Scalable, High-Quality, SAT-Based Multi-Layer Escape Routing

Sam Bayless, Holger H. Hoos, and Alan J. Hu
University of British Columbia, Canada
{sbayless,hoos,ajh}@cs.ubc.ca

ABSTRACT

Escape routing for Printed Circuit Boards (PCBs) is an important problem arising from modern packaging with large numbers of densely spaced pins, such as BGAs. Single-layer escape routing has been well-studied, but large, dense BGAs often require multiple PCB layers to be fully escaped. Unfortunately, multi-layer escape routing is much more challenging than single-layer escape routing, and currently lacks scalable, high-quality, automatic solutions. As a result, multi-layer escape routing for high-end BGAs typically requires extensive human intervention in practice.

This paper introduces a novel approach to multi-layer escape routing. Our approach builds on recent advances in SAT (Boolean satisfiability) solving, in particular, the solver MonoSAT, which efficiently supports network-flow constraints within a general constraint-solving framework. We formulate multi-layer escape routing in this framework and demonstrate scalability to the largest BGAs presently in common use, with more than 2000 pins. Our approach supports 45- and 90-degree routing, simultaneously places traces and vias, and supports all commonly used via technologies, including through-hole, blind, buried, and any-layer micro-vias. In addition, because our approach is based on constraint-solving, it can flexibly interoperate with partial solutions from other routing techniques.

We demonstrate the utility of our technique by finding escape routings for a diverse set of large, commercial BGAs. Compared to a typical layer-by-layer approach, our approach produces better routings, often saving one or more PCB layers for larger BGAs, and in some cases, proving that no solution is possible with fewer layers. Finally, we describe how our technique can be extended to handle common additional constraints, such as differential-pair constraints.

1. INTRODUCTION

As integrated circuits have become larger and denser, escape routing for dense packages such as ball grid arrays (BGAs) has become an increasingly important and challenging problem. High density BGAs may have 2000 or more pads, and there may only be room for a single trace between adjacent pads, requiring the use of multiple layers to fully escape the part. In fact, escape-routing dense BGAs is often the single factor that forces extra layers to be added to printed circuit boards (PCBs) [29].

However, while many prior studies have considered single-layer escape routing, few have addressed multi-layer escape routing. Whereas single-layer escape routing can be reduced to a network flow problem and solved using linear programming, multi-layer escape routing — in which both vias and traces are placed — cannot be modelled as a pure network flow problem. As a result, most existing solutions to multi-layer escape routing rely on a greedy, layer-by-layer approach, which can result in sub-optimal solutions requiring many additional layers.

Here we show how, by making use of the new capabilities of a state-of-the-art, open-source[1] constraint solver, MonoSAT [2], we can achieve efficient, fully automatic, and scalable multi-layer escape routing. Our approach simultaneously places traces and vias in high-density multilayer PCBs, supporting 45-degree (as well as 90-degree) grid-based routing, as well as several common via technologies (through-hole, blind, buried, and any-layer micro-vias). We show that our approach is appropriate to modern BGAs and modern PCB printing technology, and demonstrate experimentally that our approach scales to the largest and densest BGAs in common use at the time of writing. Across a variety of PCB technologies and real BGA packages, our technique finds solutions requiring fewer layers than a standard layer-by-layer approach. Finally, we extend our approach with support for differential pair routing, providing the first robust solution for multi-layer escape routing of differential pairs.

2. BACKGROUND

In order to integrate a package onto a PCB, traces must be routed on the PCB to connect each pin or pad on the package to its appropriate destination. PCB routing is a challenging problem that has given rise to a large body of research (e.g., [18, 43, 17]). However, high-density packages with large pin-counts, such as ball grid arrays, can be so difficult to route that a sub-problem, *escape routing*, already presents an important challenge. In escape routing, the goal of connecting each signal pin on the package to its intended destination is (temporarily) relaxed. Instead, an easier initial problem is considered: find a path from each

This work is licensed under a Creative Commons Attribution International 4.0 License.

ICCAD '16 November 07-10, 2016, Austin, TX, USA
© 2016 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-4466-1/16/11.
DOI: http://dx.doi.org/10.1145/2966986.2967072

[1]http://www.cs.ubc.ca/labs/isd/projects/monosat/

signal pin to *any* location on the PCB on the perimeter of the BGA. Once such an escape routing has been found, each of those escaped traces is routed to its intended destination in a subsequent step (that subsequent routing is not typically considered part of the escape routing process). Typically, during escape routing each pin is routed to the perimeter of the ball grid array, spread over the layers of the PCB.

For our purposes, a printed circuit board consists of one or more layers, with the package connected to the top-most layer. Some layers of the PCB are reserved just for ground or for power connections, while the remaining layers, sometimes including the top-most layer that the package connects to, are routable layers. Signals along an individual layer are conducted by metal traces, while signals crossing layers are conducted by vias. Different manufacturing processes support traces or vias with different diameters; denser printing capabilities can allow for multiple traces to fit between adjacent BGA pads (or, conversely, can support tighter spacing between adjacent BGA pads).

Different manufacturing processes also support different styles of vias, with different restrictions on their use. Four of the most common via types, all of which are supported by our approach, are *through-hole*, *blind*, *buried*, and *micro*-vias. *Through-hole* vias are holes drilled straight through the PCB and then coated with a conductive surface, connecting all the layers together at that position. *Blind* vias are drilled part-way through the PCB, starting from the top or bottom of the PCB. *Buried* vias are similar to through-vias, but connect inner layers of the PCB rather than the top-most and bottom-most. *Micro*-vias connect any two adjacent layers together. Vias are typically substantially wider in diameter than traces (typically, by a factor of 1.5, 2, or 3). As a result, placing a via has the effect of reducing the space available for routing traces on the layers that the via passes through, sometimes substantially.

