SWITCHBOX ROUTING BY PATTERN MATCHING[1]

M. STARKEY
T.M. CARTER

UUCS-91-004

Department of Computer Science
University of Utah
Salt Lake City, UT 84112, USA

March 7, 1991

Abstract

Many good algorithms have been designed that provide good solutions to the wire routing problem in VLSI. Unfortunately, many of these algorithms only consider a small subset of different parameters such as number of layers, routability of layers and technology. We believe that these algorithms can be applied generally to any set of parameters by implementing the algorithms as a decscription that allows them to take advantage of this flexibility. We propose that routing algorithms either use patterns directly or can be converted to use patterns. We present a powerful formalism for describing these patterns.

[1]This work supported in part by grant J-FBI-89-102.

Switchbox Routing by Pattern Matching

M. Starkey
Department of Computer Science
University of Utah
Salt Lake City, UT 84112

T. M. Carter
Department of Computer Science
University of Utah
Salt Lake City, UT 84112

Abstract

Many good algorithms have been designed that provide good solutions to the wire routing problem in VLSI. Unfortunately, many of these algorithms only consider a small subset of different parameters such as number of layers, routability of layers and technology. We believe that these algorithms can be applied generally to any set of parameters by implementing the algorithms as a description that allows them to take advantage of this flexibility. We propose that routing algorithms either use patterns directly or can be converted to use patterns. We present a powerful formalism for describing these patterns.

1 Introduction

The problem of wire routing has been present for many years in a number of different forms. Generally, this problem describes how to find the best path from one place to another where some paths cost more to follow than others. In computer science this problem is known as the Traveling Salesperson Problem. Routing wires adds another twist to this problem. Instead of having just one tour of the nodes, there are a number of tours which must use the same set of available paths. No path can be shared among tours and generally a lower total cost of all tours is more important than the lowest cost of any individual tour. This problem is NP-Complete. However, many good algorithms have been developed to provide good solutions in reasonable amounts of computational time.

A typical method used to find solutions reasonably quickly is to use the constraints imposed by the environment to reduce the size of the problem. For example, assumptions are often made about the number of available layers and the directions routing occurs on these layers. Unfortunately, these assumptions make applying an algorithm to a different set of parameters almost impossible. By providing a flexible description language, these routing algorithms can easily be used in many different environments. This description must be powerful enough to allow the algorithm to be implemented easily as well as not create a large amount of overhead which greatly increases the execution time of the algorithm.

The method we have found to be the most flexible in this regard involves patterns. Patterns have been used in a number of different algorithms. They are found in some of the best existing algorithms.

The most closely related work was presented by Soukup and Fournier[1]. They identified the regularity of circuit design and described a pattern router to utilize this. They developed a simple pattern description language and provided a number of results of routing printed circuit boards.

WEAVER[2] has provided some of the best results in switchbox routing. This Knowledge-based routing system has a number of "experts" which implement different phases of wire routing. One of these experts is the pattern router expert which attempts to find certain patterns and route them.

The BEAVER[3] router is also targeted towards routing switchboxes. It attempts to interconnect pins with certain patterns. Some patterns are given priority over others. It provides good results on the classic examples.

In addition to these direct implementations of patterns, Burstein and Pelavin use small patterns. These patterns describe all of the possible 2xN Steiner Trees which are possible in their Hierarchical Routing algorithm[4].

These approaches to using pattern matching to route wires prove that patterns have value in finding good solutions in reasonable amounts of time. We expand on this assumption by proposing that many routing algorithms can be implemented in terms of patterns. Also, these patterns increase the flexibility and power of routing algorithms by allowing them to operate in many different environments. We present a formal pattern grammar and some powerful properties of the chosen description method.

2 Environment Independence

A number of good routing algorithms exist which have been applied to problems in certain environments. Usually, these environments have restrictions on the number of layers which can be used to route and which directions can be used along these layers. There are many different permutations of these constraints. The goal is to enable utilizing some of the good routing algorithms in any environment. Some of these environments include different numbers of layers, different technologies such as Gallium Arsenide and angled routing directions.

