Inverted Ant Colony Optimization for Die/Package/PCB Co-design

Daniel Boyne
Austin Community College
danboyne@gmail.com

Abstract

Routing software is often optimized for only one of the three physical domains of electronic systems: the silicon die, the printed circuit board (PCB), or the intervening package. System-level architects must ensure that physical connectivity can be achieved among these domains, in addition to satisfying system-level electrical performance requirements. Simply achieving connectivity can require weeks or months of engineering effort, with experts in each domain iteratively adjusting their physical designs using specialized software tools.

To address system-level connectivity, the present work describes an open-source, exploratory tool to provide initial guidance to system, package, or PCB designers. The tool is not expected to provide an optimal solution within any given domain; its utility instead is its ability to span multiple domains without dependencies on other software tools – proprietary or otherwise.

Given the person-weeks of effort often required for traditional die/package/PCB co-design, this utility prioritizes capability over speed. The proof-of-concept version achieved full connectivity within four days for an industry-like routing configuration that include a PCB, a ball-grid-array (BGA) package with a C4-bumped die, and coarse routing on the silicon die. The rip-up-and-reroute algorithm employs A* pathfinding in a three-dimensional, grid-based array; negotiated congestion that dissipates over time, similar to inverted ant colony optimization; and concepts borrowed from simulated annealing to avoid local minima in the routing cost.

Keywords

Physical design, co-design, inverted ant-colony optimization, A* algorithm, negotiated congestion.

1 INTRODUCTION

Simultaneously designing a silicon chip, printed circuit board (PCB), and intervening package is necessary for launching new electronic systems that are optimized for performance or cost. The technical and organization challenges of this co-design process are well documented, e.g. [1,2], including newer challenges associated with the increasing use of packages containing multiple, heterogeneous silicon [3]. Electronic design automation (EDA) vendors offer a growing array of tools aimed at these challenges. Some tools are vendor-agnostic, although many are understandably optimized to integrate with software tools from a single vendor. Inter-company co-design, which can be complicated by organizational barriers, may also suffer from a lack of EDA interoperability. These challenges serve as the motivation for an open-source utility‡ intended to enable co-design from on-silicon I/O buffers to components on the PCB.

1.1 Problem Scope

With the challenges described above, a non-proprietary utility would be useful for system-level designers to work with chip-, package-, and PCB-designers for assessing the routability of various design configurations across these three domains. For any one domain, a fully optimized and manufacturable design is beyond this work's scope; EDA vendors provide solutions for domain-specific optimization, from length-matching to design-for-manufacturability. As depicted in Figure 1, the scope of the current work is the routing of:

- Top-level silicon traces, such as a coarse redistribution layer (RDL), with C4/pillar connections to a flip-chip package,
- Flip-chip packages with a ball-grid array (BGA) interface to a PCB,
- PCB routing.

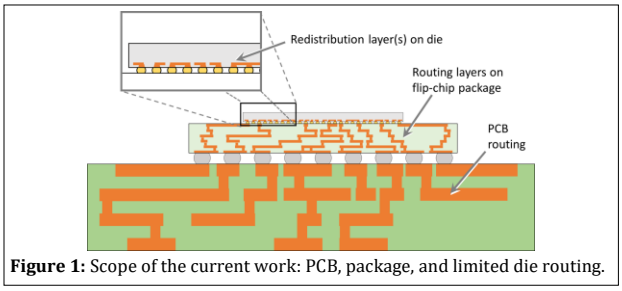

Figure 1: Scope of the current work: PCB, package, and limited die routing.

The scope of the current work is limited to planar routing on a die, package, and PCB. Connections between routing layers are limited to vertical connections such as laser vias, solder balls, copper pillars, or C4s. It is understood that linewidths and spacings vary dramatically between the die, package, and PCB. Likewise, such dimensions can vary within different regions of the package and from net to net.

Finally, the current work is limited to routing two-terminal nets, including differential pair (DP) nets. Such nets frequently offer bigger co-design challenges than multi-terminal power and ground nets.

