider a variation of Steiner trees whose edges are composed of horizontal or vertical line segments. Such a Steiner tree is called a Rectilinear Steiner Tree (RST) [1]. Here the length of an edge is the number of segments contained in that edge. This problem is NP-complete [2]. The computational problem we are interested in is to construct an RST connecting a set of given vertices in an undirected graph in the presence of obstacles. Consider, for instance, the tree shown in Figure 1 that connects all points labeled $p0, \ldots, p30$ without passing through the obstacles, shown in black. Since the total number of segments covered by this tree is minimum, this tree is an RST.

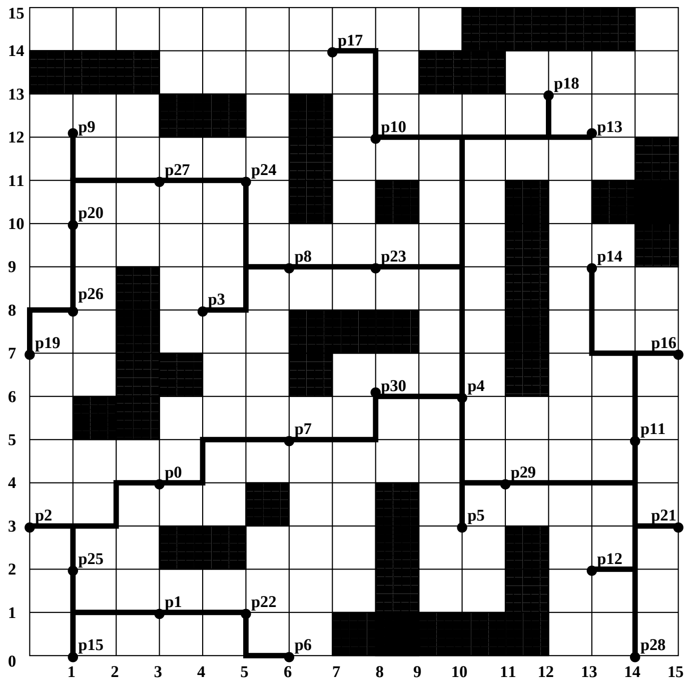


Fig. 1. An RST routing problem along with its solution.

We introduce a new method for RST construction using answer set programming (ASP) [3–5]—a declarative programming methodology that can be used to solve combinatorial search problems. The idea of ASP is to represent a given computational problem as a logic program whose answer sets (stable models) [6, 7] correspond to solutions to the given problem. Systems that compute answer sets for a logic program are called answer set solvers. For instance, SMODELS with its front-end LPARSE[3] is one of the answer set solvers that are currently available. System CMODELS[4] is another answer set solver, and it also uses LPARSE as its front-end. In the main part of this paper, we assume that the reader has some familiarity with the language of LPARSE [4].

We describe the RST construction problem as a logic program in the language of LPARSE, and use the answer set solver CMODELS to compute solutions. For instance, the solution presented in Figure 1 is computed using CMODELS.

In this paper, we consider RST construction in the context of VLSI routing. Automatic routing of wires has been the premier application of RSTs since Hanan's original paper [8]. (See [9] for the work on RST routing applications). The problem shown in Figure 1 can be viewed as a routing problem where multiple "pins", i.e., $p0, \ldots, p30$, are connected via an RST. This problem differs from other RST routing applications studied earlier in that there are obstacles

[3] `http://www.tcs.hut.fi/Software/smodels/` .

[4] `http://www.cs.utexas.edu/users/tag/cmodels/` .

on the grid. The obstacles are the parts of the chip occupied by previously routed nets (i.e., trees), or existing devices (e.g., memory, registers).

Our method always correctly determines whether a given RST routing problem is solvable, and it always produces a solution if it exists. Existing methods such as [10, 11] are mostly based on heuristics which cannot guarantee finding a routing solution even when one exists. The exact method of [12, 13] does not consider obstacles on the grid, and the exact method of [14] is for the problems with only 3 or 4 pins that lie on obstacle borders or on the border of the grid.

Unlike the existing methods, our approach can be applied to some variations of RST routing with restrictions on the lengths of wires connecting the source pin to the sink pins. Such restrictions are needed to meet signal delay constraints. It also can be applied to other wire routing problems where pairs of pins are connected by wires that do not intersect with each other and that do not go through obstacles. In this sense, our method is more elaboration tolerant [15].

