Exact Switchbox Routing with Search Space Reduction

Frank Schmiedle Daniel Unruh Bernd Becker
Institute for Computer Science, Albert-Ludwigs-University Freiburg
Am Flughafen 17, 79110 Freiburg, Germany
{schmiedl, unruh, becker}@informatik.uni-freiburg.de

ABSTRACT

We present an approach for exact switchbox routing that complements traditional routing techniques. It is particularly well suited for an application to dense problem instances and the completion of routing in subregions which turn out to be difficult for routing tools based on heuristic methods.
The exact router proposed uses symbolic methods i.e. MDDs (Multi-valued Decision Diagrams) for representation of the routing space. All possible solutions to the routing problem are represented by one single MDD and once this MDD is given, routability can be decided within constant time. To reduce the search space of possible routing solutions, so-called forced cells are computed. Finally, experimental results are given. They show the feasibility and the practicability of the approach.

1. INTRODUCTION

Physical VLSI design consists of several steps that have to be performed during the layout process of integrated circuits. One of the tasks is *Routing*. Connections between cells or blocks, resp., have to be generated under consideration of certain constraints: e.g. different nets are not allowed to intersect because such intersections produce short circuits. Additionally, wire lengths and the number of vias have to be kept as small as possible.
Two commonly-known routing issues are the *Channel Routing Problem (CRP)* and a generalization of this, the *Switchbox Routing Problem (SRP)*. Both of these problems are NP-complete [11] and therefore several heuristics have been developed to solve them. Many approaches (see e.g. [13; 5; 8; 3]) have been reported for the CRP as well as for the SRP which often is considered to be the most difficult routing task in the layout process of integrated circuits. Even though these methods often compute solutions that are nearly optimal, they also produce insufficient results in other cases, e.g. if the routing space is very dense and therefore finding valid solutions is particularly difficult. Then, an enormous effort may be spent on trying to complete routing where it is not even theoretically possible to complete and a large amount of computation time is wasted by this.

In such cases it would be helpful to have a technique that decides routability, i.e. whether there is a solution at all, and also it would be very desirable to get an exact solution[1] at least for the part of the routing space that makes the routing difficult. On the other hand, for large problem instances, the possibility for computation of all solutions within reasonable time bounds seems to be feasible only in special cases. Thus an exact router should be used only to route *parts* that are cut out of large problem instances.
As a result, interaction of conventional routing heuristics and an exact router is necessary and it can be expected that by such a cooperation, it becomes possible to solve problem instances for which both approaches fail when being applied standalone. A majority of the necessary connections can be generated by the heuristics and the exact router is applied to the smaller regions where the heuristics cannot find a solution. Hence it completes routing there.
Methods for *Exact Routing* have been proposed in [12; 6; 2]. All these approaches make restrictions on the routing model they use. In [12] and [6], exclusively routing of FPGAs is considered and in [2], only routing without doglegs is allowed. Recently, for the first time an exact method for channel routing that does not use such restrictions has been presented in [7]. Like in the other techniques mentioned above, routing constraints are more or less transformed into boolean satisfiability in this approach. MDDs [10] are used for function representation. The implementation uses a preliminary MDD package and demonstrates the feasibility of the method without performing any further optimization steps. MDDs in most cases grow too large and thus an exact solution is obtained for very small problem instances only.
The exact switchbox routing technique that is presented in this paper is a generalization of the method reported in [7]. It also is not restricted to routing without doglegs, and additionally, new features that speed up computation considerably and make the router more flexible have been included. Main features are integration of a state-of-the-art MDD package, arbitrary pin location and handling of obstacles, pre-routed nets and non-cuboidic routing regions. Thus, in contrast to [7] and all other symbolic approaches it becomes suitable for an interactive use with other heuristics in the way previously described.
Additionally, the exact router introduced here contains a preprocessing step that accomplishes certain pruning techniques thus eliminating paths that do not lead to valid solutions at an early stage of the routing process. By this, the search space of possible routing solutions can be reduced in many cases and as a result, the subsequent fixed point iteration is carried out much faster then. Experiments showed

[1]I.e. a set of *all* solutions.

that runtimes can be reduced by more than 95% on average in comparison to the previously presented approach for exact routing without restrictions concerning the routing model.
The paper is structured as follows: In Section 2 the basic concepts, i.e. MDDs and detailed routing are introduced. Additionally, it is shown how to represent routing spaces by symbolic methods and how to compute all solutions of the routing problem by fixed point iteration using this technique. Flexibility requirements to an exact router are discussed in Section 3 and the issue of Section 4 is search space reduction by pruning. In Section 5, experimental results are given to demonstrate the feasibility of the approach. Finally, a summary is given in Section 6.