In the interest of brevity, the routing utility is referred to as *Acorn*; a name based on its inspiration from Ant-Colony Optimization for Routing Nets. Inspired by biological, stigmergic systems like insect colonies and hives [4], Acorn employs the A* algorithm so that nets are repelled by congestion (pheromones) deposited by other nets (organisms).

1.2 Previous Work

Two overlapping areas of previous research are described in this section related to the chip, package, and PCB domains: (1) general net-routing solutions, and (2) those focused on co-design.

McMurchie et al. [5] developed a negotiated congestion (NC) router that iteratively ripped up and rerouted nets while monotonically increasing the congestion cost along occupied routing paths. Tessier augmented NC with depth-first A* pathfinding [6]. Chan et al [7, 8] accelerated the compute-time of the NC router using separate CPUs to route each net in parallel, in addition to other speed-up features. Lin et al. [9] utilized NC and A* pathfinding in a system with flexible physical constraints, including irregular arrangements of terminals.

Beyond the microelectronics literature, Dias et al. [10] combined inverted ant-colony optimization (IACO) with the breadth-first Dijkstra algorithm for optimizing roadway traffic. Nguyen et al. [11] used a similar approach called the 'inverted pheromone model', introducing randomization to avoid oscillatory traffic behavior.

In the co-design literature, Fang et al. [12,13] addressed chip, package, and PCB co-design using integer linear programming (ILP) techniques, including the routing of DP nets and on-die routing. This work was later extended to allow the router to select different solder bumps, or waypoints, between the die and package to optimize the routing [14]. Further efforts addressed multiple layers of on-die routing, or redistribution layers (RDLs) [15].

‡ Source code available at https://github.com/danboyne/ACORN

1.3 New Contributions of Current Work

Like the NC-based router in Lin et al. [9], this work allows for flexible definitions of the layout constraints. Acorn differs from previous NC-based routers by allowing for the cost of congested routing resources to reduce over time as nets negotiate alternative routes, as is common in IACO-based methods used in optimizing roadway traffic. Acorn augments previous grid-based routers by allowing for up to 16 routing directions: 0°, 26°, 45°, 63°, 90°, …. Further, subsets of these directions can be applied to different regions, e.g., to allow only Manhattan routing.

Unlike [9], the connections in Acorn between the traces of differential pair nets and their terminals are flexible enough to minimize trace lengths at each terminal, regardless of whether the pair is P/N-swappable. The same applies to the connections between differential pair traces and each inter-level via.

This work introduces 2- and 3-dimensional routing zones with near-zero routing cost and zero congestion cost, and in which design-rule violations are allowed. By placing terminals within such zones, the associated nets become 'swappable' among each other. In the extreme case, all nets can have one of their two terminals placed in such a zone to use the router for unordered, multilayer escape-routing.

To the author's knowledge, this is the first rip-up-and-reroute algorithm that can make large, stepwise changes to routing costs and subsequently assess those changes' effects on routing quality, thereby predicting whether additional changes would further improve the routing.

2 Problem Formulation

Acorn reads all input information from a text file, including the netlist, a physical description of the available routing regions, design rules, and a small number of control parameters. As in [16], another input is the maximum number of iterations to execute before giving up. This acknowledges that convergence to a solution free of design-rule violations is not guaranteed by negotiated congestion (NC) or inverted ant-colony optimization (IACO) algorithms.

2.1 Routing Boundaries and Costs

The input text file defines the number of routing layers and intervening via layers, including the allowed areas for routing on each of these layers. Routing barriers of arbitrary shape are accommodated, resulting in systems like that shown in Figure 2(b). In this example are 5 routing layers and 4 via layers, including the regions on each layer where lateral and vertical routes are allowed.