Most algorithms are designed around a process which permits two layers of routing material. There are then two ways to allocate the tracks and columns for routing. One is a strict definition of how the layers are to be used. This method defines one layer to have horizontal wires and another vertical wires. The only method of jogging around an obstacle is to change layers, jog around the obstacle, and change back. The extra vias which are introduced are usually undesirable. This has led to an environment where the layers each have a preferred direction but this can be violated to make small jogs or doglegs.

Introducing a routing algorithm designed for one of these environments to the other environment can be detrimental to routing a circuit. If an algorithm expecting to make same-layer jogs is placed in an environment where doing so is impossible, it could conceivably fail when it tried to avoid an obstacle. Similarly, if an algorithm was introduced to an environment where the directions were only preferred then the result may not be as compact as possible and may contain an unacceptable number of vias.

These differences require rewriting a routing algorithm to be flexible as far as the directions it can expect, the layers these directions can be on and the number of layers available. It may also take into account the relative cost of using certain layers over others. This is facilitated by using a powerful representation such as pattern descriptions.

In addition to having a method of describing patterns, there must be a method of interacting with the underlying system. Four functions are defined to provide this interface. The $owner(x_1, y_1, z_1, x_2, y_2, z_2)$ function returns the net to which the straight segment between these points has been allocated or 0 if it is not allocated. In traditional routing terminology, the x values represent the columns, the y values are the tracks and the z values correspond to the layer number. Wire segments can be unavailable if they are eliminated due to wires or cells already being placed there. These obstacles may be functional elements which must be routed around, *owner* will return a non-zero, invalid net number in such cases.

Three additional functions are also required. The $allocate(net, x_1, y_1, z_1, x_2, y_2, z_2)$ function allocates the segment between (x_1, y_1, z_1) and (x_2, y_2, z_2) to *net*, $free(x_1, y_1, z_1, x_2, y_2, z_2)$ will deallocate the segment and $cost(x_1, y_1, z_1, x_2, y_2, z_2)$ returns the cost of the wire segment.

The *cost* function is very important for reducing the search space of the problem and also to ensure that a lowest cost route is obtained. In wire routing, there are a number of costs which are associated with different

solutions to interconnecting an entire circuit. These costs are dependent on physical properties of the routing media. Capacitance and resistance play a major role in circuits. Limiting the number of vias used is also important as is minimizing the number of jogs in a connection.

3 Pattern Descriptions

Patterns offer a very powerful method with which to describe algorithms. Patterns can describe everything from a single segment of a wire to a complex pattern shape.

3.1 Describing Patterns as Regular Expressions

A powerful method of describing patterns is as regular expressions. Regular expressions are a natural way to think of describing patterns. For example,if a string of symbols is desired which starts with a 1 followed by one or more 0s then any number of 0s or 1s finally terminated by a 1, then it can be simply described as $10^{+}(0+1)^{*}1$. The pattern 10001 satisfies this regular expression but 110 does not. Although the acceptance of strings for this regular expression can be analyzed by inspection, regular expressions can become complex and acceptance can be difficult to verify.

Regular expressions utilize some operations that facilitate describing complex expressions. These operations are the Kleene star (*) which permits zero or more replications of the symbol, the plus ($^{+}$) which signifies one or more replications of the symbol and the or symbol (+) which provides alternation among symbols. In addition, parentheses identify sub-expressions.

In wire routing, these regular expressions are not only used to accept patterns but also define how the pattern should be generated. Therefore, information must be encoded in the characters which are used for the above operations. The meanings of the or symbol and the parentheses remain, the changes affect the Kleene star and the plus. When running wires in certain directions, we usually wish to maximize or minimize the length of the connection in that direction. For example, vias are usually desired to be minimized. Other directions may be preferred and therefore maximized. This information is encoded in the symbols $^{<}$ and $^{>}$. The $^{<}$ symbol replaces the Kleene star and is defined to allow zero or more replications of the symbol, trying to minimize its use. The $^{>}$ is similar to the plus, there must be one or more replications of the symbol and its use should be maximized. Naturally, the * and $^{+}$ characters could be used but the new symbols clarify the goals of the regular expression in terms of its function in routing.