In the following, after describing RST construction, we provide a detailed description of the method mentioned above as it applies to RST routing and its variations. After we present some experimental results, we conclude with a comparison of our approach to the related ones, and a discussion of how our method applies to other wire routing problems mentioned above. The programs describing the RST routing domain and the RST routing problems below can be obtained from the first author.

2 RST Construction

We describe RST construction as a graph problem. Recall that a *Steiner tree* for a set S of vertices in a graph (V, E) $(S \subseteq V)$ is a tree (V', E') that is a subgraph of (V, E) where $S \subseteq V'$ and the total length of edges in E' is minimum. A *Rectilinear Steiner Tree (RST)* for a set S of vertices in a graph (V, E) $(S \subseteq V)$ is a Steiner tree (V', E') for S in (V, E) such that the edges in E' are horizontal or vertical line segments. In this paper, the length of an edge is the number of segments contained in that edge. The problem of computing an RST is NP-complete [2].

In the computational problem we are solving, the input graph (V, E) is a grid. Given a set S of points on this grid, our goal is to find a tree in the grid that connects all the points in S such that the total number n of unit segments covered by the tree is minimum, or to determine that there does not exist such a tree. Note that since the edges E of the given graph are vertical or horizontal, the edges of the tree in this graph are also vertical or horizontal. Therefore, the trees we compute are RSTs.

To make sure that all points in S are connected via a tree on the grid, we use the following proposition:

Proposition 1 *For any finite graph (V, E), and any set $V' \subseteq V$, the following conditions are equivalent: (a) there exists a subgraph of (V, E) with the set V' of vertices that is a tree; (b) there exists a vertex $v \in V'$ such that every vertex in V' is reachable from v in V'.*

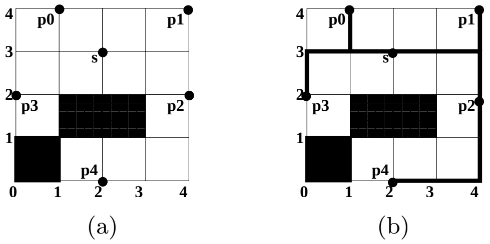


Fig. 2. (a) An RST routing problem with the source pin s and 5 sink pins $p0, \ldots, p4$. A solution to this problem is presented in (b).

```
const maxX = 4. const maxY = 4.

source(2,3).
sink(0,1,4). sink(1,4,4). sink(2,4,2). sink(3,0,2). sink(4,2,0).

obstacle(1,3,1). obstacle(0,1,0).
```

Fig. 3. Input file for the problem from Figure 2(a).

Proposition 1 allows us to specify one of the given points as the "source" point, and ensure that, for every other point ("sink" point), there is a path connecting it to the source point in the grid. When n is minimum, the union of paths connecting sink points to the source point forms an RST.

In some variations of RST construction, we put restrictions on the lengths of paths between the source point and sink points. For instance, we can ensure that the length of each path connecting a source point to a sink point is at most l. In another variation, we can put restrictions on the lengths of specific paths.

When we put restrictions on the lengths of paths, we may not find an RST. However, a smallest graph connecting the sink points and the source point that satisfies the given length restrictions is a tree:

Proposition 2 *Let $G = (V, E)$ be a finite graph, and s be a vertex in V. Let V' be a subset of V, and $H = (V, E')$ be a connected subgraph of G. If the total length of edges in E' is minimum subject to the condition, for every vertex $x \in V'$, the length of a path connecting x to s in H is less than some given number l_x, then H is a tree.*

3 Input and Output of CMODELS

In our approach, the solutions of the RST routing problems are characterized by the truth values of the atoms `covered(S,X,Y)` ("If `S=h` then the horizontal segment connecting the points `(X,Y)` and `(X+1,Y)` is covered by the graph; if `S=v` then the vertical segment connecting the points `(X,Y)` and `(X,Y+1)` is covered by the graph").