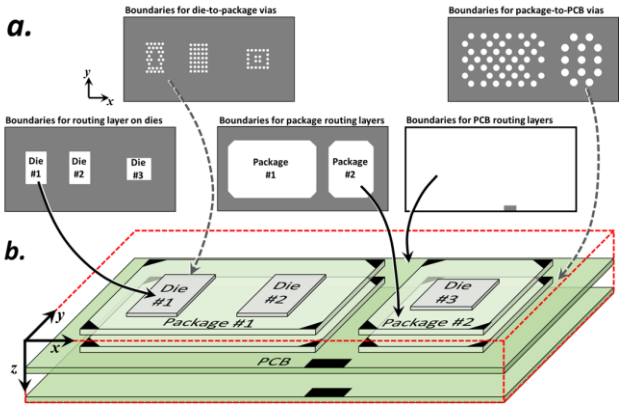

Figure 2: The definition of the routing map (b) includes the number of routing layers and lateral extents (a) for each layer, including via layers (top).

The input file may include increased costs for routing in any region of any layer, thereby prompting Acorn to avoid routing in such regions.

2.2 Routing Design Rules

Acorn checks for minimum spacing between different nets, which consist of traces and interlevel vias. Trace widths and via diameters are specified in the input file and guaranteed by design. For design-rule checking, vias are subdivided into two types: upward-going and downward-going vias, each of which may have different diameters and minimum spacing. Figure 3 depicts the three shape-types (Trace, Via Up, and Via Down) and the 9 design-rules required in the input file, including the minimum allowed spacings between each of the three shape-types.

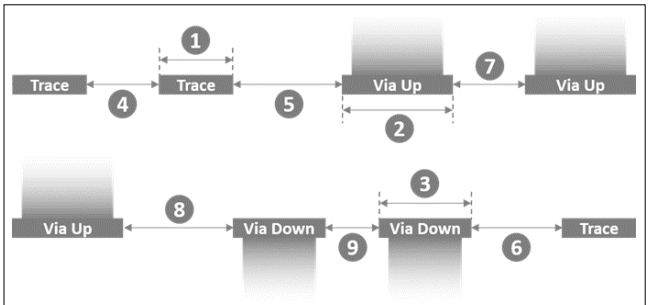

Figure 3: Acorn creates traces and vias of the widths specified by in the input file, and denoted above as rules 1, 2, and 3. Design-rule violations occur if the minimum spacing rules, 4 through 9, are not satisfied by the routing.

In addition to widths and spacings, the input file may specify the allowed routing directions. In this way, the allowed angles of routing may be constrained: all-angle routing, subject to the constraints listed in Section 3.2); Manhattan routing; 45° routing; up/down (vertical)-only routing through vias; lateral-only routing that prohibits vias; and four other options.

Nets may be assigned to groups that have design rules different from other nets, e.g., larger trace-widths for power/ground nets, or larger spacings for electrically sensitive nets. For DP nets, the input file specifies the pitch of DP traces. It is understood that DP traces should not be routed as close as possible if the target characteristic impedance, e.g., 100 Ω, requires greater spacing.

Co-design requires different design rules for different physical regions, e.g., fine-pitch lines on the die and coarse routing on the PCB. The input file therefore defines design-rule zones to which each design-rule set applies. Such zones can apply to an entire routing layer, or to sections of a given layer (e.g., using finer design rules for escape routing).

2.3 Pin Swapping

Early in the co-design process, there may be flexibility in the assignment of pins to/from a given component. RAM data-busses are common examples, in which the ordering of bits within an eight-bit byte may be scrambled to optimize physical routing. Such pin-swapping is achieved in Acorn by locating the terminals of pin-swappable nets into a pin-swap zone, as illustrated in Figure 4(a). In this example, two separate pin-swap zones (in yellow) allow 8 pins to be swapped on the left, and a separate 8 pins to be swapped on the right. Because the two pin-swap zones are not contiguous, pins from the left swap-zone cannot be swapped with those from the right.

To use Acorn for escape routing, a terminal of every net is placed within a single swap-zone. For example, Figure 4(b) illustrates single-layer escape-routing in which a terminal of every net was placed at the lower-left corner of the perimeter pin-swap zone. Within pin-swap zones, design-rule violations are allowed.