Traditionally, only six directions are used for wires. These are horizontal tracks, vertical columns and vias. Picturing this by looking down on the routing area and superimposing compass coordinates on the wires available for routing, a set of symbols is developed. This set of symbols is enough for most routing environments, however, other symbols can be easily added for any additional directions or any other distinctions between directions. Regular expressions are said to have an alphabet, Σ, in this case $\Sigma = \{N,S,E,W,U,D\}$. The first four symbols represent the directions on the compass as described above. The last two symbols specify changing layers.

An example of a regular expression is one which represents a step of the "Greedy" channel router[5]. As this algorithm proceeds from left to right column by column, each point on the top and bottom of the channel which is in the current column must be brought into the channel only as far as required to continue rightward on a track. The description which can be converted to a regular expression is "minimize the wire segments in the north or south direction (depending on whether the connection is on the bottom or top) where there is an available track which allows the connection to extend to the right". As a regular expression this becomes $(N+S)^{<}(U+D)^{<}E$. The $(U+D)^{<}$ is required since a layer change may be necessary to change from one

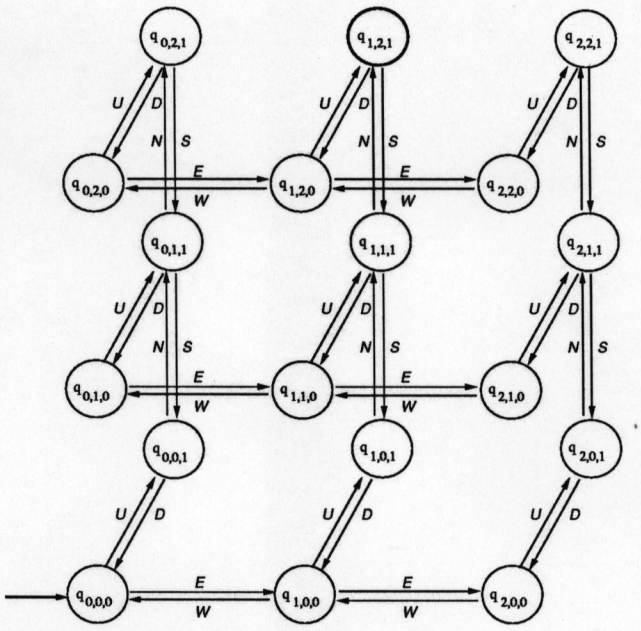

Figure 1: The Deterministic Finite Automaton corresponding to the bounds $0 \leq x \leq 2$, $0 \leq y \leq 2$,$0 \leq z \leq 1$ where only column wires are allowed on layer 1 and track wires on layer 0. The arrow indicates the start state for a string starting at $q_{0,0,0}$ and proceeding to $q_{1,2,1}$.

direction to another. This direction change will be minimized and will therefore not generate any symbols if the direction can be changed without changing layers.

Using regular expressions provides a very simple, concise and understandable method to describe patterns. Regular expressions also have some other desirable properties which can be exploited. Regular expressions are closed under certain operations. These operations include replicating expressions, extracting sub-expressions, concatenating multiple expressions and combining expressions into more complex expressions. These closures ensure that the resulting regular expression is also regular and therefore can be accepted by a Deterministic Finite Automaton. These operation are extremely useful for regular expressions which represent wire routing directions. Hopcroft and Ullman provide an excellent description of regular expressions and Deterministic Finite Automata in [6].

3.2 Accepting Patterns With Deterministic Finite Automata

The acceptance of strings of symbols matching certain regular expressions is performed by a Deterministic Finite Automaton (DFA). A DFA is defined by a 5-tuple, $(Q, \Sigma, \delta, q_0, F)$. The state space, Q, is a set of all states in the DFA. Σ is the alphabet whose symbols form the string to be accepted. The δ function describes the transition from one state to another beginning with the start state, q_0, until one of the final states in the set F is reached.

For routing wires, the obvious method to build the DFA is to have the arcs represent wires in the circuit. Therefore, if a transition is taken from one state to another, the arc used represents the wire which is allocated to the net being routed. Figure 3.2 shows the DFA for a traditional routing environment. This environment has wires running horizontally on one layer and vertically on a second layer. Layers can be connected by running vias between them. To clarify the correspondence between position in the circuit and the wires being allocated, a state is identified as $q_{x,y,z}$ where x is the column, y is the track and z represents the layer.