Consider, for instance, the problem shown in Figure 2(a). This problem is described to CMODELS by the file presented in Figure 3. The size of the grid is represented in that file by the constants `maxX` and `maxY`. These numeric values are defined in each particular routing problem. The source pin and the sink pins are specified. Next, we describe the shape of the obstacles in this example by `obstacle(1,3,1)` and `obstacle(0,1,0)`. Here, `obstacle(X1,X2,Y)` expresses that there is an obstacle occupying the points covered by the rectangle defined by the points `(X1,Y)`, `(X2,Y)`, `(X1,Y+1)`, and `(X2,Y+1)`.

To find a solution to this problem we need the file `rst.lp`, describing the RST routing domain, and the file `point.lp`, describing the grid points that are not covered by obstacles. Parts of file `rst.lp` are discussed in the next section. Given the files `rst.lp` and `point.lp` with the file presented in Figure 3, and an upper bound `n=12` on the total wire length, CMODELS finds the following tree:

```
Answer set:
covered(h,0,3) covered(h,1,3) covered(h,2,0) covered(h,2,3)
covered(h,3,0) covered(h,3,3) covered(v,0,2) covered(v,1,3)
covered(v,4,0) covered(v,4,1) covered(v,4,2) covered(v,4,3)
```

shown in Figure 2(b). CMODELS also determines that there is no tree whose total length is less than `12`. It follows that the tree above is a solution.

4 The RST Routing Domain

As described in Section 2, we construct RSTs from paths connecting sink points to the source point. Every path is characterized by the truth values of the atoms `in(S,N,X,Y)` ("If `S=h` then the horizontal segment connecting the points `(X,Y)` and `(X+1,Y)` occurs in Path `N`—a path connecting sink pin `N` to the source pin; if `S=v` then the vertical segment connecting the points `(X,Y)` and `(X,Y+1)` occurs in Path `N`").

In the file `rst.lp`, first sets of atoms of the form `in(S,N,X,Y)` are "generated" by the choice rule

```
{in(S,N,X,Y)} :- segment(S), path(N), point(X,Y).
```

where `point(X,Y)` defines, in the file `point.lp`, the grid points that are not blocked by obstacles. Then these sets are "tested" with some constraints expressing the following:

(i) the set describes a subgraph of the grid,
(ii) the subgraph contains paths connecting the sink points to the source point, and
(iii) the size of the subgraph is minimum.

Due to Proposition 1, condition (ii) expresses that the subgraph contains a tree connecting the sink points and the source point. With condition (iii), this subgraph forms a solution to the RST routing problem.

First, for (i), we eliminate the sets that contain vertical segments connecting a point `(X,Y)` on the grid that is not blocked by an obstacle to the point `(X,Y+1)` that is blocked by an obstacle (or is above the upper edge of the grid):

```
:- in(v,N,X,Y), path(N), point(X,Y), not point(X,Y+1).
```

A similar constraint is added for horizontal segments.

Next, for (ii), we eliminate the sets that do not contain paths connecting the sink points to the source point by adding the following constraints.

To express that the end points of a path should be in that path, we define the atom `at(N,X,Y)` ("the point `(X,Y)` is in Path `N`"). We express that the end points of Path `N`, specified in a problem description file, should be in Path `N` by the constraint:

```
:- not at(N,X,Y), ends(N,X,Y).
```

We need to make sure that the end points of Path `N` cannot be connected to two or more points in the path, whereas each of the other points of Path `N` should be connected to exactly two points in the path. For that, we define the atom `at(N,X,Y,D)` ("the unit segment that begins at the point `(X,Y)` and goes in the direction `D` occurs in Path `N`"). We make sure that the end points of Path `N` cannot be connected to two or more points by the constraint

```
:- 2{at(N,X,Y,D):direction(D)},
   path(N), ends(N,X,Y).
```

Each of the other points of Path `N` cannot be connected to exactly one point

```
:- 1{at(N,X,Y,D):direction(D)}1,
   path(N), point(X,Y), not ends(N,X,Y).
```

and cannot be connected to more than three points.

```
:- 3{at(N,X,Y,D):direction(D)},
   path(N), point(X,Y), not ends(N,X,Y).
```

That is, each of these points should be connected to exactly two points.

