SetA* Applied to Channel Routing

Rune M. Jensen Randal E. Bryant
Manuela M. Veloso

September 2002
CMU-CS-02-172

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

This report describes an application of the SetA* algorithm to VLSI channel routing. We consider an extended form of the classical routing problem where pins of nets can occur anywhere within the channel. The derived algorithm can use general cost functions and heuristics given that the total routing cost equals the sum of routing costs for each column. For this class of problems we show a graph-based approach for deriving an admissible heuristic for any cost function. The approach is evaluated on a subset of classical routing problems generated from ISCAS-84. We obtain results similar to the most efficient current approaches.

This work was supported in part by the Danish Research Agency and the United States Air Force under Grants Nos F30602-00-2-0549 and F30602-98-2-0135. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force, or the US Government.

Keywords: Channel Routing, OBDD-based search, Heuristic search

1 Introduction

Channel routing is a fundamental subtask in the layout process of VLSI-design. It is an NP-complete problem which makes exact solutions hard to produce. Various heuristics have been applied, but in practice solutions are often far from optimal. SAT algorithms based on the *reduced Ordered Binary Decision Diagram* (OBDD, [1]) have been applied to either extract optimal solutions within small critical areas of the channel [5] or investigate routing of long channels with a restricted number of tracks [7]. Previous results have been limited. The approach described in [5] is only feasible for extremely small and restricted problems and [7] fails to solve classical routing problems with more than 13 tracks.

In this report, we introduce a new approach for deriving optimal solutions to an extended class of routing problems where pins are allowed within the channel. This opens up for a more general use of the routing algorithm to over-the-cell routing, intra-cell routing, or more interestingly, as a leaf operation in global routing. Similar to [7], we consider a search starting at the left-most column in the channel. In contrast to [7], however, we apply SetA* [3] for expanding only the most promising partial routings. SetA* combines A* [4] and symbolic search. Similar to A*, it focuses the search on a subset of the possible partial routings with lowest expected cost. SetA* performs a complete search and finds optimal solutions given an *admissible* heuristic (that is, a heuristic underestimating the remaining cost). Given that the total routing cost is a sum of routing costs for each column, we show a general approach for deriving an admissible heuristic function. As an example, we consider the sum of vias as cost function.

We evaluate the approach on instances from two problem classes: extended and classical channel routing. Surprisingly, the results show that the search fringe for channel routing problems, when applying blind search, only grows moderately. This is in sharp contrast to hard AI search and planning problems where the search fringe typically blow-up [3]. The OBDDs representing only most promising partial routings seem to grow as fast as the blind search fringe, and consequently no particular complexity reduction is obtained by applying SetA*. The performance of our approach is similar to [7].

The remainder of this report is organized as follows. In Section 2, we describe previous work. We then, in Section 3, define the extended routing problem and our approach for deriving admissible heuristics. In Section 4, the application of SetA* is described. Experimental results are presented in Section5, and finally in Section 6, we conclude and discuss directions for future work.

2 Previous Work

The first application of OBDDs for channel routing was presented in [5]. In this paper a general channel routing problem was considered with an arbitrary number of layers and pins at any position. Every grid location is represented by an MDD variable [6] indicating which net occupies the position. In this way, a single MDD representing all possible routings can be build. However, due to the extensive use of variables, the approach is only feasible for extremely small and restricted problems.

A less general, but more efficient encoding, is described in [7]. In this encoding, a single horizontal and single vertical layer is assumed. Nets shift tracks by doglegs in the vertical layer. Wires are not allowed to fork and can only run from left to right in the channel. Several implementations were experimentally evaluated using problems derived from the ISCAS-85 circuits. An MDD implementation had best performance and solved up to 13 tracks.

In [8], a similar approach was considered, this time using an MTBDD to enable optimal solutions to be generated. The approach is feasible up to 15 tracks, but on a different problem set than [7]. For both [8] and [7] the quality of the symbolic implementation is unclear. The work in [8] apparently relied on some sort of transition relation encoding, while [7] applied specialized MDD operations. The work presented in this report uses a classical conjunctive partitioning of a transition relation for each column, mapping a set of in-track assignments to their possible out-track assignments. In contrast to [8], we consider a complete search approach that can focus on the most promising partial routings without pruning any reachable routings.