The literature on escape routing can be roughly divided into several categories:

1. Analytical or optimal solutions:

   A small body of work proposes analytical solutions for optimal BGA breakout – but often with major limitations that would limit their utility in practice – usually, that the layout is for a single layer PCB. There are analytical solutions [27], optimal algorithms [42, 25], and also fast, heuristicly-guided (but not necessarily optimal) algorithms [32], all for variations of single-layer BGA breakout.

2. Design patterns and heuristics:

   Much of the literature on multi-layer escape routing proposes specific design patterns or heuristics for via placement and trace routing. For example, [36] proposes routing triangular sub-sections of each layer first, while [31] proposes a 'column-by-column' routing pattern for routing hexagonal (non-grid) pin-arrays. Much attention has also been given to the question of how to position BGA pads and vias on the very top layer of the PCB [16]. (However, in recent years technological improvements allowing vias to be placed directly underneath the BGA pads may have partially obviated the utility of such patterns, at least in high-end electronics [13].) A survey of such design patterns can be found in [30].

3. Network flow solutions:

   Many researchers have modelled single-layer escape routing as a network flow problem. Although in the general case, the disjoint-paths problem (in which the goal is to find disjoint paths in a graph between multiple *specific* start and end pairs) is NP-hard [26, 34], the special case in which all destinations are the same (or are interchangeable) can be transformed into a maximum-flow or a minimum-cost maximum-flow problem, and solved in polynomial time. Network flow-based single-layer escape routing has been considered in many studies [41, 35, 3, 12, 11, 38]; a good survey of these can be found in [39].

   The exact way in which the flow graph is constructed depends on a number of factors, including whether 45-degree routing is supported and whether the grid is rectilinear or hexagonal. Typically, the problem is modelled as a network flow instance in which each each edge *and each node* in the graph have capacity 1. (Node capacity constraints can be converted into edge capacity constraints by replacing each node with two nodes, and adding a single, 1-capacity arc between them.) In order to improve scalability, some encodings allow higher capacities per node (typically, 2 or 3), in which case some nodes correspond to multiple traces [38].

   Unfortunately, unless the positions of the vias are fixed in advance (in which case the problem reduces to a series of independent single-layer instances), multi-layer escape routing cannot be directly modelled as a network flow problem; this is because the vias that connect layers together are wider than normal traces. As a result, flow pushed along an edge representing a via must occlude any flow along adjacent edges to the via, on the layers above and below the via. This occlusion cannot be directly modelled as a pure network flow instance.

   For this and similar reasons, most existing network flow models of escape routing come with major limitations — typically that only a single layer is being routed, or that all wires and vias are the same width — or make strong simplifying assumptions about the nature of the package to be routed. For the special case of routing *buses*, in which two dense pin-arrays have a set of contiguous signals that must be routed together in a consistent order between the two pin arrays, an optimal, poly-time, multilayer routing algorithm is known [20, 37, 19].

4. Constraint-based approaches:

   Recently, several studies have explored using integer linear programming (ILP) solvers to combine network flow encodings with additional constraints. Although we will use a SAT solver rather than an ILP solver, these works are the most similar to ours in the literature (though they all differ from our contribution in key ways). Both [12] and [14] apply ILP solvers to single-layer escape routing. In [15], this work was extended to support multi-layer escape routing for staggered pin arrays, in which pins on alternating rows of the BGA are assumed to be offset from the previous row, forming a hexagonal pattern, rather than a rectlinear grid. Their encoding is specific to staggered pin grid arrays (just as ours is specific to rectilinear grid arrays), and so cannot be directly compared to our approach. Their approach also does not support differential-pair routing, whereas ours does. Another recent study, [22], also considered an ILP formulation for solving multi-layer (grid-based) escape routing as part of a chip co-design method. This work does support differential pairs, however, their formulation assumes that the solver can choose the positions of the signal pins on the package, as opposed to the usual case where the BGA's pin-out is given (the case we consider here), and so this work is also not directly comparable to our contribution.

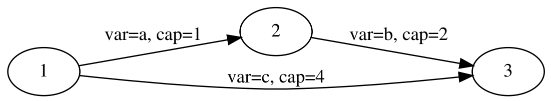
Many variants of the escape routing problem have been considered in the literature. [10, 21, 24, 33] consider (single-layer) escape routing in the presence of ordering constraints requiring some of the signals to be routed next to each other on the perimeter (this is a common requirement for signals representing contiguous bits of a word or address). [24] uses a SAT solver to perform ordered, single-layer escape routing. This approach is likely the most similar to ours in the literature; however they do not support via-routing or multi-layer routing, and report scalability only up to ≈ 256 pins, whereas our approach scales to more than 2000 pins and can route vias over multiple layers. On the other hand, we do not support ordering constraints. Some approaches also consider differential pair routing ([40, 23, 22]), which we will discuss in Section 6, or matched-length routing.

Overall, work on escape routing fits into the broader topic of trace routing. There is a large body of work addressing PCB and VLSI wire routing [18, 43, 17], including several approaches applying constraint solvers: [7, 6, 8, 4] applied SAT and answer-set-programming (ASP) solvers to rectilinear wire routing, and [9] applied an augmented SAT solver to clock routing. However, while these approaches are similar to ours in that they apply SAT solvers and related technologies to wire routing, none of them is appropriate for escape routing.