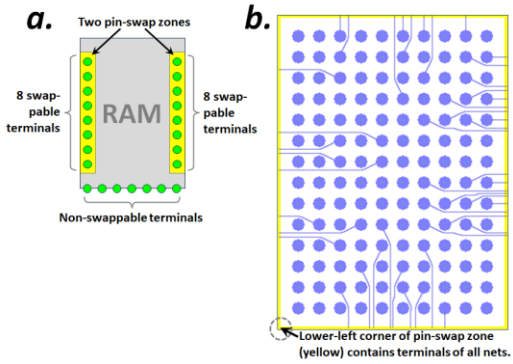

Figure 4: Pin-swap zones in (a) allow for 8 nets to be interchanged in each byte. In (b), all nets are placed in a single pin-swap zone to assess escape routing.

Differential pairs present a special case of pin-swapping; some circuits allow the positive (P) and negative (N) nets of the DP to be swapped without affecting performance. Such nets can be identified with a flag in the netlist; there is no need to locate the DP terminals in a pin-swap zone. Nonetheless, placing DP terminals in a pin-swap zone automatically defines the DP as P/N-swappable, but the P- and N-traces of the DP are always routed together. For example, the P-net is never arbitrarily exchanged with an unrelated net from the same pin-swap zone.

2.4 Escape Routing

As illustrated above, a carefully constructed pin-swap zone can be used for escape routing on a single layer. This concept may be extended to study escape routing across one or more domains of the die/package/PCB system. Figure 5(a) depicts a pin-swap zone (yellow) beneath all the package-to-PCB solder balls. By locating the terminals of all nets in this pin-swap zone, one may determine viable pin-maps for the package-to-PCB connections. Extending this concept, a single pin-swap zone can extend vertically across all routing layers surrounding the perimeter of a PCB, as shown in Figure 5(b). Locating terminals of all nets in this pin-swap zone prompts Acorn to find viable, violation-free routing across the die, package, and PCB, as detailed further in Section 4.3.

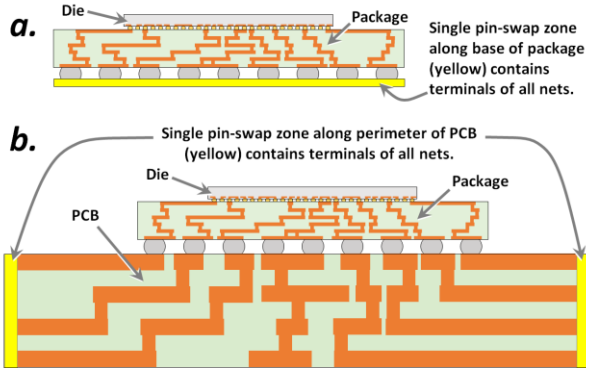

Figure 5: Escape routing can be assessed with low-cost pin-swap zones (yellow) placed horizontally beneath a package, as in (a), to assess viable BGA maps. In (b), escape routing on multiple PCB layers can be assessed with vertically stacked pin-swap zones.

3 Algorithm

An overview of the algorithm is presented in the flowchart of Figure 6. Before entering the iterative rip-up and reroute loop, Acorn reads user-defined information from an ASCII text file, including the netlist; the layout of die, package, and/or PCB; and the grid resolution, i.e., size in µm of each cell in the map. In general, this dimension should be smaller than the smallest linewidth plus spacing.