3 Channel Routing as State Space Search

The channels we will consider, have a single lower layer with horizontal tracks and a single upper layer with vertical tracks. A track can hold at most one wire. Wires are not allowed to fork and are assumed only to run from left to right. A wire with a particular in and out track over a column is assumed to have a unique cost. The total cost of a routing is the sum of the column costs.

Definition: *Optimal channel routing problem*

Input: A channel with t tracks and c columns, A list of nets each defined by a set pins given by grid positions, and a cost function.

Output: A minimum assignment of nets to tracks for each column such that all nets are connected.

Consider the example shown in Figure 1 (a). This problem has three tracks, four columns, and two nets. Two solutions are shown in Figure 1 (b) and (c). In order to apply SetA* for

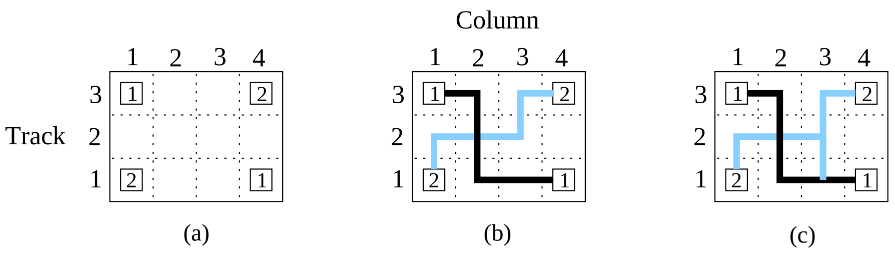

Figure 1: An example problem (a) and two solutions (b) and (c).

this problem, we need to restate it as a state space search problem. Given that columns are routed left to right (from 1 to m), the initial states of this search is the set of valid track assignments after the first column, and the goal states is the set of valid track assignments before column m. Let a *state* be a track assignment s_i of active nets between column i and $i+1$. A partial routing of columns 1 to k is then a *path* $s_1 \cdots s_k$. Thus, a solution to a routing problem with m columns is a path $s_1 \cdots s_{m-1}$.

We can now define the cost function $g : \mathit{path} \rightarrow \aleph$ and the heuristic function $h : \mathit{state} \rightarrow \aleph$. The cost of a partial routing $p = s_1 \cdots\ s_k$ is

$$g(p) = \sum_{i=1}^{k} c_i(s_{i-1}, s_i),$$

where $c_i(x, y)$ is the cost of routing column i with in-tracks x and out-tracks y. The initial state s_0 is an arbitrary track assignment since no nets enter column one. For a track assignment s_i the heuristic function is

$$h(s_i) = \sum_{n \in A(i)} e(n) + \sum_{n \in U(i)} e(n),$$

where $A(i)$ is the set of active nets in column i and $U(i)$ is the set of unrouted nets located after column i. The function $e(n)$ estimates of the remaining cost of routing net n.

In order to define $g(p)$ and $h(s)$, we introduce the concept of a *cost graph*. The cost graph of a net covering columns 1 to k in a channel with n tracks is a layered graph where vertices form a $n \times k - 1$ grid with a beginning and end vertex. Each layer is fully connected (see Figure 2). Each vertex in layer i denotes a track assignment of the net between column i

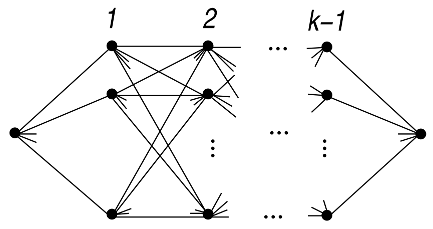

Figure 2: The structure of the cost graph.

and $i+1$. An edge between two vertices denotes a routing of the net over the associated column. The edge cost corresponds to the cost of this routing.