2. PRELIMINARIES

2.1 Multi-valued Decision Diagrams

As well-known, each Boolean function $f : \mathbb{B}^n \mapsto \mathbb{B}$ can be represented by an *ordered binary decision diagram* (BDD) [1], i.e. a directed acyclic graph where a Shannon decomposition is carried out in each node.
BDDs can be extended to *Multi-valued decision diagrams* (MDDs) [10] representing functions $f : \{0, .., k-1\}^n \mapsto \{0, .., l-1\}$. Each internal node has k outgoing edges. It has been shown that the efficient operations known for BDDs can also be carried out on MDDs using a *case*-operator instead of the *ite*-operator [10]. Analogously to Chapter 4 of [4], we only consider functions $f : \{0, .., k-1\} \mapsto \mathbb{B}$ and therefore our MDDs like BDDs only contain the terminal nodes 0 and 1.

2.2 Detailed Routing

The task of detailed routing in layout design is to determine the exact location of wires connecting some pins belonging to different cells or blocks. The wires have to be located in a given routing space in a way that prevents the occurrence of short circuits, i.e. wires belonging to different nets are not allowed to intersect. Two additional optimization criteria are wire length and number of vias. Minimization of the values for both criteria improves performance of the resulting integrated circuit considerably. For modeling the routing space, we use a matrix M that represents a 3-dimensional Cartesian grid. In x-direction, there are so-called *columns*, in y-direction, different *tracks* are available and the *layers* used can be distinguished by their coordinates in z-direction. The number of columns, tracks and layers, resp., determines the size of the routing space M. Grid points may be connected to adjacent points only and as a result, a path in the routing space is given as a sequence of adjacent grid points.
A routing instance can be defined by the routing space M and a set $N = \{\{t_{11}, \ldots, t_{1k_1}\}, \ldots, \{t_{n1}, \ldots, t_{nk_n}\}\}$ of nets. The elements $\bar{N}$ of N describe the nets and the elements t_{ij} of $\bar{N}$ the pins or terminals of the net $\bar{N}$. A solution to the routing problem is a set C of paths. The first constraint that has to be fulfilled means that for two different pins of the same net there has to be a path in the solution that connects them. A second constraint makes sure that every path in the solution set starts and ends at a pin location. Finally, it has to be guaranteed that no grid point belongs to paths of two or more different nets at the same time, i.e. no short circuits occur.
In the CRP, all pins are located in the layer with the smallest index on two opposite boundaries of the grid. Note that, in contrast to various definitions of the CRP that allow insertion of additional tracks in order to guarantee routability at the expense of larger routing spaces, the channel height is fixed here.
Typical channel routers assume that there are two additional restrictions:

1. the *MANHATTAN* routing model is used. This means that the term “adjacency” is modified in a way that only neighbours in y-direction are considered to be adjacent in layers with odd indices and only neighbours in x-direction are considered to be adjacent in layers with even indices. I.e. only horizontal wires are allowed in even layers while in odd layers, only vertical connections are possible.

2. no doglegs are used, i.e. only one horizontal segment per net is generated.

As can easily be seen, the CRP is a special case of the detailed routing problem.
In the SRP, pins may be located at any grid point on the side boundaries of the routing space. The routing model used depends on the application and there are no restrictions concerning doglegs. The router presented in this paper allows arbitrary pin location throughout the whole routing space. In this sense, the routing model is more general than switchbox routing. The routing model may be chosen between the *MANHATTAN* and the (unrestricted) *FREE* model. Nevertheless, we call our router a switchbox router, too, to emphasize that it is a router with a very high flexibility embedded in a “switchbox environment”.

2.3 Symbolic Representation of Routing Space

For an exact router, it is crucial to provide the possibility of expressing the set of all solutions to the routing problem. This means that it must be able to describe alternative routing solutions completely and at the same time as efficiently as possible.
The router presented in this paper like those proposed in [2; 6; 7] uses Boolean functions to describe routing solutions. DDs are well-known to offer a good compromise between representation size and efficiency of manipulation. For these reasons, analogously to [7] we represent Boolean functions by MDDs.
For every grid point $(x, y, z) \in M$, a new MDD variable m_{xyz} is introduced. Let the netlist $N = \{\bar{N}_1, \ldots, \bar{N}_n\}$. The set of legal values for the MDD variables is $\{0, \ldots, n\}$ then. $m_{xyz} = 0$ means that (x, y, z) is not occupied by any net whereas that point contains a wire segment that belongs to net $\bar{N}_i$ iff $m_{xyz} = i$. In this way we are able to express each specified wiring situation in the channel by giving just the values of the set of MDD variables corresponding to the appropriate points of M.
Furthermore, a path $p = (p_1, \ldots, p_l)$ in M that belongs to some net $\bar{N}_j$ can be described as a Boolean function $\tilde{p}$ by (1):

$$\tilde{p} = (m_1 = j) \wedge \ldots \wedge (m_l = j) \tag{1}$$

where the m_i denote the MDD variables that correspond to the grid points p_i for $i \in \{1, \ldots, l\}$. In Figure 1, an example for an MDD representation of a path p in a given routing space is shown.