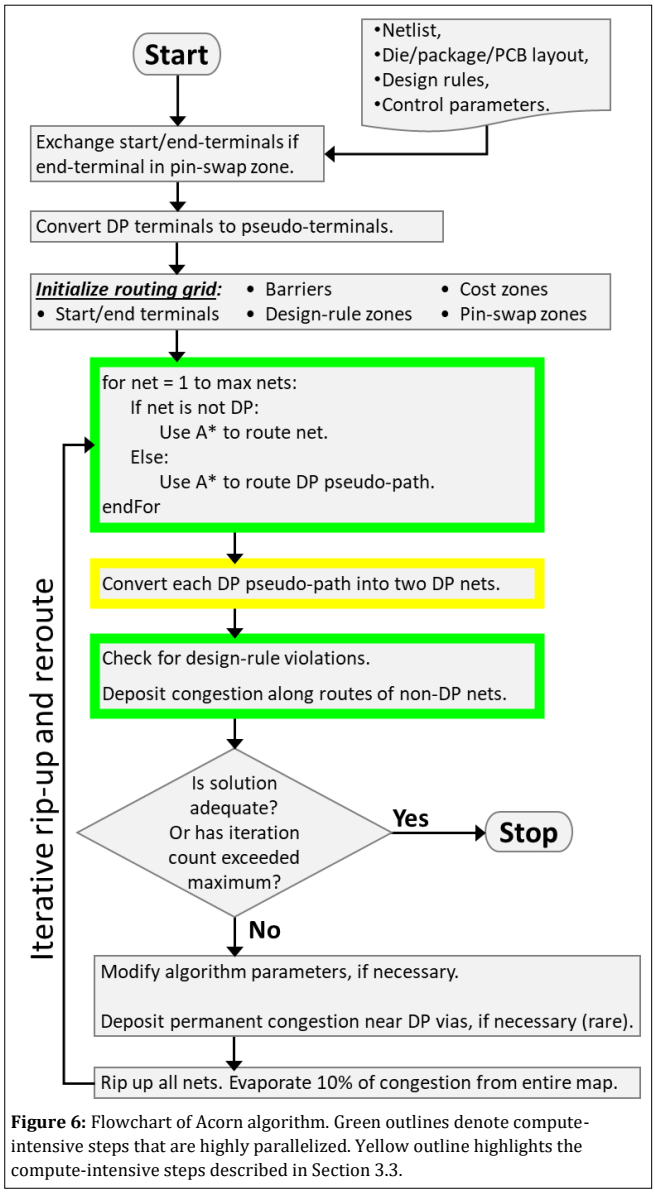

Figure 6: Flowchart of Acorn algorithm. Green outlines denote compute-intensive steps that are highly parallelized. Yellow outline highlights the compute-intensive steps described in Section 3.3.

The algorithm terminates if (a) the number of iterations has exceeded the user-defined limit, or (b) the solution is deemed adequate. An adequate solution is achieved if at least N_{good} iterations resulted in routing that is free of design-rule violations. A complex design with many nets should naturally require more iterations than a simpler design with fewer nets. Likewise, the minimum threshold N_{good} also depends on N_{nets}. This dependence was arbitrarily chosen as $N_{good} = 20 \cdot \log_{10}(N_{nets})$, i.e., N_{good} grows slowly with N_{nets}. For example, in a 100-net design, at least 40 iterations must result in violation-free routing. Consequently, a successful run results in many viable routing options. Acorn highlights which iteration represents the lowest-cost option, i.e., that with the lowest via count and shortest aggregate net length, accounting for user-defined cost-zones.

3.1 Routing Grid Definition

For a system with N_{layers} routing layers, the routing grid consists of N_{layers} two-dimensional grids that represent the X/Y extent of the entire map. At each cell, Acorn initially stores the following information:

- which design-rule applies;
- whether the cell is part of a user-defined cost-zone;

- whether the cell is part of a pin-swap zone; and
- whether the cell contains a barrier to prohibit routing in the X/Y plane or vertically to another layer.

Acorn does not maintain separate layers for inter-level vias, so via barriers are stored in the routing cell with the ability to distinguish between barriers to upward-going and downward-going vias, and/or barriers to lateral routing in the X/Y plane.

For each pair of DP nets, Acorn creates a pseudo-path whose width spans the widths of both DP nets when routed at their prescribed pitch (as described in Section 3.3). Likewise, Acorn creates pseudo start- and end-terminals for each pair. These terminals are located at the midpoints of the DP nets' terminals.

3.2 Iterative A* Routing

Excluding DP nets, A* pathfinding is applied to each net and pseudo-net during every iteration. (DP net routing is derived from these pseudo-nets, and is covered in the next section.) Because each net's routing depends only on the congestion costs from previous iterations, the results do not depend on the sequence of routing the nets.