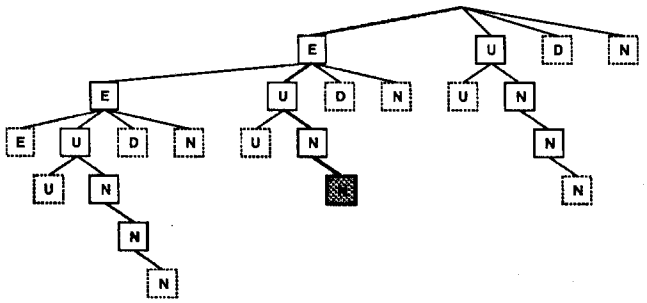

Figure 2: Search tree for the pattern $\boldsymbol{E}^{<}(\boldsymbol{U}+\boldsymbol{D})^{<}\boldsymbol{N}^{<}$ starting at $q_{0,0,0}$ and terminating at $q_{1,2,1}$. Dashed boxes indicate pruned nodes, the shaded box is the node where the solution is reached.

Each of the components of the DFA $M = (Q, \Sigma, \delta, q_0, F)$ can now be described. To increase the power of the DFA, a bounding box of the area defined as $[x_{min}, x_{max}]$, $[y_{min}, y_{max}]$, $[z_{min}, z_{max}]$ must be provided. Also the node where the route is to start must be specified. Initially, the DFA will be described for finding point to point connections. This constraint will be relaxed to allow routed wires to stop when a segment of the same net is reached. Also, the initial definition will only provide for a pattern using unallocated segments, this constraint will also be relaxed.

The state space consists of all of the states which are nodes in the routing area. More formally, $Q = \{q_{x,y,z} | x_{min} \leq x \leq x_{max}, y_{min} \leq y \leq y_{max}, z_{min} \leq z \leq z_{max})\}$. The alphabet is the same alphabet as for the regular expressions, $\Sigma = \{\boldsymbol{N},\boldsymbol{S},\boldsymbol{E},\boldsymbol{W},\boldsymbol{U},\boldsymbol{D}\}$. The δ function describes which arcs are present for routing the current net. The *owner* function, described previously, is required to find the owner of an arc. The δ function then becomes,

$\delta(q_{x,y,z},\boldsymbol{E}) = q_{x+1,y,z} | owner(x,y,z,x+1,y,z) = 0$ $\quad$ $\delta(q_{x,y,z},\boldsymbol{W}) = q_{x-1,y,z} | owner(x,y,z,x-1,y,z) = 0$

$\delta(q_{x,y,z},\boldsymbol{N}) = q_{x,y+1,z} | owner(x,y,z,x,y+1,z) = 0$ $\quad$ $\delta(q_{x,y,z},\boldsymbol{S}) = q_{x,y-1,z} | owner(x,y,z,x,y-1,z) = 0$

$\delta(q_{x,y,z},\boldsymbol{U}) = q_{x,y,z+1} | owner(x,y,z,x,y,z+1) = 0$ $\quad$ $\delta(q_{x,y,z},\boldsymbol{D}) = q_{x,y,z-1} | owner(x,y,z,x,y,z-1) = 0$

The start state, q_0, is $q_{x,y,z}$ where the node at (x,y,z) is where the route is to start. The final state at node (x_f, y_f, z_f) will be the only member of the set $F = \{q_{x_f,y_f,z_f}\}$.

The DFA defined above demonstrates a route which will only follow unused segments of wire. Usually, a connection is made if the current net being routed, *current_net*, connects to a wire segment already allocated to this net. Then the test in each δ function can be relaxed to be $owner(x,y,z,x',y',z') = 0$ or *current_net*. The set of acceptable final states becomes $F = \{q_{x_f,y_f,z_f}\} \cup \{q_{x,y,z} | owner(x,y,z,x,y,z) = current\_net\}$.

This Deterministic Finite Automaton is very powerful for accepting patterns in the current routing environment. It is very general and can be applied to any physical constraints. The *owner* function provides the information on the directions which can be used on certain layers. The use of the DFA aids in reducing the search space required for pattern string generation from regular expressions. It also allows a string of symbols to be accepted or not accepted for allocating or deallocating wire segments as well as calculating a total cost for all segments allocated.