Finally, for (iii), we eliminate the sets where the total wire length is larger than `n` by adding the constraint

```
:- n+1 {covered(S,X,Y):segment(S):point(X,Y)}.
```

where `covered(S,X,Y)` is defined as

```
covered(S,X,Y) :- in(S,N,X,Y), segment(S), path(N), point(X,Y).
```

If the total wire length n is minimum then the graph generated by the program above is an RST. For instance, with the routing domain described above, CMODELS finds the tree presented in Figure 1 with `n=76`. For `n=75`, CMODELS determines that there is no answer set for the program; therefore, the tree presented in Figure 1 is an RST.

5 Restricting the Lengths of Wires

An RST routing problem may involve constraints on the lengths of some wires connecting the sink pins to the source pin, to meet some signal delay constraints. We can express that any wire cannot be longer than a specific value, say `l`, by adding to the problem description the constraint

```
:- l+2 {at(N,X,Y):point(X,Y)}, path(N).
```

In other words, no path connecting the source pin to a sink pin covers `l+1` unit segments on the grid. Consider, for instance, the problem shown in Figure 4(a), along with a solution. In this solution, the total wire length is minimum (`n=26`), and each wire length can be at most 13 (`l=13`). There is no solution for this problem with `n=26` and `l=12`.

We can put restrictions on the length of a specific wire as well. For instance, for the problem described in Figure 4(a), we can express that the wire corresponding to Path 0 cannot be longer than 8 by adding to the problem description the constraint

```
:- 8+2 {at(0,X,Y):point(X,Y)}.
```

After this change, a tree can be found by CMODELS as shown in Figure 4(b). Here, the total wire length is 28, and each wire length is at most 13; there is no solution with the total wire length being less than 28.

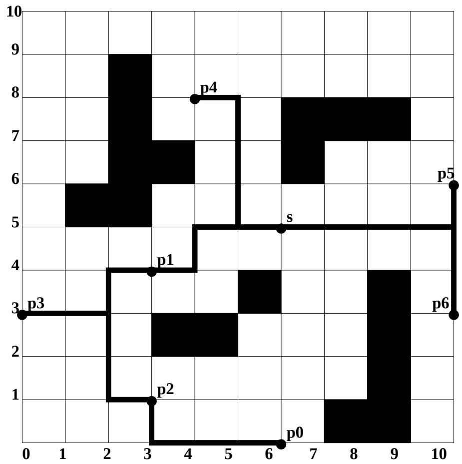
When we put restrictions on the lengths of wires, we may not find an RST. For instance, when we restrict the length of Path 0 above, a connected graph with minimum total wire length `n=26` does not exist. However, due to Proposition 2, we know that a smallest graph connecting the sink pins to the source pin that satisfies the given length restrictions is a tree. For instance, the graph in Figure 4(b) satisfies the restriction on the length of Path 0, and there is no such graph that has a smaller total length (`n<28`).

6 Constructing Trees

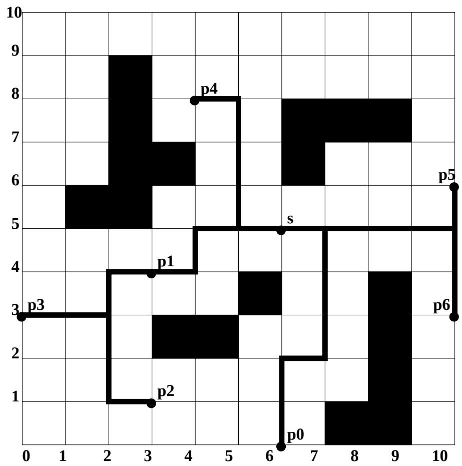
If the upper bound n for the total number of the segments covered by the paths is not tight enough then the graph generated by the program described in Section 4 may not be a tree, i.e., there may be cycles as in Figure 5(a). Since a path connecting a sink point to the source point does not contain any cycles, these cycles are of the two forms: either they do not contain any of the sink points and the source point, or they are formed by different paths. In Figure 5(a), we can see both kinds of cycles.