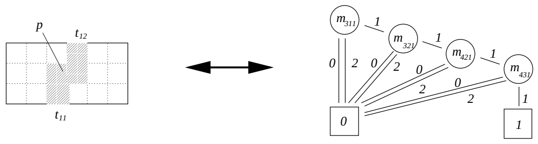


Figure 1: Example of a symbolic representation of a path

As can easily be seen, at the symbolic level the representation of a path can be obtained by carrying out conjunction operations. The same holds for a routing solution. Since such a solution $C = \{c_1, \ldots, c_{max}\}$ consists of a set of paths that exist in the routing space at the same time, the Boolean function $\tilde{c}$ for C can be computed according to (2):

$$\tilde{c} = (\tilde{c_1}) \wedge \ldots \wedge (\tilde{c}_{max}) \tag{2}$$

Given net $\bar{N}_j$, the functions $\tilde{c}_i$ can be computed by (1). Finally, a set of routing solutions obviously can be represented by an MDD using disjunction operations. Given the MDD representations of the solutions $s_1, \ldots, s_{max}$, the function $\tilde{s}$ corresponding to the set of these solutions is given by:

$$\tilde{s} = (\tilde{s_1}) \vee \ldots \vee (\tilde{s}_{max}) \tag{3}$$

The order in that MDDs for single paths are combined to the final result, i.e the MDD representing the set of all routing solutions, is slightly different in the method described in the following sections, but logic operations used are the same.
Conjunction and disjunction are basic operations in MDDs and thus the necessary MDD operations can be carried out efficiently. This is one reason why symbolic representation of routing space is well suited for this approach. Another advantage is that occurrences of short circuits are prevented elegantly. A short circuit is generated if the same grid point is used for paths belonging to different nets. At the symbolic level this means that two different values are assigned to one single MDD variable. This ambiguous assignment is not possible since the attempt in doing so is detected during MDD minimization and thus "solutions" with short circuits are conveniently avoided.

2.4 Exact Routing using Fixed Point Iteration

Computation of the Boolean function $\tilde{s}$ representing the set of all solutions to a given routing problem is the issue of this section. The approach is based on adaption of the ideas in [7] to the more general context in this paper. We present the main points. First of all, so-called *connectivity predicates* are computed by a fixed point iteration. After all predicates for a net have been determined, an MDD representing all possible connections for this net is generated. Finally, constraints for different nets resp. the corresponding MDDs are combined to one MDD representing the set of all possible solutions to the routing problem. In the following, the single computation steps are considered more detailed. For this, the following definition is necessary:

Definition 1. *For a given net $\bar{N}_i \in N$ and a selected grid point $t \in M$ a connectivity predicate $C_{i,t} : M \mapsto \mathbb{B}$ is introduced by*

$$C_{i,t}(p) :\Leftrightarrow p \textit{ is connected with } t \textit{ via net } \bar{N}_i \tag{4}$$

In the routing approach that is proposed here, t is chosen to be an arbitrary[2] pin of net $\bar{N}_i$ (notice that the pins of $\bar{N}_i$ are contained in any legal routing solutions).
$C_{i,t}(p)$ is represented by an MDD and for computation of this MDD, we make use of a close relationship (5) between the truth values of the predicates for adjacent points in M. This relationship is also illustrated in Figure 2. For $p = (x, y, z) \in M$, we have:

$$C_{i,t}(p) \;\Leftrightarrow\; (m_{xyz} = i) \wedge (\exists p' : C_{i,t}(p') \vee (p = t)) \tag{5}$$
$$\text{where } p' \text{ and } p \text{ are adjacent}$$

In order to generate MDDs for the connectivity predicates for all $p \in M$, we start an iteration with $C^0_{i,t}(t) = (t = i)$

[2]W.l.o.g. we choose pin t_{i1}.

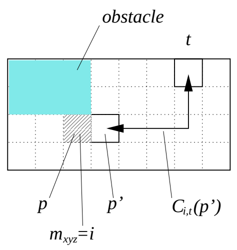


Figure 2: Relationship between connectivity predicates for adjacent points in M

and $C^0_{i,t}(p) = \mathit{false}$ for all $p \neq t$. In iteration j, we compute $C^j_{i,t}(p)$ for all $p = (x, y, z) \in M$ according to (6).

$$C^j_{i,t}(p) \;=\; C^{j-1}_{i,t}(p) \vee \Big(\bigvee_{p'} (C^{j-1}_{i,t}(p') \wedge (m_{xyz} = i))\Big) \tag{6}$$
$$\text{where } p' \text{ and } p \text{ are adjacent}$$