The cost graph can be used to efficiently find a minimum routing cost of the net by performing a Dijkstra search from the end vertex. However, notice that this approach assumes that the cost of routing a net over a column is uniquely defined by the in and out track. This may not always be the case as depicted in Figure 1 (b) and (c), where the routing of net 2 over column 3 is different but with the same in and out-tracks. For our purpose, unique routings are obtained by restricting the vertical extension of a routing to the minimum range occupied by ports of the net on the column and the in and out tracks. Thus, in the example, only routing (b) is valid.

An admissible heuristic is easy to derive from the cost graph. For a search state, where the net is active, its track assignment corresponds to a vertex in the cost graph. The minimum remaining cost of routing the net is the value assigned to the vertex by Dijkstra's algorithm. This is a lower bound, since it does not consider interaction with other nets.

In the remainder of this note, the cost of a column routing equals the number of vias in the routing. A via is a connection between the horizontal and vertical layer. We assume vias on ports are free. The heuristic function will be defined from the cost graph of the cost function. As an example, consider the net shown in Figure 3. The cost graph of the net is

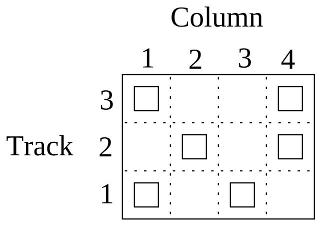

Figure 3: A net example.

shown in Figure 4. After running Dijkstra's algorithm from the right most vertex, we get

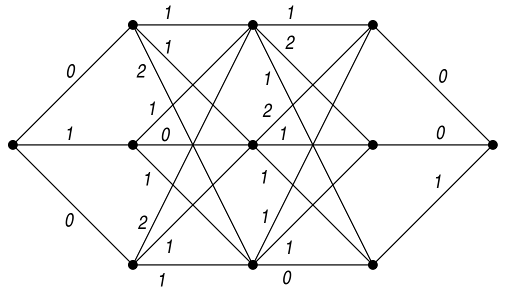

Figure 4: Cost graph of the example net.

the vertex values shown in Figure 5.

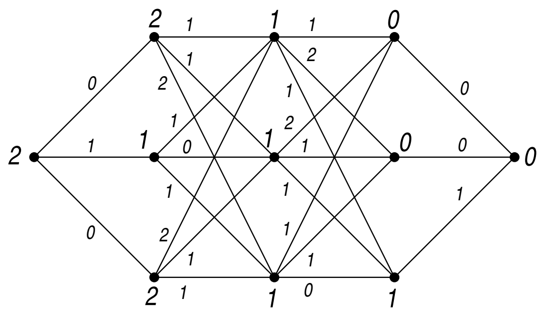

Figure 5: Vertex values of Dijkstra's algorithm.

4 Boolean Encoding and SetA*

SetA* encodes states and explores the state space implicitly via OBDDs. For details, we refer to [3]. A state equals a Boolean vector. A set of states V is represented by its *characteristic function* $V(\vec{x})$ where $\vec{x}$ is a vector of Boolean variables. The transition relation of the state space is given by a Boolean function $T(\vec{x}, \vec{x}')$. $T(\vec{x}, \vec{x}')$ is true iff there is a transition from the state denoted by $\vec{x}$ to the state denoted by $\vec{x}'$. The states that can be reached in one step from a set of states V are defined by the *Image*

$$\mathit{Image} = (\exists \vec{x} \,.\, V(\vec{x}) \wedge T(\vec{x}, \vec{x}'))[\vec{x}/\vec{x}'].$$

The OBDD-encoding of a track assignment state s_i is a binary encoding of the track numbers of the nets alive between column i and $i+1$. Given a transition relation T_{i+1} defining the possible routings over column $i+1$, we get

$$s_{i+1}(\vec{x}) = (\exists \vec{x} \,.\, T_{i+1}(\vec{x}, \vec{x}') \wedge s_i(\vec{x}))[\vec{x}/\vec{x}'].$$