3. MAXIMUM FLOW IN SYMBOLIC GRAPHS

As many studies have previously observed (see, *e.g.*, [39]), single-layer escape routing can be modelled as a maximum-flow or a minimum-cost maximum-flow problem, and solved efficiently using either dedicated maximum flow algorithms or linear programming solvers (for minimum-cost flows). However, multi-layer escape routing cannot be directly modelled as a network flow problem if vias can be freely placed, because the placement of vias will typically occlude the placement of nearby traces on the layers crossed by the via. In other words, the network topology *changes* as different via placements are analyzed. Our answer to this challenge is to employ *symbolic graphs*, in which the edges in the graph or their capacities are variables to be assigned during the search for a solution. For example, Figure 1 shows a symbolic graph with 3 vertices and between 0 and 3 edges. Each edge has an associated Boolean variable indicating whether that edge is in the graph or not, so this symbolic graph encodes 8 different concrete graphs, corresponding to the 8 possible truth assignments to the variables a, b, and c.

The use of symbolic graphs, however, introduces a new problem of how to reason about network flows on symbolic graphs. Rather than solving a standard maximum flow problem (where one is given a graph and source and sink nodes, and the goal is to find the maximum s-t flow in that graph), the challenge becomes the inverse problem: given a set of allowable configurations of the edges in a partially specified graph G, and one or more constraints on the allowable maximum s-t flow in G for source and sink nodes s and t, how does one find a concrete graph with an s-t flow in the allowable range (or prove that no such graph exists)?



$$(a \vee b) \wedge (\neg b \vee \neg c) \wedge (maxflow_{1,3} \geq 1) \wedge (maxflow_{1,3} \leq 2)$$

Figure 1: A symbolic graph G with a logical formula constraining it; edge a is in G iff Boolean variable a is assigned TRUE. Assignment $\{a, b, \neg c\}$ satisfies both the propositional constraints and also the maximum flow constraints in the associated graph, while assignment $\{a, \neg b, c\}$ is not satisfying because the maximum flow from node 1 to node 3 is at least 4 when edge c is in G, violating constraint $maxflow_{1,3} \leq 2$.

Returning to the example in Figure 1, a propositional formula ϕ constrains a symbolic graph G, and the solver must find an assignment to the variables that satisfies ϕ. (The edges are also marked with capacities, which in general can be integer-valued variables, but for our purposes will be constants.) ϕ includes logical constraints, as well as two maximum flow constraints. The predicate $maxflow_{1,3} \geq 1$ evaluates to TRUE if and only if, under assignment to the edge atoms in ϕ, the maximum flow from node 1 to node 3 is at least 1, while the predicate $maxflow_{1,3} \leq 2$ evaluates to TRUE if and only if the maximum $1 \rightarrow 3$ flow is at most 2.

It is possible to encode symbolic graphs and max-flow constraints directly in propositional logic, or, more efficiently, using SAT modulo theory (SMT) solvers with support for linear real arithmetic, such as Z3 [5] or CVC [1]. Unfortunately, doing so on graphs of non-trivial size (more than a few hundred nodes) is prohibitively expensive [2]. However, in recent work [2] we introduced MONOSAT, a SAT modulo theory solver for a wide class of 'monotonic' theories, including efficient support for many properties from graph theory, finite state machines, and geometry, which were previously prohibitively expensive to encode in SAT solvers. Particularly relevant to our discussion here, MONOSAT provides high-performance support for formulas constraining the maximum s-t flow of symbolic graphs, reasoning about the maximum flow of graphs with on the order of 10,000 nodes and 100,000 edges. In the next section, we show how to formulate the multi-layer escape routing problem using maximum flow on symbolic graphs, thereby enabling the use of MONOSAT in a novel and efficient solution to multi-layer escape routing.

4. MULTI-LAYER ESCAPE ROUTING

In single-layer escape routing methods, the positions of the vias (and the pads or pins) are generally fixed, allowing one to remove, in advance, any traces that would be blocked by pins or vias, and hence reduce the problem to a pure network-flow problem that can be solved efficiently. In con-

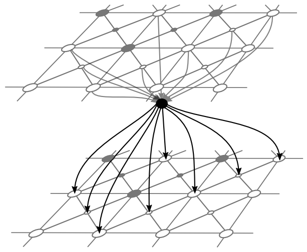
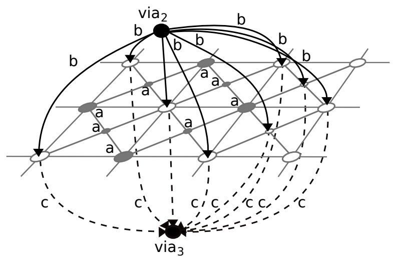


Figure 2: Mutli-layer escape routing with simultaneous via-placement. On-grid positions are shown as large nodes, while 45-degree traces pass through the small nodes. This is a *symbolic* graph, in which some of the nodes or edges in this graph are included only if corresponding Boolean variables in an associated formula ϕ are assigned to True. We construct ϕ such that nodes connecting adjacent layers (the central black node, representing a via) are included in the graph only if the nodes surounding the via (marked in gray) are disabled. The via nodes (black) are connected to all nodes around the periphery of the gray nodes, as well as connected to the central node (interior to the gray nodes).



$$via_2 \rightarrow \neg a_i$$

$$via_3 \rightarrow \neg a_i$$

$$(via_2 \wedge \neg via_3) \leftrightarrow b_i$$

$$(\neg via_2 \wedge via_3) \leftrightarrow c_i$$

Figure 3: Detail from layer 2 of Fig. 2, showing some symbolic nodes and edges controlled by Boolean variables $(a_i, b_i, c_i, via_2, via_3)$ in formula ϕ. To avoid clutter, the picture shows multiple edges with labels a, b, and c, but formally, each edge will have its own Boolean variable a_i, b_i, and c_i. All nodes and edges have capacity 1, however, nodes and edges with associated variables are only included in the graph if their variable is assigned True. Two via nodes are shown, one connecting from the layer above, and one connecting to the layer below. Nodes in the layer that are occluded if a via is placed in this position are shown in gray (in this case, the via has a diameter twice the width of a trace, but any width of via can be modelled simply by adjusting the pattern of nodes blocked by the via). The first two constraints shown enforce that if either via node is included in G, then the nodes in the layer that would be occluded by the via (in gray) must be disabled. The remaining constraints allow the nodes surrounding the blocked nodes to connect to the via if and only if the via begins or ends at this layer (rather than passing through from an upper to a lower layer). These constraints are included in ϕ for each potential via location at each layer in G.

trast, we will allow the constraint solver to choose where to place vias. As a result, except for the top-most layer of the PCB (in which the positions blocked by pads are fixed), the positions of the vias will not be known in advance.