To make sure that the graph generated by the program described in Section 4 is a tree, due to the definition of a tree, we need to make sure that every point covered by the graph is reachable from the source point via exactly one path. For that, we add to the program describing the RST routing domain the constraint

```
:- at(N,X,Y), at(N1,X,Y), not reachable(N,N1,X,Y),
   point(X,Y), path(N;N1), N<=N1.
```

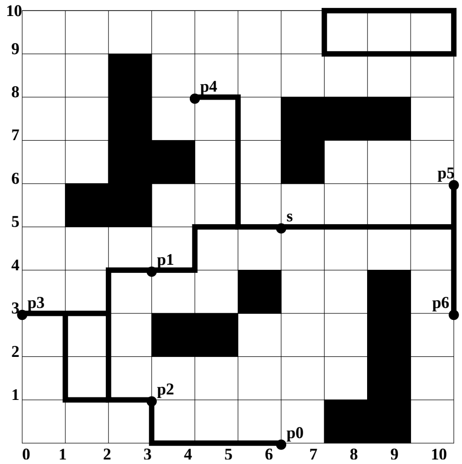


(a)

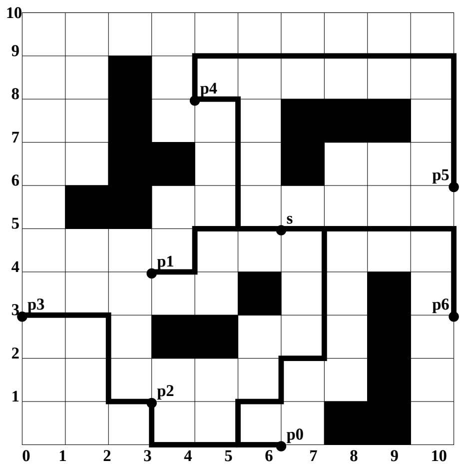


(b)

Fig. 4. (a) An RST routing problem with 7 sink pins, along with a solution where the total wire length is minimum, and the length of each wire is at most 13. (b) A near-solution to the RST routing problem in (a) where the length of Path 0 is required to be at most 8.



(a)



(b)

Fig. 5. (a) A cyclic graph and (b) a tree connecting the sink pins $p0, ..., p6$ to the source pin s.

Here `reachable(N,N1,X,Y)` expresses that every point `(X,Y)` covered by Paths `N` and `N1` is reachable from the source point via the segments covered by both paths, and it has a straightforward recursive definition. The base case is defined by the rule

```
reachable(N,N1,X,Y) :- source(X,Y), path(N;N1), N <= N1.
```

and the inductive step is defined by rules like

```
reachable(N,N1,X+1,Y) :- reachable(N,N1,X,Y), path(N;N1), N <= N1,
  in(h,N,X,Y), in(h,N1,X,Y), point(X,Y), point(X+1,Y).
```

In the case of $\texttt{N} = \texttt{N1}$, the constraint above expresses that every point covered by a path is reachable from the source point. This eliminates the cycles of the first form, and makes sure that the graph is connected.

In the case of $\texttt{N} \neq \texttt{N1}$, the constraint expresses that a point covered by two different paths is not reachable from the source point via two different paths. This eliminates the cycles of the second form, and, since a path connecting a sink point to the source point does not contain any cycles, it makes sure that, in a connected graph, paths do not form any cycles.

For instance, for the problem presented in Figure 5(a), if the upper bound on the total wire length is set to 40 then, with the constraint above, CMODELS computes the tree in Figure 5(b). If the total wire length is minimum (`n`=26) then CMODELS computes the RST in Figure 4(a).

Note that the program describing the routing domain in Section 4 is a "tight" program, and it becomes "nontight" with the definition of reachability above [16].

7 Experimental Results

In the computational problem we are solving, the input consists of a program describing the locations of the pins and the obstacles on a grid (Section 3), the program describing the RST routing domain (Section 4), and an upper bound n on the total wire length. Given this input, CMODELS computes a graph that connects the pins without going through the obstacles where the total wire length is at most n, or determines that such a graph does not exist. Our goal is to compute an RST (Section 2), if it exists. For that, first we make sure that the given upper bound is not too small so that CMODELS can compute a solution, and then we call CMODELS on the given input programs with decreasing values of the upper bound until we reach the minimum total wire length. (Alternatively, binary search can be used.) We reach the minimum total wire length n, when CMODELS computes a solution for the upper bound n, and determines that a solution does not exist for the upper bound $n - 1$.