4.1 Implementation I

Two transition relation implementations were made. In the first, T_i is defined as a conjunction of constraints for each unordered pair of live nets (i, j). The expression $\mathit{constraint}(i, j)$ states that there is no conflicts between the routing of net i and j. It can be decomposed into a horizontal and vertical part

$$\mathit{constraint}(i, j) = \mathit{horizontal}(i, j) \wedge \mathit{vertical}(i, j).$$

The horizontal constraint states that

1. i cannot touch a port of j
2. j cannot touch a port of i
3. All i's out-tracks are different from j's in and out-tracks
4. All j's out-tracks are different from i's in and out-tracks
5. i and j's out-tracks are different

Notice that this definition ignores special cases for nets being born and dying on the column. These are of course considered in the SetA* implementation. It also should be noted that we do not need to state that i and j's in-tracks are different. This will hold by induction. The vertical constraint is more complicated. Its abstract definition is

$$\mathit{dogleg}(i) \wedge \mathit{dogleg}(j) \Rightarrow \mathit{disjoint}(\mathit{dogleg}_i, \mathit{dogleg}_j)$$

Again we ignore details. The constraint expression is reflexive and only depends on the variables used to define the in and out-tracks of net i and j. This allows for a conjunctive partitioning [2] of T_i where the first partition contains the constraints between the first live net and all subsequent live nets. The second partition contains the constraints between the second live net and all subsequent live nets etc..

When routing a column, SetA* needs to be able to extract transitions with a particular change in g and h-value. We obtain this simply by conjoining the transition relation of a column with an OBDD representing such transitions. This OBDD is generated with a special OBDD function that in a topdown fashion tracks all possible transitions and only picks those corresponding to a particular change. Separate OBDDs $\delta(g, h)$ are computed using this function. Given r live nets on column i, the resulting image computation finding s_{i+1} for $\delta(g, h)$ is

$$s_{i+1}(g,h) = (\exists \vec{x^r_{i+1}} \,.\, T^r_{i+1} \wedge \cdots (\exists \vec{x^1_{i+1}} \,.\, T^1_{i+1} \wedge \delta(g,h) \wedge s_i) \cdots)[\vec{x}/\vec{x}\,']$$

where T^j_i is the j'th partition of the transition relation of column i and $\vec{x}^j_i$ is the variables this partition depends on.

4.2 Implementation II

In a second implementation, the image computation was broken up into a number of iterations. In each iteration, a single net is routed over the column without conflicting with any previous nets routed over the column in earlier iterations. This implementation allows the change in g and h-value to be computed along the way according to the g and h change of each net. This implementation assumes a classical routing problem with ports only on the top and bottom of the channel. Thus, for this implementation we do not use two tracks for these ports.

The search performed by SetA* proceeds in the usual way. First the transition relation of each column is computed. Then the search queue is initialized with the possible track assignments of live nets after column one. In each iteration, the search node with the lowest f-value ($f = (1-w)g + wh$) is popped from the search queue. The data structure of the search queue nodes has been extended such that it stores which column the set of track assignments belong to. This information is used to apply the right transition relation. Children corresponding to particular g and h-value changes are generated and reinserted in the queue. Notice that the SetA* implementation of channel routing handles general transition costs (the original version of SetA* assumed unit edge costs).

5 Experimental Evaluation

Our SetA* channel router has been evaluated on number of problem sets. In this report, we describe two of these experiments. The first has been generated from the ISPD-98 global routing benchmarks placed with Timberwolf. These channel routing problems are in the extended form with ports within the channel. The second problem set is a subset of the ISCAS-85 benchmarks used in [7]. These problems are classical channel routing problems with ports only on the bottom and top of the channel.