From each cell in the map, the algorithm evaluates jumps in up to 18 directions, as depicted in Figure 7. While short of any-angle grid-based schemes (e.g., [17]), Acorn's scheme allows lateral routing at angles of 0°, 26.6° [$\tan^{-1}(½)$], 45° [$\tan^{-1}(1)$], 63.4° [$\tan^{-1}(2)$], 90°, etc. The input file defines which of these directions are allowed in any given region of the design.

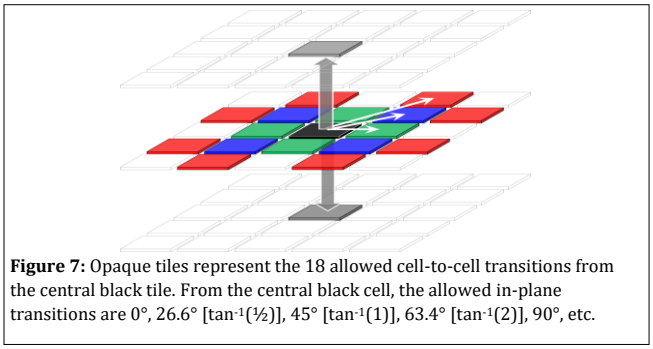

Figure 7: Opaque tiles represent the 18 allowed cell-to-cell transitions from the central black tile. From the central black cell, the allowed in-plane transitions are 0°, 26.6° [$\tan^{-1}(½)$], 45° [$\tan^{-1}(1)$], 63.4° [$\tan^{-1}(2)$], 90°, etc.

A* attempts to find the path with the minimum F-cost [18]. At each cell (x, y, z) in the explored map, F is the sum of a G-cost and a heuristic function, H:

$$F(x,y,z) = G(x,y,z) + H(x,y,z)$$

The heuristic function is a lower estimate of the G-cost between a given cell and the net's end-terminal. If the absolute values of the components of this distance are denoted by Δx, Δy, and Δz, Acorn calculates the heuristic based on the allowed directions that are permitted from the given cell:

$$H(\Delta x,\Delta y,\Delta z) = \begin{cases} b_{xy}\sqrt{\Delta x^2+\Delta y^2} + b_z\Delta z & for\ all-angle\ routing \\ b_{xy}(\Delta y\sqrt{2}+\Delta x-\Delta y) + b_z\Delta z & for\ Manhattan\ and\ diagonal\ routing\ with\ \Delta x > \Delta y \\ b_{xy}(\Delta x\sqrt{2}+\Delta y-\Delta x) + b_z\Delta z & for\ Manhattan\ and\ diagonal\ routing\ with\ \Delta y > \Delta x \\ b_{xy}(\Delta x+\Delta y) + b_z\Delta z & for\ Manhattan-only\ routing \end{cases}$$

In the expressions above, b_{xy} and b_z are the base values for, respectively, a lateral move to an adjacent cell and a vertical jump to one layer above or below. These values are significantly smaller in the small pin-swap zones than in normal, non-pin-swap zones that constitute most of the routing area. The values of b_{xy} and b_z are tabulated in Table 1 below.

Table 1: Base values b_{xy} and b_z for calculating H- and G-cost values.

	Pin-swap Zones	Normal Zones
b_{xy}	10	$10{\cdot}2^{30}$
b_z	10	$10{\cdot}2^{30}\cdot N_{vertCost}$,

The base-costs in pin-swap zones are at least 10^9 (2^{30}) times smaller than in the normal, non-pin-swap zones. (The factor of 2^{30} enables fast binary arithmetic.) The purpose of pin-swap zones is to allow nets to freely explore regions with relatively low cost. Acorn prohibits A* from routing any net *into* these low-cost pin-swap zones. Instead, nets may route *within* such zones, and they may *exit* such zones into higher-cost areas. This ensures that the heuristic function remains admissible, since pathfinding for pin-swappable nets always begins in the low-cost pin-swap zones.

The parameter $N_{vertCost}$ in Table 1 is a dimensionless ratio (>1) of the user-defined $D_{vertCost}$ divided by the cell size. $D_{vertCost}$ represents the lateral distance that Acorn should laterally route a trace around an obstacle rather than creating vias to route above/below the obstacle. Larger values of $D_{vertCost}$ result in routing with fewer vias.