To make the computation more efficient, we introduce two "circles" with a given `radius` around the endpoints of a path we are looking for, and require that the path be contained in the union of these circles. This modification sometimes improves the computation time of CMODELS significantly. For instance, for the problem shown in Figure 1, introducing circles of `radius=8` around the endpoints improves the computation time of a solution (`n=76`) by a factor of 10. However, adding circles around endpoints can prevent CMODELS from finding a solution to a solvable problem if the value of `radius` is too small. For instance, for the problem above, if we reduce `radius` by 1 then we cannot find a solution. For this reason, when we reach an upper bound, for which there is no solution with circles, we continue the process of computing an RST without introducing any circles.

Another way to make the computation of an RST more efficient is to prohibit "adjacencies" of unit segments. (Two unit segments are *adjacent* if they form the opposite sides of a unit square.) This is possible due to the following proposition.

Proposition 3 *Let (V, E) be a rectilinear tree where each edge is a unit segment. Then there exists a rectilinear tree (V, E') without any adjacencies of unit segments such that $|E'| = |E|$.*

With no adjacencies, for instance, the computation time of a solution (`n=76`, `radius=8`) for the problem in Figure 1 can be improved by a factor of 7.

Given the programs describing a specific routing problem and describing the RST routing domain, with `n` and `radius` specified, CMODELS transforms the programs into a propositional theory [17], and calls a SAT solver to compute the models of this theory, which are identical to the answer sets for the given programs.[5] In our experiments, we use CMODELS (Version 2.01) and LPARSE (Version 1.0.13), with the SAT solver ZCHAFF (Version Z2003.11.04).[6] All CPU times presented below are in seconds for a PC with a 733 MHz Intel Pentium III processor and 256MB RAM, running SuSE Linux (Version 8.1).

We consider 4 problems $A, \ldots, D$ on a grid of size 15×15 with 15, 20, 25, 30 sink pins respectively. Note that the RST problems in VLSI routing typically have small number of sink pins (e.g., less than 10), so our test problems are of reasonable size for that application. Figure 6 shows the computation times for these problems, when `radius=8`. This value of `radius` is small enough to find a solution to each problem with the minimum total wire length. The character $*$ denotes that the problem does not have a solution. For instance, problem A does not have a solution where the total wire length is 57; CMODELS finds this out in 67 seconds, without introducing any circles.

When we relax the restriction on the total wire length to compute a tree (as in Section 6), the computation times sometimes decrease. For instance, for problem B, a solution with the minimum total wire length (`n=68`) is computed in 17 seconds. If we allow the total wire length to be at most `85`, a tree of size 79 is computed in 7 seconds.

Figure 7 shows the computation times for problems $A, \ldots, D$ when the length of each wire connecting a sink pin to the source pin is bounded by `l`. The results are for `radius=8`. Compared to the results in Figure 6, we can see that restricting the length of each wire by `l` sometimes increases the computation

[5] For nontight programs, CMODELS operates with a "generate and test" approach [18].
[6] `http://www.ee.princeton.edu/~chaff/zchaff.php` .

Problems	# of sink pins	`n`	CPU time
A	15	58	14
		57*	67
B	20	68	17
		67*	128
C	25	74	21
		73*	450
D	30	76	23
		75*	164

Fig. 6. Computation times for problems $A, \ldots, D$ on a 15×15 grid when the total wire length (`n`) is minimum.

Problems	# of sink pins	`n`	`l`	CPU time
A	15	58	16	17
			15*	16
B	20	68	19	40
			18*	59
C	25	74	17	32
			16*	40
D	30	76	21	69
			20*	82

Fig. 7. Computation times for problems $A, \ldots, D$ on a 15×15 grid when the total wire length (`n`) is minimum, and each wire length is at most `l`.

time. For instance, for problem D, computing a solution with `n=76` takes 23 seconds whereas a solution with `n=76` and `l=21` takes 69 seconds.

Another answer set solver that we can use to solve the routing problems with our formalizations above is SMODELS. However, solutions to the problems above can be computed faster using CMODELS. For instance, a problem with 7 pins on a 10×10 grid can be solved by CMODELS in less than a second whereas a solution can not be found in less than a minute with SMODELS (Version 2.27).