This iteration is continued until after iteration j_{max} the following holds:

$$\forall p \in M : C^{j_{max}}_{i,t}(p) = C^{j_{max}-1}_{i,t}(p)$$

This means that there has been no change during the last iteration or in other terms, that a fixed point has been reached. At that time, we have:

$$\forall p \in M : C^{j_{max}}_{i,t}(p) = C_{i,t}(p) \tag{7}$$

An MDD $\tilde{n}_i$ representing all possible sets of paths that connect net $\bar{N}_i$ can now easily be derived from the $C_{i,t}$'s that have been computed during the previous fixed point iteration. Since net $\bar{N}_i$ is completely connected iff from each pin t_{ij} a path to t_{i1} exists, $\tilde{n}_i$ can be generated according to (8):

$$\tilde{n}_i = \bigwedge_{j=2}^{|\bar{N}_i|} C_{i,t_{i1}}(t_{ij}) \tag{8}$$

Finally, the MDDs representing routing solutions for single nets can conveniently be combined by (9) resulting in an MDD $\tilde{s}$ that represents the set of all solutions to the routing problem.

$$\tilde{s} = \bigwedge_{\bar{N}_i \in N} \tilde{n}_i \tag{9}$$

Given the MDD $\tilde{s}$, routability for a problem instance I can be decided very efficiently since

$$I \textit{ is routable} \Leftrightarrow \tilde{s} \not\equiv 0 \tag{10}$$

Checking if an MDD is equal to 0 can be done within constant time.

3. FLEXIBILITY REQUIREMENTS

It has been mentioned in previous sections that the exact router presented in this paper is suitable to be used in combination with one or some of the various well-known heuristics that exist for the routing problem. The task of the exact router in this cooperative routing process is routing some small regions that are dense and therefore difficult to route since —if at all— only few solutions exist. If no solution exists, this can also be detected by the exact router and in those cases, one or some nets may be ripped up and rerouted by the heuristics.

The regions passed to the exact router are cut out of the three dimensional grid that forms the routing region of the original problem instance. Thus, the input of the exact router in such cases does not always have the typical characteristics of an input to a conventional routing problem as described in Section 2.2. Therefore, there are some additional requirements to the flexibility of the exact router concerning

1. location of pins
2. obstacle handling and shapes of routing regions
3. consideration of pre-routed nets
4. routing models

In the following sections, we subsequently describe in detail what kind of flexibility is required for the single items listed above and how the approach described in Section 2.4 can be adapted to provide this flexibility.

3.1 Arbitrary Pin Location

In the CRP, pin location is restricted to certain grid points. Only points that belong to one of two opposite boundaries of one layer may contain pins. The SRP is less restrictive, but pins are still allowed only at the side boundaries of the routing space. However, in the context described above, connections to pins of the original problem may be brought up to the small routing region passed to the exact router at any grid point on the surface of the boundaries of this region. The end points of these connections are the pins of the routing subtask that is solved by the exact method. Thus, pins may also be located not only in any arbitrary layer at the side boundaries of the routing region, but also in the inner part of the top or bottom layer. This is shown in Figure 3 a) for some pins. The input region for the exact routing subtask is illustrated as a grey cuboid and pins and wires are represented by balls and cylinders, resp.
While generating the set of solutions, the constraints for all paths of one net $\bar{N}_i$ are combined in one MDD by collecting the connectivity predicates for all the terminals except t_{i1} according to (8). If the pin locations are given by their 3 coordinates, the SRP can be solved exactly since $C_{i,t_{i1}}(p)$ has been already been computed for *all* $p \in M$ during fixed point iteration. Even more, also pins that are located inside the routing space can be handled in that case.

3.2 Handling of Obstacle and Non-Cuboidic Shapes of Routing Regions

In some cases, single points in the grid or even subregions of the routing space e.g. due to some technical constraints or fixed power and ground location are forbidden to be used for wires. These grid points or subregions are called *obstacles* and may be included arbitrarily in the routing space of problem instances given as input to the exact router presented here. Routing then has to be accomplished by finding paths

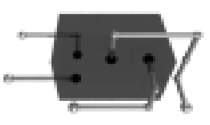
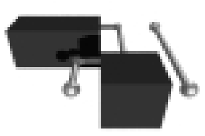
a) 
b) 

Figure 3: a) Possible pin locations for problems solved by the exact router b) Routing space containing obstacles

that lead around these obstacles. Figure 3 b) shows an example where the grey cuboids do not represent subareas of the routing space but cells that are obstacles.
Channels and switchboxes have always cuboidic shape. But subregions that are most suitable to be cut out of the routing region and solved exactly may also have different shapes. Therefore the exact router should be able to route regions of non-cuboidic shape. For modeling such regions, again obstacles can be used. A cuboidic routing space M that encloses the region $\bar{M}$ that is about to be routed (i.e. $\bar{M} \subseteq M$) is given as input to the exact router and all grid points $p \in M \backslash \bar{M}$ are declared to be obstacles. Figure 4 a) illustrates this. Here, different from Figure 3 b), $\bar{M}$ that contains all the pins corresponds to the grey area while the obstacles placed around this area are transparent.
Computation of the connectivity predicates is different if obstacles exist in the routing space. Still fixed point iteration can be used, but when for a grid point p constraints due to neighbours p' are added by (6), then only those neighbours may be considered that are not obstacles. For obstacle points, no connectivity predicates are computed. Furthermore, obstacle points reduce computation time since they cannot be part of a legal routing.