For the G-cost, Acorn includes two components: a distance-related cost, G_{dist}, and a congestion-related cost, G_{cong}.

$$G(x,y,z) = G_{dist}(x,y,z) + G_{cong}(x,y,z)$$

G_{dist} is the cost of travelling from the start-terminal to (x,y,z). Like the heuristic function, G_{dist} is based on the base costs b_{xy} and b_z, whose values differ markedly between pin-swap and normal zones (Table 1). In addition, G_{dist} may account for user-defined cost zones. In such optional cost zones, G_{dist} is increased by an integer multiplier to minimize routing on a certain layer or in a certain area.

The congestion-related G-cost, G_{cong}, is an added penalty for routing through cells that, in previous iterations, were traversed by other nets. There is no penalty for a path to traverse a cell that previously was traversed only by the same net; self-congestion adds no penalty, unlike foreign congestion. (In the context of a hypothetical ant colony, each ant is repulsed by the pheromones of other ants, but is not repelled by its own scent.)

In pin-swap zones, G_{cong} is defined as zero. In the remainder of the map, G_{cong} is the product of five factors for each cell along a path:

- The number of times the cell has been traversed by foreign nets,
- A multiplier that scales with the base costs of the cell (b_{xy} or b_z),
- A geometric multiplier that estimates a typical distance needed to detour around the nets that caused the congestion (generally based on design rules and net lengths),
- The user-defined cost-zone multiplier, if applicable, as described above, and
- A dynamic sensitivity factor that increases slowly during early iterations, and which Acorn can later adjust with the goal of reducing design-rule violations.

Consistent with the decay of pheromones in IACO, the first factor in the above list is reduced at the end of each iteration. For each path that traverses the cell, Acorn reduces the count by 10%.

In the context of IACO, the last factor in the above list – the sensitivity to foreign congestion—may be interpreted as the degree to which an ant is repelled by the pheromones of other ants. Varying this factor has proven effective in achieving good routing results. Unlike changes to the amount of deposited pheromones (congestion cost), the sensitivity factor can be modulated for a single iteration without affecting the pheromone (congestion) map.

During A* pathfinding, the congestion-related G-cost is calculated for only the centerline of the net being routed. To avoid foreign nets, therefore, Acorn deposits congestion far from each net's centerline, and indeed beyond the width of its traces and via. Figure 8 illustrates the location of deposited congestion (grey) for six nets (red) routed on a single layer. The deposited congestion repels the centerlines of foreign nets by increasing their congestion-related G-costs. Note that congestion overlaps with foreign traces but does not overlap with the centerlines of such traces. By calculating the appropriate locations for depositing congestion, Acorn can converge to routing that is free of design-rule violations.

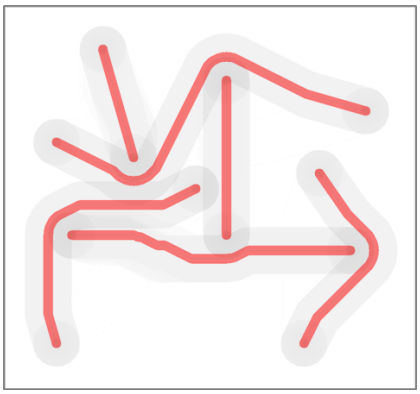

Figure 8: Traces (red) and their associated congestion (grey). In the absence of design-rule violations, no trace's centerline overlaps with another net's congestion.

To avoid design-rule violations in complex designs, Acorn calculates an appropriate distance from each net for depositing congestion, including congestion for:

1. Different width and spacing rules for traces, upward-going vias, and downward-going vias,
2. Different width and spacing rules for power/ground nets, signal nets, and DP pseudo-nets (described in next section),
3. Boundaries between different design-rule zones, at which linewidths and spacings can change.

As an example of item (1), Acorn deposits trace-related congestion nearby a trace, but must deposit *via* congestion farther away from the trace to repel large-diameter vias from being routed near the trace. For this reason, Acorn keeps track of three congestion attributes at each cell: the amount of congestion, the path that caused it, and the shape-type that caused it (trace, upward-, or downward-going via).