In order to support the placement of vias, we will model multi-layer escape routing as a formula ϕ over a constrained, symbolic graph G, in which some of the edges are included in G only conditionally, depending on the assignment of variables in ϕ (as described in the previous section). This will allow us to model mutually exclusive edges for vias and the traces they occlude.

Figure 2 illustrates the symbolic flow graph we propose to model multi-layer escape routing. Each layer of this flow-graph is similar to typical single-layer network flow-based escape routing solutions (see, *e.g.*, [38]), except that in our graph all positions are potentially routable, with no spaces reserved for vias or pads. Each node in the graph has a node capacity of 1 (with node capacities enforced by introducing pairs of nodes connected by a single edge of capacity 1).

Instead of choosing the pads or vias in advance, we include *potential* vias in the graph, spaced at regular intervals, that the solver may choose to include or exclude from the graph. Potential vias are shown as black nodes in Figure 2, with neighbouring nodes shown in gray, indicating that they would be blocked by that via. In Figure 2, we show in gray the nodes that would be blocked by a via with a radius roughly 1.5 times the width of a trace. However, different via widths can be easily supported by simply altering the pattern of gray nodes to be blocked by the via.

Each via node is connected to the nodes surrounding the gray nodes that are blocked by the via, as well as the central node interior to the gray nodes (see Figure 3). These represent routable connections on the layer if the via is placed, allowing traces to be routed from the via nodes to the nodes surrounding the blocked nodes, or allowing the via to route through the layer and down to the next layer below. Each via node is associated with a Boolean variable (via_2 in Figure 3), such that the via node is included in the graph if and only if via_2 is True in ϕ. The potentially blocked nodes around each via are also associated with variables (for clarity, drawn as a in Figure 3, however in ϕ each edge will actually have a unique variable a_i). For each via, we include constraints $via \rightarrow \neg a_i$ in ϕ, disabling all the immediate neighbouring nodes if the via is included in the graph. Any satisfiable assignment to these constraints in ϕ selects a subset of the nodes of G representing a compatible, non-overlapping set of vias and traces.

Four configurations are possible for the two via nodes shown in Figure 3: (a) neither via node is enabled, allowing traces to be routed through the gray nodes on this layer, (b) the via enters from above, and connects to this layer, (c) the via begins at this layer, connecting to a layer below, and (d) the via passes through this layer, connecting the layer above to the layer below. By adding constraints restricting the allowable configurations of vias, as described in Figure 4, our approach can model vias as either through-hole vias, buried vias, blind vias, or any-layer micro-vias. With minor adjustments to the constraints, these different via types can be combined into a single model or can be restricted to specific layers of the PCB, allowing a wide variety of PCB manufacturing processes to be supported.

We add an additional source node s and sink node t to the graph, with directed, capacity-1 edges connecting the s to each signal in the graph, and directed, capacity-1 edges connecting all nodes on the perimeter of each layer to t. Finally, a single flow constraint $maxflow_{s,t} \geq |signals|$ is added to ϕ, ensuring that in any satisfying assignment, the subset of edges included in the graph must admit a flow corresponding to a valid escape routing for all of the signal pins.

Via Type	Constraint
Through-hole	$via_j \rightarrow (\neg a_i^1 \wedge \neg a_i^2 \wedge \ldots \neg a_i^n)$
Blind	$via_j \rightarrow (\neg a_i^1 \wedge \neg a_i^2 \wedge \ldots \neg a_i^j)$
Buried	$via_j \rightarrow (\neg a_i^s \wedge \neg a_i^{s+1} \wedge \ldots \neg a_i^t)$
Micro	—

Figure 4: Constraints enforcing different via models in ϕ, for $via_1 \ldots via_n$, where variables a_i^k control the potentially blocked nodes of via_j. (For variable a_i^k, the index k indicates the layer number, and the constraint is enforced for all values of i of neighbouring nodes to the via.) For buried vias, s and t are the allowable start and end layers for the buried via, determined by the type of buried via. Micro-vias allow any two adjacent layers to be connected and require no additional constraints (the default behaviour); if only a subset of the layers support microvias, then this can be easily enforced. Each of these via types can also be combined together in one routing or (excepting through-holes) restricted to a subset of the layers.

A solution to this formula will correspond to a feasible multi-layer escape routing, including via and trace placement. However, as MonoSAT only supports maximum flow constraints, and not minimum-cost maximum flow constraints, the trace routing in this solution is typically far from optimal (with traces making completely unnecessary detours, for example). For this reason, once MonoSAT has produced a feasible escape routing, including a placement of each via, we then apply an off-the-shelf minimum-cost maximum flow solver [28] to find a corresponding locally optimal trace routing for each individual layer of the feasible routing. This can be solved using completely standard linear-programming encodings of minimum-cost maximum-flow, as the vias are already placed and the layer that each signal is to be routed on is already known.