3.3 Pattern String Generation

Deterministic Finite Automata are useful primarily as accepters of strings of symbols (Σ^*). Therefore, some method is required to generate these strings. The string generation is performed by the traversal of a tree corresponding to the regular expression. The DFA helps to limit the search of this tree by providing information which prunes the tree. Figure 3.3 demonstrates the tree required for the search space defined in Figure 3.2 for

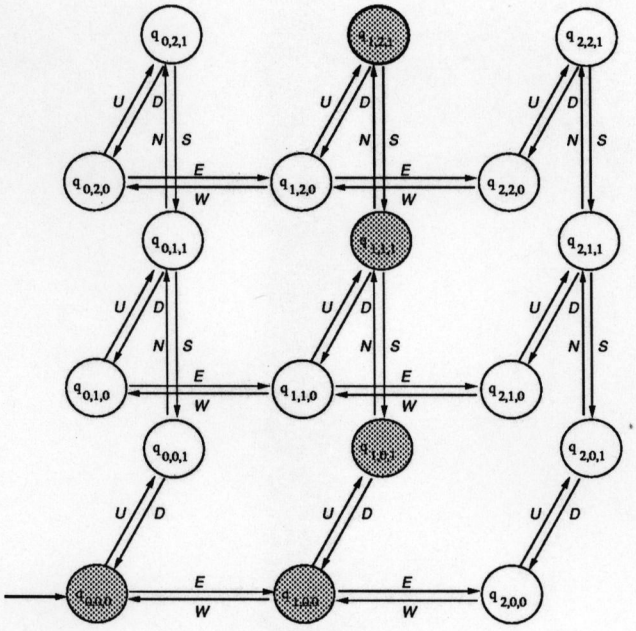

Figure 3: The DFA showing the path from Figure 3.3.

the pattern $\boldsymbol{E}^{<}(\boldsymbol{U}+\boldsymbol{D})^{<}\boldsymbol{N}^{<}$ starting at $q_{0,0,0}$ and trying to reach $q_{1,2,1}$. The result of the path on the DFA is shown in Figure 3.3.

A pruning method is very helpful to limit the search space. The best pruning method is to define the regular expression to be as descriptive as possible. However, sometimes this is not desired if a number of patterns or variations on a pattern need to be compared for a best solution.

Another method of pruning is the branch and bound method[7]. Finding a method to calculate a reasonable lower bound for each node of the current configuration is difficult, the actual cost can be easily calculated using the *cost* function defined previously. The lower bound on the cost can be obtained by finding the smallest distance possible along the available routing directions if none of the wire segments along that path were allocated. This can provide a method of deciding which of a number of paths should be attempted.

In addition to these methods, the states in the Deterministic Finite Automaton are also used to prune the tree. As the tree is traversed, each node visited will correspond to a state in the DFA. The tree can be pruned if the DFA does not have an arc which will permit the corresponding traversal.

Finally, there is one additional method of pruning where a number of solutions exist and a global physical constraint is known. As the traversal of the tree proceeds, the cost of the wires required to implement the symbols is accumulated. If this cost exceeds some desired maximum along any branch, it can be pruned since any further allocation along that path will only increase the cost.

As an example of this tree, assume that the first net must start at $q_{0,0,0}$ and reach $q_{1,2,0}$ with the pattern specified by $\boldsymbol{E}^{<}(\boldsymbol{U}+\boldsymbol{D})^{<}\boldsymbol{N}^{<}$. There is only one solution to this pattern in this environment demonstrated by Figure 3.3. This is allocated in the routing area (Figure 3.3) and forms an obstacle to the second net. The obstacle is handled by the DFA described above. This obstacle essentially eliminates the arcs which have been implemented as wires to make the first connection as well as arcs which are directed away from or towards any nodes along the path of the first route.