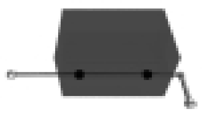
a)
b) 

Figure 4: a) A non-cuboidic routing space b) Net pre-routed in a subregion of the routing space

3.3 Handling of Pre-routed Nets

When dense regions are passed to the exact router, heuristics that have been used before may have already routed several nets in this region that have to remain unchanged, e.g. high-performance nets. The exact router then has to find connections for the remaining nets without using grid points that are already occupied by one of the pre-routed nets. Furthermore, the pre-routed wires must be included in the set of all routing solutions. A routing task containing a subregion with a pre-routed net is shown in Figure 4 b). To avoid usage of points that are occupied by pre-routed nets, the router considers such points as obstacles and hence they are not used for routing due to the neighbour selection that has been described in Section 3.2. For a pre-routed net $\bar{N}_i$, instead of computing the set of all possible connections, an MDD $\tilde{p}$ representing the given paths has to be generated. This can be done according to (1). The set of all routing solutions $\tilde{s}$ then can be computed by (9) using $\tilde{p}$ instead of $\tilde{n}_i$ for net $\bar{N}_i$.

3.4 Routing Models

The routing model for routing a subregion is determined by the routing model of the original problem. Besides the *FREE* model that is used by default the exact router also works for the *MANHATTAN* model that normally is used for solving the CRP. By this, interaction with typical channel routers is possible.
Again, the changes that have to be made during computation of routing solutions concern the generation of MDDs representing the connectivity predicates. For a grid point (x, y, z) belonging to a layer z where only vertical wires are allowed, $(x+1, y, z)$ and $(x-1, y, z)$ that are horizontally adjacent to p are not considered during computation

of (6). Analogously, the vertical neighbours $(x, y+1, z)$ and $(x, y-1, z)$ do not directly affect the connectivity predicate for (x, y, z) if only horizontal wires are allowed in z.

4. SEARCH SPACE REDUCTION FOR EXACT ROUTING

The most expensive step for the computation of an exact solution $\tilde{s}$ to an instance of a routing problem is the generation of the connectivity predicates for a net $\bar{N}_i$ by fixed point iteration. This iteration is expensive in computation time as well as in memory requirement. The connectivity predicates $C_{i,t}(p)$ are needed in (8) to compute $\tilde{n}_i$, i.e. the set of all wiring possibilities connecting $\bar{N}_i$ completely. In many cases, most of the possible paths for connecting $\bar{N}_i$ are not part of a valid routing solution since they prevent the successful generation of connections for some other net. Such paths are removed again when $\tilde{s}$ is computed by (9). At that time, constraints between different nets are considered by MDD reduction techniques and invalid solutions are eliminated.

However, a considerable amount of time and memory is spent in computation of such useless paths by the approach described in Section 2.4. In this section, a pruning method is introduced that especially in dense routing regions is able to detect points that are needed to place wire segments without which certain nets cannot be connected. Hence, it in advance eliminates many of the paths that make a complete routing of all nets impossible. This technique is applied *before* the fixed point iteration is carried out and thus time and memory consumption for computation of the connectivity predicates can be reduced. In special cases, exact solutions even can be generated only by this preprocessing and without any fixed point iteration.

For explanation how this reduction of the search space for exact routing proceeds, we need the following

DEFINITION 2. *A forced cell of net $\bar{N}_i$ is a point $p \in M$ that belongs to a path connecting $\bar{N}_i$ in any of the solutions represented by $\tilde{s}$. p is called a forced cell if it is a forced cell for one of the nets. $FC(i)$ denotes the set of all forced cells of $\bar{N}_i$ and FC the set of all forced cells in M.*
For any $p \in M$ that is forced cell of $\bar{N}_i$, $q \in M$ is called available for p iff q is adjacent to p and q is neither occupied by an obstacle nor by a pre-routed wire segment, a pin or a forced cell.