All experiments were carried out on a Linux kernel 2.4.16, 500 MHz Pentium III with 512KB L2 cache and 512MB RAM. The time limit (TIME) was 600 seconds and the memory limit (MEM) was 200 MB. The upper bound of the size of OBDDs in the search queue, the number of allocated OBDD nodes and the size of the operator cache of the OBDD package was hand-tuned for best performance. The weight of SetA* was set to 0.5 to generate optimal solutions. The measured data are:

c	:	Number of columns
t	:	Number of tracks
n	:	Number of nets
d	:	Density (average number of active nets per track)
T_r	:	Time (sec) for building transition relation in seconds
T	:	Total CPU time (sec)
C	:	Cost of solution
#it	:	Number of iterations
$\|Q\|_{max}$	:	Max size of queue during search
u	:	Upper bound of OBDDs in the search queue

5.1 Implementation I

For the ISPD-98 experiment, we generated problems with increasing number of tracks. The results are shown in Table 1. For all problems the upper bound u was set to 500. The

c	t	n	d	T_r	T	C	#it	$\|Q\|_{max}$
40	10	25	0.23	1.7	2.5	10	28	70
40	10	30	0.33	2.6	4.4	6	45	83
40	10	35	0.28	1.5	2.1	8	20	54
20	15	25	0.23	1.7	2.5	10	28	70
20	15	30	0.33	2.6	4.4	6	45	83
20	15	35	0.28	1.5	2.1	8	20	54
20	20	42	0.26	5.6	10.4	13	163	84
20	20	40	0.26	5.1	8.5	12	95	71
20	20	41	0.28	8.3	14.5	15	67	111
20	25	35	0.17	1.7	4.7	5	20	55
20	25	40	0.21	28.0	46.1	10	158	102
20	30	30	0.17	8.5	11.5	10	32	125

Table 1: Results of the ISPD-98 experiment.

performance of SetA* is promising on this problem set. However, an evaluation of the algorithm can not be based on these results since the locality of nets may be crucial for the good performance, and we have no previous results to compare with.

The ISCAS-85 problems were obtained from Kolja Sulimma and were used for the experimental evaluation of the channel router algorithm described in [7]. It is unclear though, how large a fraction these problems constitute of the benchmark suit used in [7]. The problems are classical channel routing problems with no predefined number of tracks. For each problem, we generated a number of subproblems with increasing number of tracks. The track numbers reported below include two tracks used to hold the ports. There are two problem sets *Add* and *C432*. Table 2 shows the results of the Add set. A dash (-) in the cost entry C indicates that no routing exists. The results for the C432 set are shown in Table 3. The

c	t	n	d	T_r	T	C	#it	$\lvert Q\rvert_{max}$	u
38	4	10	0.48	0.1	0.3	19	96	14	∞
47	6	27	0.49	0.6	1.5	-	214	27	∞
41	4	12	0.53	0.1	0.4	20	153	22	∞
46	8	20	0.47	1.3	19.3	47	408	93	∞
25	5	6	0.32	0.1	0.2	8	45	15	∞

Table 2: Results of the ISCAS-85 Add set.

c	t	n	d	T_r	T	C	#it	$\lvert Q\rvert_{max}$	u
83	4	33	0.47	0.2	0.3	-	58	16	∞
83	5	33	0.38	0.4	0.8	29	231	30	∞
89	11	58	0.56	-	TIME	-	-	-	200
101	9	57	0.60	22.9	57.8	-	75	-	200
99	8	58	0.56	4.0	113.3	-	200	61	200
97	10	63	-	-	TIME	-	-	-	-
101	7	53	0.64	4.1	10.4	-	170	43	200
101	8	53	0.56	-	TIME	-	-	-	-
95	9	48	0.58	-	TIME	-	-	-	-
84	5	23	0.40	0.5	2.7	-	30	6	∞
84	6	23	0.33	0.6	3.2	19	250	54	∞

Table 3: Results of ISCAS-85 C432 set.

performance of SetA* on the ISCAS-85 problems is similar to the algorithm described in [7] and [8]. As [8], we observe very large intermediate OBDDs in the image computation. In addition, the OBDD representing each partition is about twice the size of the sum of the OBDDs representing the constraints they are a conjunction of. Variable reordering can reduce this size, but since a set of OBDD variables is used to represent different nets, the optimal variable ordering changes from column to column. Another interesting observation is that the search frontier of blind OBDD-based search grows only moderately. Focusing on the most promising track assignments does not seem to reduce the size of the frontier OBDDs. This is often the case for hard AI search and planning problems. Consequently applying SetA* to this problem does not scale the size of problems that can be handled. Performing best-first search by changing w from 0.5 to 1.0 does not reduce the search complexity, but results in solutions with substantially lower quality.