One way to extend our approach to problems with larger grid size is to consider a small set of points on the grid that would be sufficient to construct a tree. These points can be identified by the following hierarchical process in a multi-level manner. At the first level, we consider the given $n \times n$ grid as an $(n/m) \times (n/m)$ small grid. Each point of this small grid represents a $m \times m$ subgrid of the given grid. We construct an RST on this small grid using CMODELS, and obtain the points of the given grid covered by this RST. We repeat this process by decreasing m at each level until we obtain a small set of points on the grid that would be sufficient to construct a tree. Note that a tree computed by this process is not guaranteed to have the minimum total length. We have implemented the algorithm above as a PERL program, and experimented with two problems E and F on a 100×100 grid with 5 and 10 pins respectively. For each problem, first we have obtained a subgraph of the grid that is sufficient to find a tree that connects the given pins, by considering the given grid first as a 10×10 grid, then as a 20×20 grid, then as a 25×25 grid, and finally as a 50×50 grid. This process takes 20 seconds for problem E and 36 seconds for

Problems	# of pins	`n`	CPU time
E	5	111	16
		115	9
		119	6
F	10	157	75
		163	64
		170	32

Fig. 8. Computation times for problems E and F on a 100×100 grid when the total wire length is at most `n`, using hierarchical routing.

problem F. After that, we have computed some trees connecting the given pins over these subgraphs. Figure 8 shows these computation times when `radius=30`. For instance, we can compute a tree of size 157 in 75 seconds for problem F.

8 Discussion

We have introduced a formal method for RST construction, in the context of VLSI routing, using answer set programming. This method provides a concise formal representation of the problem as a logic program whose answer sets correspond to solutions. The answer sets for the given formalism are computed using the answer set solver CMODELS.

Our method always correctly determines whether a given RST routing problem, possibly with the presence of obstacles, is solvable, and finds a solution if one exists. The two other exact methods for RST routing are presented in [14] and [12, 13]. In [14], the authors consider RST problems with 3 or 4 pins. They identify the possible topologies, and then describe exact algorithms for each topology. Our method is more general in that any number of pins can be connected by an RST without having to identify all possible topologies. In [12, 13], the authors do not consider obstacles. The idea for RST construction is first to generate some full Steiner trees (Steiner trees where the source and the sink points are the leaves of the tree) for subsets of the given pins, and then to construct a Steiner tree from a subset of these full Steiner trees. Our method is more general in that RST construction in the presence of obstacles can be handled.

Other existing methods for RST construction such as [10, 11] are based on heuristics which cannot guarantee finding a routing solution even when one exists.

Another difference of our method from the ones mentioned above is that it allows us to solve variations of RST routing where some restrictions are put on the lengths of paths connecting the sink points to the source pin. For instance, we can ensure that the length of each path connecting a source pin to a sink pin is at most l. In another variation, we can put restrictions on the lengths of specific paths. In such cases, our method guarantees that we find a tree even when the total wire length is not minimum.

In some routing problems, the goal is to connect pairs of pins with wires, and the solutions consist of paths that are required not to intersect with each

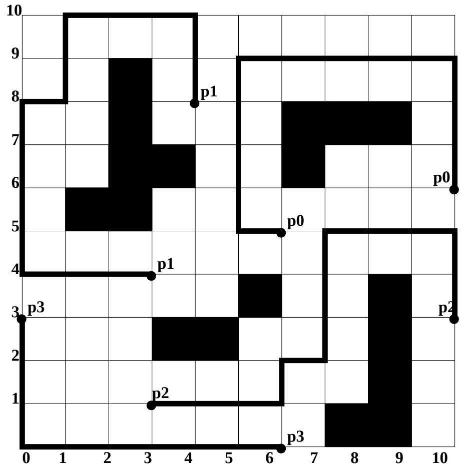


Fig. 9. A routing problem where pairs of pins are connected.

other, and that do not intersect with obstacles. For instance, a routing problem, along with its solution computed by CMODELS, is displayed in Figure 9. Such problems can be solved with our formalization (Section 2), by adding to the problem description the constraint

```
:- at(N,X,Y), at(N1,X,Y), N<N1, path(N), path(N1), point(X,Y).
```

expressing that no two paths intersect.[7] As in the case of RST routing problem, variations of these routing problems, in which lengths of wires and distances between them come into play, can easily be represented by slight modifications. For instance, we can solve bus routing problems where all wires connecting pairs of pins should be of the same length so that the signal delays through all wires are equal. In another enhancement, we can prohibit the adjacency of wires, to avoid signal interferences. In this sense, this representation method is elaboration tolerant.