The second net now starts at $q_{0,0,1}$ and tries to connect to $q_{2,0,1}$ with the pattern $\boldsymbol{N}^{<}(\boldsymbol{U}+\boldsymbol{D})^{<}\boldsymbol{E}^{<}(\boldsymbol{U}+\boldsymbol{D})^{<}\boldsymbol{S}^{<}$. If the first net had not already been allocated then the solution would be $\boldsymbol{DEEU}$ however, this is not possible

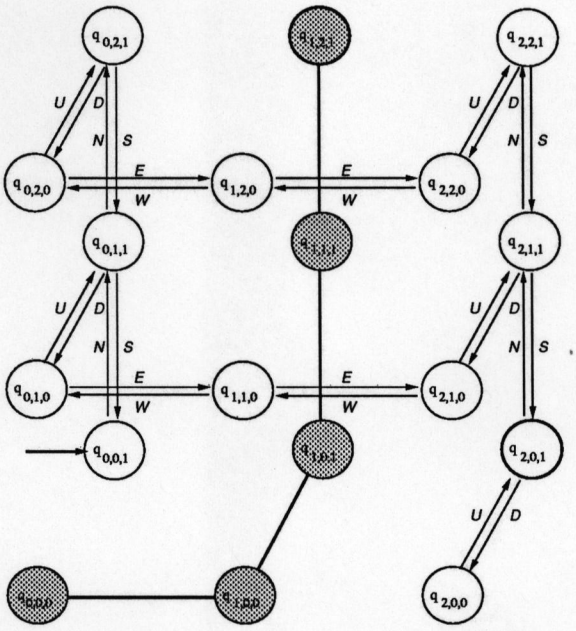

Figure 4: Routing another net from $q_{0,0,1}$ to $q_{2,0,1}$. Arcs which cannot be used by this net have been removed.

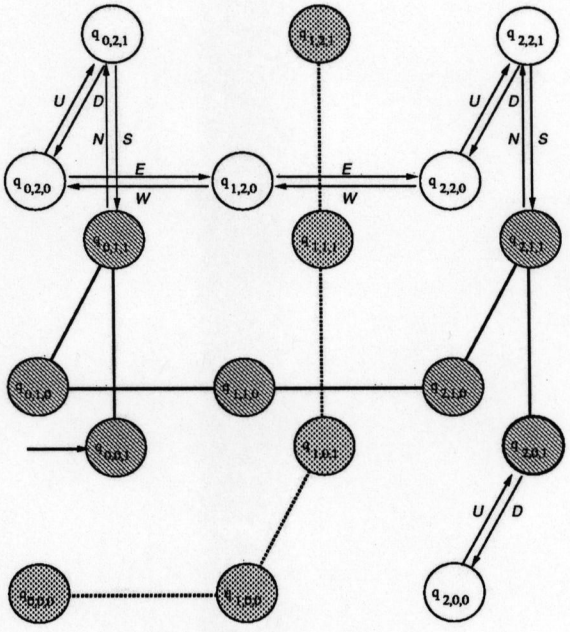

Figure 5: Routing area containing two routed nets.

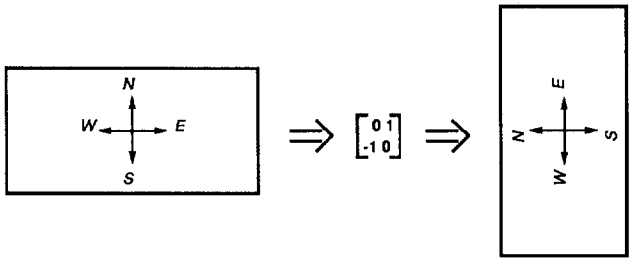

Figure 6: Transforming the desired routing area to the real area.

since the arc from $q_{0,0,0}$ to $q_{1,0,0}$ is not available. Therefore, the resulting pattern is $\boldsymbol{NDEEUS}$ shown in Figure 3.3.

4 Abstraction

There are many benefits to using patterns and regular expressions to describe them. One of these benefits is the abstraction to patterns of what is actually occurring physically. Usually, routing algorithms have a desired orientation of a routing area. This desired orientation may just require a simple rotation from the real orientation. Using patterns, the algorithm can be written and any transformations which are required can take place to map the symbols to the Deterministic Finite Automaton describing the routing area.