5. EVALUATION

We evaluate our procedure on a wide variety of dense ball grid arrays from four different companies, ranging in size from a 28x28 pad ARM processor with 382 routable signal pins to a 54x54 pad FPGA with 1755 routable signal pins. These parts, listed in Tables 1 and 2, include 32-bit and 64-bit processors, FPGAs, and SoCs. The first seven of these packages use 0.8mm pitch pads, while the remainder use 1mm pitch pads. Each part has, in addition to the signal pins to be escaped, a roughly similar number of power and ground pins (for example, the 54x54 Xilinx FPGA has 1137 power and ground pins, in addition to the 1755 signal pins). Most parts also have a small number of disconnected pins, which are not routed at all. Typically, power and ground pins are routed to a number of dedicated power and ground layers in the PCB, separately from the signal traces; we assume that the bottom-most layers of the PCB contain the power and ground layers, and route all power and ground pins to those layers with through-hole vias. This leaves only the routable signal pins to be escaped in each part on the remaining layers.

For comparison, we implemented a simple network-flow based single-layer escape routing algorithm, similar to the one described in [38]. We then implemented a greedy, layer-by-layer router by routing as many signals as possible on the top-most layer (using maximum flow), and, while unrouted signals remain, adding a new layer with vias connecting to each unrouted signal. This process repeats until no unrouted signals remain. As can be seen in Table 1, this layer-by-layer routing strategy is simple but effective, and has been previously suggested for multi-layer escape routing in several works in the literature (for example, [36] combines this strategy with their single-layer routing heuristic to create a multi-layer escape routing method).

In Table 1, we compare our approach to the layer-by-layer strategy using blind vias, and, in Table 2, using through-hole vias. All experiments were run on a 2.67GHz Intel x5650 CPU (12Mb L3, 96 Gb RAM), in Ubuntu 12.04. Although our approach supports buried and micro-vias, we found that all of these instances could be solved by MonoSAT with just 2 or 3 signal layers, even when using the more restrictive through-hole and blind via models, and so we omit evaluations for these less restrictive models (which, in two or three layer PCBs, are nearly equivalent to blind vias).

In Tables 1 and 2, MonoSAT finds many solutions requiring fewer layers than the layer-by-layer strategy (and in no case requires more layers than the layer-by-layer approach). For example, in Table 1, MonoSAT finds a solution using blind vias (for the TI AM5K2E04 processor, packaged in a dense, 33x33 pad, 0.8mm pitch BGA) which requires only 3 signal layers, whereas the layer-by-layer approach requires 4 signal layers for the same part. In this case, MonoSAT was also able to prove that no escape routing using 2 or fewer layers was possible for this circuit (assuming the same grid-model is used). In Table 2, using more restrictive through-vias, there are several examples where MonoSAT finds solutions using 1 or even 2 fewer signal layers than the layer-by-layer approach. We can also see, in Tables 1 and 2, that MonoSAT tends to find solutions with slightly longer net lengths on average.

A multi-layer routing produced by MonoSAT is shown in Figure 5. This escape routing, for an ARM processor, required 2 fewer signal layers than the solution found by greedy layer-by-layer routing. Examining Figure 5 suggests a likely explanation for how MonoSAT was able to reduce the layer count: in several places near the periphery of the escape routing, MonoSAT has left channels open at lower layers by routing several adjacent pads at the top layer (which does not require vias), instead of routing them at lower layers using through-vias. These channels then allowed large numbers of traces to be routed through the spaces left open at lower layers, which would otherwise have been mostly blocked by through vias.

6. EXTENSIONS

One common requirement in practical escape routing is to support differential pair routing, in which some of the signals are pairs that should be routed together wherever possible (and, in particular, should be routed on the same layer and to adjacent positions in the escape routing).

Because our approach relies on a general purpose constraint solver, it can be augmented with additional constraints. We extend our approach to support differential pair routing for through-vias in two steps. First, for each differential pair (A, B), we add a constraint enforcing that they be routed to the same layer (while still allowing the solver to choose which layer to route them on). This is done simply by forcing the through-vias for the paired signals to be assigned to the same value at each layer (while still al-

Part	Size	#Signals	Layer-by-Layer			MonoSAT		
			Layers	Length (mm)	Time (s)	Layers	Length (mm)	Time (s)
TI AM5716	28x28	382	3	8.2	5.4s + 63.6s	**2***	8.5	54.4s + 81.2s
TI AM5718	28x28	390	3	8.2	5.4s + 70.4s	**2***	8.3	47.4s + 118.4s
TI AM5726	28x28	477	3	6.8	5.3s + 49.7s	3	7.5	53.3s + 492.8s
TI AM5728	28x28	485	3	6.7	5.3s + 48.6s	3	7.2	53.8s + 387.2s
TI TMS320C	29x29	533	4	8.0	7.7s + 81.5s	**3**	8.8	75.0s + 497.7s
TI AM52E02	33x33	676	4	8.9	9.5s + 107.7s	**3**	9.6	103.8 + 921.8
TI AM5K2E04	33x33	690	4	8.7	9.2s + 96.8s	**3***	9.9	114.7s + 962.0s
TI 66AK2H1	39x39	876	3	16.0	24.0s + 508.0s	**2***	16.8	338.4s + 878.1s
Lattice M25	32x32	601	2*	11.8	10.4s + 160.5s	2*	12.3	140.1s + 306.7s
Lattice M40	32x32	687	3	11.6	114.6s + 205.6s	**2***	12.0	194.0s + 364.4s
Lattice M40	34x34	729	3	13.0	18.5s + 300.0s	**2***	13.1	254.3s + 425.2s
Lattice M80	34x34	785	3	12.1	17.8s + 266.5s	**2***	13.0	411.3s + 505.8s
Lattice M80	42x42	1133	3	14.4	27.0s + 499.0s	3	15.0	810.3s + 882.4s
Lattice M115	34x34	785	3	12.1	16.9s + 274.1s	**2***	13.0	392.9s + 578.2s
Lattice M115	42x42	1171	3	14.1	27.4s + 461.4s	3	14.6	242.5s + 254.7s
Altera 10AX048	28x28	448	2*	10.2	8.0s + 109.3s	2*	11.3	85.1s + 183.4s
Altera 10AX066	34x34	636	2*	12.7	13.1s + 218.3s	2*	13.7	151.1s + 371.2s
Altera 10AX115	34x34	652	2*	13.2	13.9s + 286.8s	2*	13.6	168.5s + 501.9s
Altera 10AT115	44x44	994	3	16.7	29.6s + 579.5s	3	17.4	384.8s + 928.8s
Altera EP4S100	44x44	1008	3	16.9	31.0s + 698.6s	3	17.6	401.8s + 1154.6s
Xilinx XCVU160	46x46	981	3	17.5	34.3s + 617.9s	3	18.3	414.3s + 977.5s
Xilinx XCVU440	49x49	1536	4	19.2	52.2s + 1167.5s	**3***	20.1	1246.9s + 2133.7s
Xilinx XCVU440	54x54	1755	4	20.0	60.9s + 1438.1s	**3***	21.6	1597.3s + 2726.9s