Obviously, pins are forced cells. But also grid points that are not pins may be forced cells and the task of pruning is to find as many of them as possible. Complexity of computation of the $C_{i,t}$ is reduced by forced cells for $\bar{N}_j, j \neq i$ in two ways: first of all, we do not need to compute $C_{i,t}(p)$ if $p \in FC(j)$ since p in that case cannot be connected with any point via $\bar{N}_i$ without causing a violation of routing constraints. Furthermore, when applying (6) in order to compute $C_{i,t}$, adjacent points q with $q \in FC(j)$ for some $i \neq j$ need not be considered. I.e. forced cells for different nets are treated like obstacles in this context. Thus, by forced cells, many MDD operations can be saved. Detection of forced cells is based on the following

LEMMA 1.

$$(p \in FC(i), |\bar{N}_i| > 1 \wedge |\{q \mid q \text{ is available for } p\}| = 1) \Rightarrow q \in FC(i)$$

Figure 5 shows an instance of a routing problem that contains 6 forced cells. Besides the pins, $(1,2,0)$ and $(3,2,0)$ are forced cells since $(1,2,0)$ is the only point available for $(1,3,0)$ and for $(3,1,0)$, only $(3,2,0)$ is available.

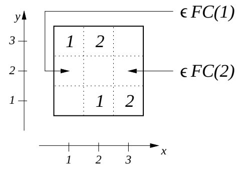


Figure 5: Routing example containing forced cells

The algorithm by which we detect forced cells is given in Figure 6. First, the FC's are initialized with the set of pins of the appropriate nets and afterwards, forced cells are added iteratively. In iteration j, all the forced cells that have been detected until iteration $j-1$ are considered and if exactly one point q is available for one of them, q is detected to be a forced cell, too. This forced cell detection can also be

```
Forced Cell Detection:
for all nets $\bar{N}_i$
    $j = 0$;
    $FC^{-1}(i) = \emptyset$;
    $FC^{0}(i) = \{t_{ij} \mid 1 \leq j \leq |\bar{N}_i|\}$;
while $\cup_{i=1}^{n}(FC^{j}(i) \backslash FC^{j-1}(i)) \neq \emptyset$
    $j{+}{+}$;
    for all nets $\bar{N}_i$   $(|\bar{N}_i| \neq 1)$
        $FC^{j}(i) = FC^{j-1}(i)$;
        for all $p \in FC^{j-1}(i)$
            $A = \{q \mid q$ is available for $p\}$;
            if $|A| = 0$
                return no_solution_possible;
            else if $|A| = 1$
                    $FC^{j}(i) = FC^{j}(i) \cup A$;
```

Figure 6: Algorithm for detection of forced cells

viewed as a movement of pins in the routing space. The points the pins visit during this movement may not be considered during the following fixed point iteration that now has to connect pins with modified locations. Afterwards, the constraints of the forced cells have to be added to the routing solution by (2). If a pin runs into a deadlock during the preprocessing step, then we know that no solution to the routing problem exists and therefore can abort computation before starting the time consuming fixed point iteration.

Finally, in Figure 7 an example is given to demonstrate how forced cells are detected and to show characteristics of the routing space that favor occurrences of forced cells. At the beginning, only the pins are forced cells. In the next iteration 4 further forced cells are detected. Two further forced cells can be found and afterwards, connectivity predicates are computed by the fixed point iteration introduced in Section 2.4. Pin locations moved forward during preprocessing and thus the pins of net 2 now are located at $(2,3,0)$ and $(3,2,0)$. For carrying out the fixed point iteration for one net, the originally 16 points to be considered can be reduced to 6 (the 2 pins of the nets and the 4 empty cells). If obstacles or pre-routed nets are included, the probability for detection of forced cells increases since the routing space gets more dense. E.g. if there is an obstacle at $(3,3,0)$, all the other 15 points become forced cells. And if an additional obstacle is added at $(4,4,0)$, the router recognizes without fixed point iteration that net 3 is not routable and that therefore no solution to the routing problem exists.

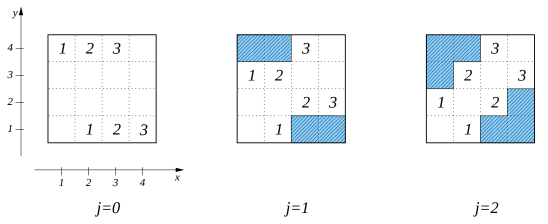


Figure 7: Detection of forced cells

5. EXPERIMENTAL RESULTS

We implemented the method described in the previous sections in C++. MDDs are simulated on top of BDDs and for realization of BDDs the state-of-the-art CUDD package [9] has been used. The experiments have been carried out on two workstations, one Sun Ultra 4 with 1.5 GB memory (Table 1+3) and one Sun Ultra 2 with a main memory of 1GB (Table 2). Runtimes are measured in CPU seconds.