There are two ways of providing this transformation. One method essentially transforms the points which are to be connected, transforming the DFA and the *owner* function, performing the route and then transforming the wire segments to their proper position. This method is efficient unless obstacles exist in the routing area. If there are obstacles, then the DFA is difficult to transform. This difficulty is caused by the *owner* function since it must transform all of the obstacles. The exact dimensions of these obstacles in terms of the wire segments which they void is unknown to the router, except by trying to allocate them. Therefore these obstacles cannot be easily transformed.

A better method is to incorporate the transformation in the symbols. For example, if $\boldsymbol{N}$ is defined as a unit vector in the direction $(0,1,0)$ then a 90 degree rotation of this would produce a unit vector $(-1,0,0)$. Figure 4 shows the transformation of the entire set of directions. Since this is only a two-dimensional rotation, the symbols for changing layers are not transformed. The identical patterns and regular expressions using them can then be used.

Another useful feature which the patterns abstract is the actual dimensions of the segment. Each symbol represents a unit length wire segment. However, the units can be different for different layers along the same direction or the same layer in different directions. For example, the $\boldsymbol{N}$ symbol on one layer may have a different length then a $\boldsymbol{N}$ on another layer. In addition the $\boldsymbol{E}$ symbol may represent yet another unit length on each layer in a different direction. There is also no requirement that the opposite direction on the same layer have the same unit length.

5 Examples in Algorithms

Three algorithms are chosen here to illustrate how simply patterns can be used. These algorithms provide a small cross-section of different types of algorithms.

5.1 "Greedy" Channel Router[5]

This algorithm is designed to route a channel - a routing area with all of the connections on the top and bottom of the bounding area. The algorithm is not described with patterns explicitly, however it can be easily converted.

Starting from the left for each column in turn until the rightmost column is reached, do the following:

1. Bring the connections on the top and the bottom of the channel into the routing area. The connection should be brought to the closest track which ensures that the track which the new point is on can be extended to the right. If an available track cannot be found then insert a track. The pattern used here is simple. The directions up or down, depending on whether the connection is on the top or the bottom, should be minimized. The pattern must check also that a segment can be placed in a track. The pattern $(N + S)^{<}(U + D)^{<}E$ provides this. This pattern can then be intersected with the pattern $(N + S)^{<}(U + D)^{<}$ to remove from the initial string, the symbol which verifies that the track is available.

2. Collapse any nets which may be occupying two different rows in the same column by connecting them if possible. The two points which are to be collapsed will be provided to the pattern matcher as it tries to maximize the number of symbols between these two points. The pattern used is $(U + D)^{<}(N + S)^{>}(U + D)^{<}$. The layer changing symbols are required in case the two points are on different layers.

3. Shrink the number of rows between connections on the same net in the same column which could not be collapsed. The idea is to take the connections as close to each other as possible since they could not be connected. The pattern $(N + S)^{>}(U + D)^{<}E$ with the correct bounding box will perform this.

4. Make preference jogs which attempt to get the wire to a row closer to the next point on the net. The pattern required for preference jogs is the same as for shrinking nets since it requires essentially the same functionality.

5. Extend each wire from this column into the next one to the right. This requires simply adding a single segment. The pattern $(U + D)^{<}E$ will change layers if the one it is on does not permit wires in this direction.

5.2 BEAVER[3]

The BEAVER router uses five patterns to interconnect points on the same net in a switchbox. Each connection on the boundary of the switchbox is given some length of the row or column perpendicular to its side. This is unusable by other patterns until a connection is made thereby freeing up some of this segment. The amount of the row or column wire which is reserved for a connection starts at 50%. This number is decreased until the entire route is complete. Figure 5.2 demonstrates this algorithm.

Start the factor at 50% and keep decreasing until finished. For each value of the factor, perform the following steps for each net:

1. Allocate wire segments for all nets except for the one being routed. The bounding box used for pattern generation is based on the current factor. The patterns are just single direction connections and therefore can be described by $(N^{>} + S^{>} + E^{>} + W^{>})$. Save these patterns to permit them to be freed later.

2. For the current net, try to connect with corner patterns as many points as possible to other points or wire segments on the net. The description for any corner is
$((N + S)^{<}(U + D)^{<}(E + W)^{<}) + ((N + S)^{<}(U + D)^{<}(E + W)^{<})$.