Table 1: **Multi-layer escape routing with blind vias.** *Run-times are reported as a + b, where a is the time to find a feasible multi-layer routing, and b is the time to post-process that feasible solution using minimum-cost maximum flow routing. Boldface highlights when our approach required fewer layers; solutions that use a provably minimal number of layers are marked with *. Length shows the average trace length in mm. These solutions ignore differential pair constraints (routing differential signals as if they were normal signals).*

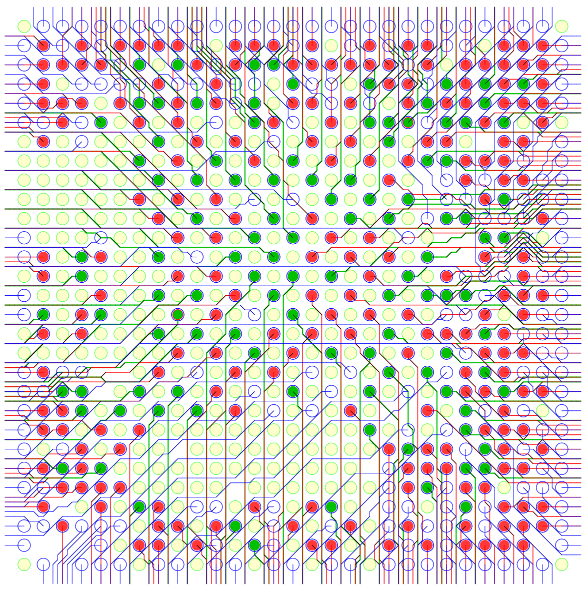


Figure 5: A multi-layer (3 signal layers + power and ground) escape routing produced by MonoSAT, using through-hole vias, for the TI TMS320C, a 29x29, 0.8mm pitch BGA. This routing required 238 seconds, and required 2 fewer signal layers than a greedy layer-by-layer approach. Pads and traces on the top layer are in blue, vias and traces on layer 2 are in red, and on layer 3 are in green. Ground and power vias (yellow) are routed to layers not shown.

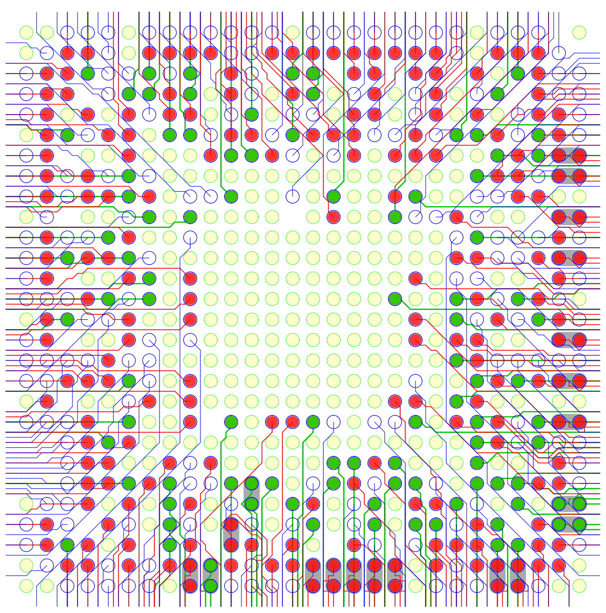


Figure 6: A multi-layer (3 signal layers + power and ground layers) escape routing for the TI AM5728 ARM processor, produced by MonoSAT. As in Figure 5, traces on layer 1 are blue, on layer 2 are red, and on layer 3 are green. This escape routing uses blind vias (allowing traces on layer 3 to be routed beneath vias to layer 2), supports differential pairs (pairs shown in gray), and required just over 200 seconds.