The following experiments have been conducted. First of all, we compare our method to the preliminary results of [7]. Further on, we use routing problems with pins that are distributed at random throughout the whole routing space to show that efficiency of our approach is closely related to the density of the routing space. Additionally, the practicability of using the exact routing method in combination with traditional routing heuristics has been examined. Results for a small channel containing a cyclic VCG (Vertical Constraint Graph) conclude this section.

In [7], the routing problems `unsolvable`, `solvable` and `easy` have been introduced and their properties have been described in detail. Runtimes and sizes of solution MDDs of the approach presented there have been given for these examples. Table 1 compares the runtimes of the method presented in this paper with (FC) and without (NFC) computation of forced cells to the results given in [7]. In addition, we give the peak size during MDD construction for `solvable` (notice that final sizes are small in comparison to peak sizes). The parameter n denotes the number of tracks for `unsolvable` and `solvable` and the number of columns for `easy`. In `solvable`, there are only few forced cells and for this, results for routing with or without preprocessing, resp., are only slightly different. Therefore, only the results including forced cell detection are shown in the table. For `easy` and `unsolvable`, the number of detected forced cells is given in the columns denoted by "fc".

As can easily be seen, the approach presented here outperforms the method proposed in [7] by far. For `unsolvable` and `easy`, the whole routing space consists of forced cells and all of them are detected during preprocessing. This means that due to search space reduction no fixed point iteration is necessary and hence runtimes for method FC are reduced by more than 95% on average in comparison to those obtained when no preprocessing is used. When comparing computation times with those of the previously presented approach, a reduction by more than 99% in any case can be reported. For this, the maximum size of the problem instances that can be handled by the new method is increased by more than factor 10 on average. For `solvable`, node sizes and hence memory consumption become large if the size of the routing space is increased. The reason for this is that `solvable` is not dense.

To show that the exact router performs better if the density of the routing region is high, we generated problem instances with 3 nets, 3 layers and varying number of columns and tracks, resp., with pin densities 90%, 70% and 50%. A density of 50% means that 50% of the grid points are occupied by pins. For each size and density, 10 instances have been generated and average runtimes are shown in Table 2. Columns denoted by "fail" provide information about how often the result "no legal routing possible" occurred. In examples with a pin density of 50%, $3 \times 6 \times 3$ grids can be handled. A legal routing solution is found for half the instances while for the rest, the router detects that such a solution does not exist. When the size of the routing space is increased, time and memory requirements get too large, i.e. computation time exceeds 6000s, and therefore no results can be obtained then. If the pin density is increased to 70%, routing is no longer possible in almost all cases and this is detected by exact routing for $4 \times 10 \times 3$ grids. For a pin density of 90%, search space reduction is even more profitable and thus runtimes are much lower then, i.e. results can be computed within less than 1 second.

pin dens.:		90%		70%		50%	
#c.	#t.	time	fail	time	fail	time	fail
6	3	0.06	100%	0.61	90%	3856	50%
6	4	0.10	100%	0.62	100%	—	—
8	3	0.10	100%	2.57	100%	—	—
8	4	0.13	100%	12.57	100%	—	—
10	3	0.10	100%	27.70	100%	—	—
10	4	0.16	100%	54.82	100%	—	—

Table 2: Results for routing in regions with varying density

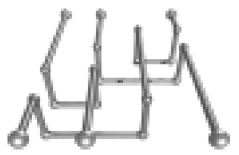


Figure 8: Routing in a channel with cyclic VCG.

Table 3 shows computation times, peak sizes of MDDs and numbers of forced cells detected during preprocessing when the exact router is applied to the example `vcgcycle` that is a channel with a cyclic VCG, (i.e. traditional channel routers are not feasible for this example). It contains 3 nets that have to be connected in the *MANHATTAN* model within 2 layers. Numbers of tracks and columns have been varied to make the example more general. Figure 8 illustrates a legal routing solution (4 columns, 4 tracks). Two of the nets may be generated by a conventional channel router, but the net connecting the pin on the lower left boundary with the one on the upper right boundary includes a dogleg and therefore it must have been generated by another technique like e.g. exact routing. The exact router is able to solve this problem standalone, but if 1 or even 2 nets are prerouted by a channel routing heuristic, MDD node sizes and therefore computation time are reduced by more than factor 10 on average as can be seen in the table. Situations given in `vcgcycle` often occur in parts of routing spaces during channel routing and in that cases, efficient routing becomes possible by cooperative use of channel routers and the exact router.