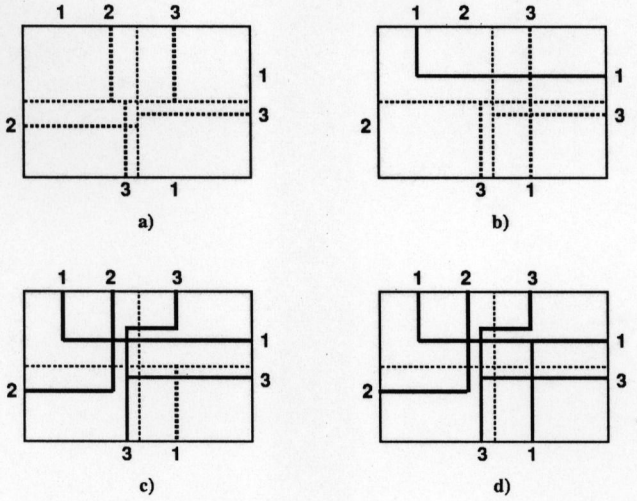

Figure 7: BEAVER algorithm example. a) allocate half of each row or column which any point not in net 1 is on. b) allocate a corner connections between two net 1 connections, net 3 precludes the other point being connected. c) allocate net 2 and net 3, first with a corner then a stairstep. d) finally complete net 1 since net 3 has now been routed.

3. For the current net, try to connect any points not connected with corners using straight, dogleg, stairstep or horseshoe patterns. These patterns can all be combined into one regular expression. This expression is
$((N+S)^<(U+D)^<(E+W)^<(U+D)^<(N+S)^<)+((E+W)^<(U+D)^<(N+S)^<(U+D)^<(E+W)^<)$.
4. Free all the wire segments which were used to reserve columns and tracks for unconnected points.

5.3 The Lee Algorithm[8]

This algorithm is one of the oldest and also one of the most widely used. It is usually incorporated inside more complex algorithms to provide optimal point-to-point connections. The algorithm starts at a point and propagates outward one unit in each direction if possible. A cost is associated with each of these new locations and is the cost of the initial location plus the additional cost required to get to the new location. This is continued until the destination point is reached. By backtracking along the path, always taking the lowest cost direction, the optimal path is assured.

This algorithm is defined by the simple pattern $(N^< + S^< + E^< + W^< + U^< + D^<)^<$. This demonstrates the need for good pattern tree pruning, otherwise the search space is very large.

6 Conclusions

This powerful pattern description and matching system can be used for many existing algorithms. These algorithms are not restricted to methods which already are described with patterns. The use of these patterns will produce more flexible algorithms which can take advantage of the actual environment of the layout being performed.

In addition, once these algorithms are implemented with a common basis in patterns, some of the similarities and differences will become apparent. These comparisons will enable algorithms to be analyzed to possibly discover why some methods perform better than others in certain situations.

Apart from comparing different algorithms, the effects of using different patterns within the same algorithm framework can be studied. By keeping the control and strategy mechanisms constant and changing the patterns, perhaps better results in some situations can be achieved. The patterns can then be specified by the user for a specific route while maintaining the goals and functionality of the original algorithm. This will permit algorithms to be easily tuned to different examples or different environments.

References

[1] J. Soukup and S. Fournier, "Pattern router," in *International Symposium on Circuits and Systems*, pp. 486–489, 1979.

[2] R. Joobbani and D. P. Siewiorek, "Weaver: A knowledge-based routing expert," *IEEE Design and Test of Computers*, vol. 3, pp. 12–23, February 1986.

[3] J. P. Cohoon and P. L. Heck, "Beaver: A computational-geometry-based tool for switchbox routing," *IEEE Transactions on Computer-Aided Design*, vol. 7, pp. 684–697, June 1988.

[4] M. Burstein and R. Pelavin, "Hierarchical channel router," in *20th Design Automation Conference*, pp. 591–597, 1983.

[5] R. L. Rivest and C. M. Fiduccia, "A "greedy" channel router," in *19th Design Automation Conference*, pp. 418–424, 1982.

[6] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley Publishing Company, 1979.

[7] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms.* Computer Science Press, 1978.

[8] C. Y. Lee, "An algorithm for path connections and its applications," *IRE Transactions on Electronic Computers*, vol. 10, pp. 346–365, September 1961.