Part	Size	#Signals	Layer-by-Layer			MonoSAT		
			Layers	Length (mm)	Time (s)	Layers	Length (mm)	Time (s)
TI AM5716	28x28	382	3	8.2	5.4s + 58.9s	**2***	7.8	35.8s + 62.9s
TI AM5718	28x28	390	3	8.2	5.1s + 61.8s	**2***	7.5	46.0s + 69.1s
TI AM5726	28x28	477	4	7.1	7.7s + 63.1s	**3**	7.2	97.6s + 80.2s
TI AM5728	28x28	485	4	6.8	6.8s + 56.7s	**3**	7.4	110.1s + 85.4s
TI TMS320C	29x29	533	5	8.2	10.9s + 106.1s	**3**	9.2	109.6s + 128.4s
TI AM52E02	33x33	676	5	8.9	13.4s + 133.3s	**3**	9.6	203.2s + 180.4s
TI AM5K2E04	33x33	690	5	8.8	12.7s + 125.4s	**3***	9.5	291.3s + 187.8s
TI 66AK2H1	39x39	876	3	16.1	24.9s + 495.1s	**2***	16.6	347.9s + 510.7s
Lattice M25	32x32	601	2*	11.8	10.7s + 143.7s	2*	12.0	132.3s + 278.7s
Lattice M40	32x32	687	3	11.6	15.3s + 183.1s	**2***	12.2	161.5s + 311.3s
Lattice M40	34x34	729	3	13.0	18.0s + 270.2s	**2***	13.7	183.9s + 405.6s
Lattice M80	34x34	785	3	12.1	17.7s + 238.4s	3	12.6	304.8s + 638.2s
Lattice M80	42x42	1133	3	14.5	26.2s + 478.4s	3	15.1	810.3s + 882.4s
Lattice M115	34x34	785	3	12.1	16.8s + 227.1s	3	12.6	364.2s + 358.7s
Lattice M115	42x42	1171	3	14.3	27.8s + 457.3s	3	15.1	945.9s + 1500.3s
Altera 10AX048	28x28	448	2*	10.2	8.2s + 98.2s	2*	10.5	109.2s + 115.8s
Altera 10AX066	34x34	636	2*	12.7	14.0s + 212.3s	2*	14.0	203.5s + 282.9s
Altera 10AX115	34x34	652	2*	13.2	13.3s + 235.1s	2*	13.6	198.2s + 293.7s
Altera 10AT115	44x44	994	3	16.7	28.9s + 455.2s	3	16.9	616.1s + 992.9s
Altera EP4S100	44x44	1008	3	17.0	28.5s + 589.2s	3	17.5	733.5s + 834.5s
Xilinx XCVU160	46x46	981	3	17.6	32.5s + 538.3s	3	18.2	646.6s + 1216.4s
Xilinx XCVU440	49x49	1536	4	19.1	53,7s + 1051.1s	**3**	20.1	3457.5s + 1284.5s
Xilinx XCVU440	54x54	1755	4	19.9	600s + 1373.6s	**3**	21.2	6176.9s + 1861.9s

Table 2: **Multi-layer escape routing with through-vias.** *Run-times are reported as $a+b$, where a is the time to find a feasible multi-layer routing, and b is the time to post-process that feasible solution using minimum-cost maximum flow routing. Boldface highlights when our approach required fewer layers; solutions that use a provably minimal number of layers are marked with *. Length shows the average trace length in mm. These solutions ignore differential pair constraints (routing differential signals as if they were normal signals).*

lowing the solver to select which of those layers to route the pair to): $\bigwedge_{i=1}^{n} viaA_i = viaB_i$.

In the second step, we combine MonoSAT with the single-layer differential pair routing method from [40], by repeatedly solving the routing formula ϕ while adding extra constraints until MonoSAT's solution admits a valid differential routing, as follows: MonoSAT initially finds a feasible escape routing that does not route the differential pairs together, but which does (because of the above constraint) route both signals of each pair on the same layer. Next, on each layer where MonoSAT has placed differential pairs, we apply the single-layer differential pair routing technique described in [40]. If this technique fails, then there must exist a layer in which at least one differential pair failed to route, or, after successfully routing the differential pairs, at least one non-differential signal could no longer be escaped. We find one such unrouted signal or differential pair, and collect all the signals between that unrouted signal and the nearest edge of the package on the same layer. We then add a new constraint to MonoSAT, asserting that this unrouted differential pair (or unrouted signal) must be placed on a different layer than at least one of those nearby signals. After adding this constraint to ϕ, we re-solve with MonoSAT and repeat.

By repeatedly forcing the differential pairs and signals that fail to route to separate layers, as described above, this process will eventually find a valid differential routing, given enough routable layers. In the worst case, this may require an exponential number of constraint solver calls. However, in practice we found that it converges quickly. For example, Figure 6 shows a multi-layer escape routing produced by the above technique for an ARM processor, the TI AM5728, in which more than 20 pairs of signals have been routed differentially by MonoSAT while also escaping the remaining signals, across 3 layers. This solution required only 2 iterations of the above process to solve, and 204 seconds of total compute time.

7. CONCLUSION

We have introduced a fully automatic, constraint-solver based technique for efficient multi-layer escape routing and shown that in a wide variety of dense, commercial ball grid arrays, our technique can find escape routings using fewer layers than greedy, layer-by-layer escape routing. Our technique runs in just a few hours even for very dense packages with more than 2000 pins. Finally, we have shown that our technique can be extended to handle multi-layer escape routing with differential pairs, which, to the best of our knowledge, we are the first to support.

8. ACKNOWLEDGMENTS

This work was supported in part by grants from the Natural Sciences and Engineering Research Council of Canada. We also thank Jacob Bayless for his assistance, and for bringing this subject to our attention.

9. REFERENCES

[1] Clark Barrett, Christopher L Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In *CAV*, 2011.

[2] Sam Bayless, Noah Bayless, Holger Hoos, and Alan Hu. SAT Modulo Monotonic Theories. In *AAAI*, 2015.

[3] Wun-Tat Chan, Francis YL Chin, and Hing-Fung Ting. Escaping a grid by edge-disjoint paths. *Algorithmica*, 2003.

[4] Elvin Coban, Esra Erdem, and Ferhan Ture. Comparing ASP, CP, ILP on two challenging applications: Wire routing and haplotype inference. *LaSh*, 2008.

[5] Leonardo De Moura and Nikolaj Bjorner. Z3: An efficient SMT solver. In *TACAS*. 2008.

[6] Deborah East and Miroslaw Truszczynski. More on wire routing with ASP. In *Workshop on ASP*, 2001.

[7] Esra Erdem, Vladimir Lifschitz, and Martin DF Wong. Wire routing and satisfiability planning. In *Computational Logic*. 2000.

[8] Esra Erdem and Martin DF Wong. Rectilinear steiner tree construction using answer set programming. In *ICLP*, 2004.