5.2 Implementation II

Since this implementation only handles classical routing problems the ISPD-98 problems are not applicable. For the ISCAS Add and C432 problems we get the results shown in Table 4 and Table 5 (recall that the corresponding solution with implementation I normally needs more tracks)

c	t	n	d	T_r	T	C	#it	$\lvert Q\rvert_{max}$	u
38	3	10	0.59	0.1	0.2	23	40	0	∞
47	5	27	0.59	0.7	0.8	52	46	24	∞
41	3	12	0.68	0.1	0.2	27	42	0	∞
46	7	20	0.76	3.5	5.0	53	89	56	∞
25	4	6	0.33	0.0	0.1	11	30	0	∞

Table 4: Results of the ISCAS-85 Add set.

c	t	n	d	T_r	T	C	#it	$\lvert Q\rvert_{max}$	u
83	4	33	0.42	0.2	0.4	38	93	0	∞
89	11	58	0.56	107.8	MEM	-	-	-	500
101	9	57	0.60	61.48	286.1	121	113	135	500
99	8	58	0.56	13.5	34.0	117	448	59	∞
97	10	63	0.51	99.7	295.0	120	109	129	∞
101	7	53	0.64	11.5	15.7	102	101	90	∞
95	9	48	0.58	58.9	223.8	-	-	-	∞
95	10	48	0.58	-	TIME	-	-	-	∞
84	5	23	0.37	0.73	3.2	38	92	0	∞

Table 5: Results of ISCAS-85 C432 set.

The results of implementation II are better than for implementation I due to a less complex image computation. It should also be noticed that a lot fewer iterations seems to be necessary when not placing ports on the tracks since this may cause more frequent backtracking.

6 Conclusion and Outlook

We have presented an approach for applying SetA* to an extended form of channel routing problems. We show that SetA* can be extended to handle conjunctive partitions and general edge costs and develop an approach for computing an admissible heuristic for any edge cost function. The experimental evaluation of our approach shows a performance profile similar to the most efficient current algorithms. Implementation I and II are not more much more efficient than OBDD-based blind search. This indicates that the main source of complexity of channel routing problems is very large transition representations. It is unclear at this point, why channel routing problems are so different from the hard AI search problems solved successfully with SetA*. A deeper understanding of permutation problems with global change (like channel routing problems) is on our current research agenda.

References

[1] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 8:677–691, 1986.

[2] J.R. Burch, E.M. Clarke, and D.E. Long. Symbolic model checking with partitioned transition relations. In *International Conference on Very Large Scale Integration*, pages 49–58. North-Holland, 1991.

[3] R. M. Jensen, R. E. Bryant, and M. M. Veloso. SetA*: An efficient BDD-based heuristic search algorithm. In *Proceedings of 18th National Conference on Artificial Intelligence (AAAI'02)*, pages 668–673, 2002.

[4] J. Pearl. *Heuristics : Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley, 1984.

[5] F. Schmiedle, R. Drechsler, and B. Becker. Exact channel routing using symbolic representation. In *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS'1999)*, 1999.

[6] A. Srinivasan, T. Kam, S. Malik, and R. E. Brayton. Algorithms for discrete function manipulation. In *International Conference on CAD*, pages 92–95, 1990.

[7] K. Sulimma and K. Wolfgang. An exact algorithm for solving difficult detailed routing problems. In *Proceedings of the 2001 International Symposium on Physical Design*, pages 198–203, 2001.

[8] H. Xu. Structured routing with boolean satisfiability. CMU ECE Course Paper, 2001.