6. CONCLUSIONS

We presented an exact switchbox router that is suitable to be applied to dense routing regions and to be combined with traditional routing heuristics thus solving routing problems

	unsolvable				solvable			easy			
	[7]	NFC	FC		[7]	FC		[7]	NFC	FC	
n	time	time	time	fc	time	time	size	time	time	time	fc
2	0.06	0	0	6	0.13	0	2044	0.06	0.01	0	9
3	0.09	0	0	8	4.58	0.02	7154	0.08	0.04	0	12
4	0.20	0	0	10	195.18	0.10	24528	0.50	0.11	0	15
5	0.58	0	0	12	11702.30	0.29	66430	25.42	0.32	0	18
6	2.53	0.01	0	14	—	0.68	84826	—	0.83	0	21
7	8.75	0.02	0	16	—	1.83	180894	—	2.73	0.01	24
8	30.53	0.03	0.01	18	—	3.24	249368	—	6.50	0.02	27
9	106.11	0.04	0.01	20	—	5.73	379162	—	15.96	0.02	30
10	490.01	0.05	0.01	22	—	9.45	581518	—	82.76	0.02	33
20	—	0.16	0.01	42	—	214.27	7054866	—	—	0.06	63
60	—	18.21	0.01	122	—	—	—	—	—	0.58	183
100	—	91.82	0.05	202	—	—	—	—	—	1.93	303

Table 1: Comparison to [7]

		no pre-routing			1 net pre-routed			2 nets pre-routed		
#col.	#tr.	time	size	fc	time	size	fc	time	size	fc
4	4	0.25	42924	12	0.09	11242	13	0.02	2044	13
5	4	3.04	216664	12	0.45	74606	13	0.08	10220	13
6	4	28.05	968856	12	4.51	342370	13	0.54	85848	13
7	4	233.91	5376742	12	58.94	2650046	12	3.67	243236	12
4	5	1.91	156366	12	0.41	64386	13	0.07	10220	11
5	5	28.29	896294	12	4.55	359744	13	0.68	112420	11
6	5	335.50	8121834	12	50.17	1976548	13	10.14	776720	11
7	5	5670.73	36852298	12	1205.7	39924430	12	90.12	4995536	10

Table 3: Routing in a channel with cyclic VCG

that neither can be solved by heuristics nor by exact routers standalone. It has been shown that the routing approach proposed here provides the flexibility required for interaction with other routers. The set of all possible solutions is computed by a fixed point iteration and represented by an MDD. In a preprocessing step, paths that cannot be part of a legal solution are pruned and by that, search space for the time-consuming fixed point iteration is reduced.
Experimental results demonstrate that by the new technique, runtimes are reduced by up to 95%. Additionally, it has been shown that the exact router performs particularly well in routing regions with high density and that it is well suited to be used in combination with existing routing heuristics. Future work focuses on further search space reduction, e.g. by integration of more sophisticated implication based learning of forced cells and application of branch and bound methods. Further on, we plan to develop a router that realizes the idea of interaction between heuristics and exact routing.

7. ACKNOWLEDGEMENT

The authors like to thank Rolf Drechsler for helpful discussions on this work.

8. REFERENCES

[1] R. Bryant. Graph - based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[2] S. Devadas. Optimal layout via Boolean satisfiability. In *Int'l Conf. on CAD*, pages 294–297, 1989.

[3] N. Göckel, R. Drechsler, and B. Becker. A multi-layer detailed routing approach based on evolutionary algorithms. In *Int'l Conference on Evolutionary Computation*, pages 557–562, 1997.

[4] T. Kam, T. Villa, R. Brayton, and A. Sangiovanni-Vincentelli. Multi-valued decision diagrams: Theory and applications. *Multiple Valued Logic - An International Journal*, pages 9–62, 1998.

[5] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout.* Teubner, Wiley, 1990.

[6] G.-J. Nam, K. Sakallah, and R. Rutenbar. Satisfiability-based layout revisted: Detailed routing of complex fpgas via search-based boolean sat. In *Int'l Symp. on FPGAs for Custom Computing Machines*, pages 167–175, 1999.

[7] F. Schmiedle, R. Drechsler, and B. Becker. Exact routing using symbolic representation. In *Int'l Symp. Circ. and Systems*, pages VI:394–VI:397, 1999.

[8] N. Sherwani. *Algorithms for VLSI Physical Design Automation.* Kluwer Academic Publishers, Norwell, Massachusetts, second edition, 1995.

[9] F. Somenzi. *CUDD: CU Decision Diagram Package Release 2.3.0.* University of Colorado at Boulder, 1998.

[10] A. Srinivasan, T. Kam, S. Malik, and R. Brayton. Algorithms for discrete function manipulation. In *Int'l Conf. on CAD*, pages 92–95, 1990.

[11] T. Szymanski. Dogleg channel routing is NP-complete. *IEEE Trans. on CAD*, 4(1):31–41, 1985.

[12] R. Wood and R. Rutenbar. FPGA routing and routability estimation via Boolean satisfiability. *IEEE Trans. on VLSI Systems*, 6(2):222–231, 1998.

[13] T. Yoshimura and E. Kuh. Efficient algorithms for channel routing. *IEEE Trans. on CAD*, CAD-1(1):23–35, Jan. 1982.