[9] Amit Erez and Alexander Nadel. Finding bounded path in graph using SMT for automatic clock routing. In *CAV*, 2015.

[10] Jia-Wei Fang, Chin-Hsiung Hsu, and Yao-Wen Chang. An integer-linear-programming-based routing algorithm for flip-chip designs. *TCAD*, 2009.

[11] Jia-Wei Fang, I-Jye Lin, Yao-Wen Chang, and Jyh-Herng Wang. A network-flow-based RDL routing algorithmz for flip-chip design. *TCAD*, 2007.

[12] Jia-Wei Fang, I-Jye Lin, Ping-Hung Yuh, Yao-Wen Chang, and Jyh-Herng Wang. A routing algorithm for flip-chip design. In *ICCAD*, 2005.

[13] Frank Grano, Felix Bruno, Dana Korf, Eamon OKeeffe, Cheryl Kelley, and NH Salem. Impact of microvia-in-pad design on void formation. In *SMTA*, 2003.

[14] Yuan-Kai Ho, Hsu-Chieh Lee, and Yao-Wen Chang. Escape routing for staggered-pin-array PCBs. *TCAD*, 2011.

[15] Yuan-Kai Ho, Xin-Wei Shih, Yao-Wen Chang, and Chung-Kuan Cheng. Layer minimization in escape routing for staggered-pin-array PCBs. In *ASP-DAC*, 2013.

[16] Michio Horiuchi, Eiji Yoda, and Yukiharu Takeuchi. Escape routing design to reduce the number of layers in area array packaging. *Advanced Packaging*, 2000.

[17] Chin-Hsiung Hsu, Huang-Yu Chen, and Yao-Wen Chang. Multi-layer global routing considering via and wire capacities. In *ICCAD*, 2008.

[18] Richard M Karp, Frank Thomson Leighton, Ronald L Rivest, Clark D. Thompson, Umesh V Vazirani, and Vijay V Vazirani. Global wire routing in two-dimensional arrays. *Algorithmica*, 1987.

[19] Hui Kong, Tan Yan, and Martin DF Wong. Optimal simultaneous pin assignment and escape routing for dense PCBs. In *ASP-DAC*, 2010.

[20] Hui Kong, Tan Yan, Martin DF Wong, and Muhammet Mustafa Ozdal. Optimal bus sequencing for escape routing in dense PCBs. In *ICCAD*, 2007.

[21] Yukiko Kubo and Atsushi Takahashi. A global routing method for 2-layer ball grid array packages. In *ISPD*, 2005.

[22] Seong-I Lei and Wai-Kei Mak. Optimizing pin assignment and escape routing for blind-via-based PCBs. *TCAD*, 2016.

[23] Tai-Hung Li, Wan-Chun Chen, Xian-Ting Cai, and Tai-Chen Chen. Escape routing of differential pairs considering length matching. In *ASP-DAC*, 2012.

[24] Lijuan Luo and Martin DF Wong. Ordered escape routing based on Boolean satisfiability. In *ASP-DAC*, 2008.

[25] Qiang Ma, Hui Kong, Martin DF Wong, and Evangeline FY Young. A provably good approximation algorithm for rectangle escape problem with application to PCB routing. In *ASP-DAC*, 2011.

[26] Matthias Middendorf and Frank Pfeiffer. On the complexity of the disjoint paths problem. *Combinatorica*, 1993.

[27] Leonidas Palios. Connecting the maximum number of nodes in the grid to the boundary with nonintersecting line segments. *Journal of Algorithms*, 1997.

[28] Laurent Perron. Operations research and constraint programming at Google. In *CP*. 2011.

[29] Charles Pfeil. BGA breakout challenges. *PCB Fabrication Magazine*, pages 10–13, 2007.

[30] Charles Pfeil. BGA breakouts and routing. 2010.

[31] Rui Shi and Chung-Kuan Cheng. Efficient escape routing for hexagonal array of high density I/Os. In *DAC*, 2006.

[32] Xiaoyu Song, Shaodi Gao, and Tejasvi P Chakravarthy. An efficient ball grid array router. *International Journal of Electronics*, 2002.

[33] Yoichi Tomioka and Atsushi Takahashi. Monotonic parallel and orthogonal routing for single-layer ball grid array packages. In *ASP-DAC*, 2006.

[34] Jens Vygen. NP-completeness of some edge-disjoint paths problems. *Discrete Applied Mathematics*, 1995.

[35] Dongsheng Wang, Ping Zhang, Chung-Kuan Cheng, and Arunabha Sen. A performance-driven I/O pin routing algorithm. In *ASP-DAC*, 1999.

[36] Renshen Wang, Rui Shi, and Chung-Kuan Cheng. Layer minimization of escape routing in area array packaging. In *ICCAD*, 2006.

[37] Tan Yan, Hui Kong, and Martin DF Wong. Optimal layer assignment for escape routing of buses. In *ICCAD*, 2009.

[38] Tan Yan and Martin DF Wong. A correct network flow model for escape routing. In *DAC*, 2009.

[39] Tan Yan and Martin DF Wong. Recent research development in PCB layout. In *ICCAD*, 2010.

[40] Tan Yan, Pei-Ci Wu, Qiang Ma, and Martin DF Wong. On the escape routing of differential pairs. In *ICCAD*, 2010.

[41] Man-Fai Yu and Wayne Wei-Ming Dai. Pin assignment and routing on a single-layer pin grid array. In *ASP-DAC*, 1995.

[42] Man-Fai Yu and Wayne Wei-Ming Dai. Single-layer fanout routing and routability analysis for ball grid arrays. In *ICCAD*, 1995.

[43] Man-fai Yu, Joel Darnauer, and Wayne Wei-Ming Dai. *Planar interchangeable 2-terminal routing*. Technical Report, UCSC, 1995.