Acknowledgments We have had useful discussions with Vladimir Lifschitz on the formalizations of the problems. Derek Corneil has helped us with Proposition 2. Ilkka Niemelä has suggested to us introducing "circles" around the end points of paths. Sinan T. Erdoğan has implemented a program to visualize the

[7] For routing problems, where pairs of pins are connected, some other ASP representations are presented in [19–21], and one is due to Tommi Syrjänen (personal communication, July 31, 2000). With the formalization above, such routing problems can be solved more efficiently. For instance, a problem with 20 pairs of pins on a grid of 15×15 can be solved in 21 seconds with our program, using about 80MB of memory. With the encoding of [20], the computation time is 57 seconds and the used memory is about 300MB. A solution for this problem cannot be found in less than a minute with the other formalizations. In our formalization, paths are defined in terms of segments, and the grid does not have to be rectangular. This allows us to solve RST routing problems avoiding cycles and with a hierarchical approach.

solutions computed by CMODELS. Selim T. Erdoğan and Vladimir Lifschitz have provided helpful comments on earlier drafts. Much of this work was done while the first author was at the University of Toronto, and this was made possible by Hector Levesque and Ray Reiter. The work of the first author was partially supported by FWF under project P16536-N04. The work of the second author was partially supported by NSF under grants CCR-0244236 and CCR-0306244.

References

1. Hanan, M.: On Steiner's problem with rectilinear distance. SIAM Journal on Applied Mathematics **14** (1966) 255–265
2. Garey, M.R., Johnson, D.S.: The rectilinear Steiner tree problem is NP complete. SIAM Journal of Applied Mathematics **32** (1977) 826–834
3. Marek, V., Truszczyński, M.: Stable models and an alternative logic programming paradigm. In: The Logic Programming Paradigm: a 25-Year Perspective. (1999) 375–398
4. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence **138** (2002) 181–234
5. Lifschitz, V.: Answer set programming and plan generation. Artificial Intelligence **138** (2002) 39–54
6. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Logic Programming: Proc. of the Fifth Int'l Conference and Symposium. (1988) 1070–1080
7. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing **9** (1991) 365–385
8. Hanan, M.: Net wiring for large scale integrated circuits. Tech. report, IBM (1965)
9. Lengauer, T.: Combinatorial Algorithms for Integrated Circuit Design. (1990)
10. Nielsen, B.K., Winter, P., Zachariasen, M.: An exact algorithm for the uniformly-oriented steiner tree problem. In: Proc. of ESA-10 (2002) 760–771
11. Zhou, H.: Efficient steiner tree construction based on spanning graphs. In: ACM Int'l Symposium on Physical Design. (2003)
12. Warme, D.M.: A new exact algorithm for rectilinear steiner trees. In the Int'l Symposium on Mathematical Programming (1997)
13. Zachariasen, M.: Rectilinear full steiner tree generation. Networks **33** (1999) 125–143
14. Ganley, J.L., Cohoon, J.P.: Routing a multi-terminal critical net: Steiner tree construction in the presence of obstacles. In: Proc. of the Int'l Symposium on Circuits and Systems. (1994) 113–116
15. McCarthy, J.: Elaboration tolerance. In progress (1999)
16. Erdem, E., Lifschitz, V.: Tight logic programs. Theory and Practice of Logic Programming **3** (2003) 499–518
17. Lloyd, J.: Foundations of Logic Programming. (1984)
18. Lierler, Y., Maratea, M.: Cmodels-2: SAT-based answer sets solver enhanced to non-tight programs. In: Proc. of LPNMR-7. (2004) 346–350
19. Erdem, E., Lifschitz, V., Wong, M.: Wire routing and satisfiability planning. In: Proc. of the First Int'l Conference on Computational Logic. (2000) 822–836
20. Erdem, E.: Theory and Applications of Answer Set Programming. PhD thesis, University of Texas at Austin (2002), Department of Computer Sciences.
21. East, D., Truszczyński, M.: More on wire routing with ASP. In: Working Notes of the AAAI Spring Symposium on Answer Set Programming. (2001)