3.3 DP Post-Processing

ACO, IACO, and NC algorithms are well suited to attract or repel nets from each other. However, differential pair nets present unique challenges. These nets must route close to one another, but not too close. Despite attempts to develop more elegant solutions, it was decided to collapse both DP nets into a single 'pseudo-path,' which is treated by A* in a manner similar to non-DP paths (not unlike [9]).

Near the end of each iteration, after pseudo-paths have been routed along with non-DP paths, Acorn performs four post-processing steps for DP nets before checking for design-rule violations. These are listed below and illustrated in Figure 9.

1. Create traces on both sides of the pseudo-path, separated by the user-defined pitch for the given DP (Figure 9(b)),
2. Remove portions of the DP traces from step (1) to allow for graceful transitions from DP traces to vias (Figure 9(c)).
3. Create vias for each DP net where the pseudo-path transitions between layers (Figure 9(d)),
4. Connect the DP traces to DP vias using A* pathfinding, which respects the deposited congestion from neighboring nets (Figure 9(e)).

Step 4 is compute-intensive and includes pre-calculations that determine which of the two DP traces should connect to each DP via. (Unlike the example in Figure 9(e), this decision is not obvious in highly symmetric DP routing.) Despite the run-time penalty, careful attention to DP traces and vias has enabled faster convergence to violation-free solutions.

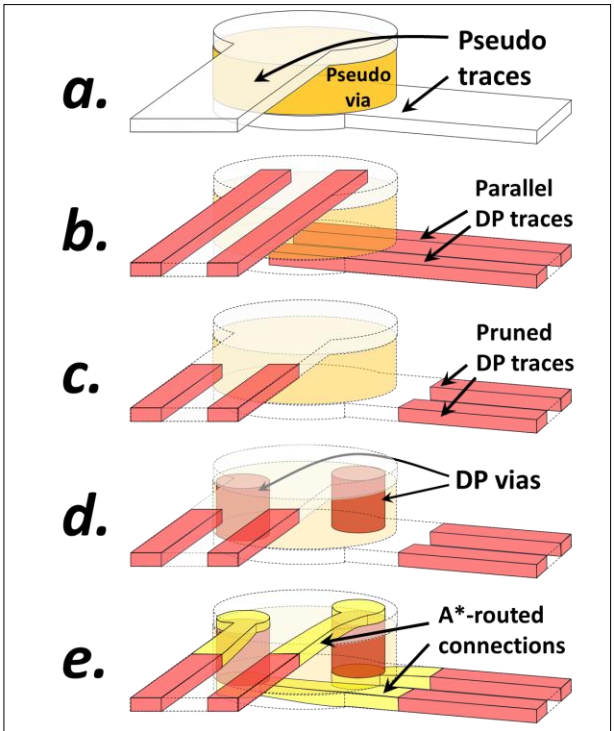

Figure 9: In place of DP nets, A* is used to route wide pseudo-paths (a), consisting of pseudo-traces (white) and pseudo-vias (gold). Acorn then creates parallel traces along the edges of the pseudo-traces (b) before removing segments near pseudo-vias (c). DP vias are created (d), consistent with design rules for each layer. Finally, A* routing is used to bridge the gaps along the DP nets, depicted by the yellow traces in (e).

3.4 Real-time Algorithm Tuning

Over the course of many rip-up-and-reroute iterations, the routing occasionally converges to a steady state that contains design-rule violations. Three strategies are employed to avoid such situations:

1. Exchanging the start- and end-terminals of nets that have violations,
2. Changing the congestion sensitivities for all nets, thereby changing the G-cost in the A* pathfinding, and
3. Rarely, depositing permanent (non-evaporating) congestion in regions with persistent design-rule violations.

As a concrete example, Figure 10 shows the aggregate path length as a function of Acorn run-time over 857 iterations for a 356-net, 5-layer test case. The path length (green) initially increases before converging to a relatively stable value. Each blue dot represents one of 139 iterations with no design-rule violations. When Acorn detects an unwanted plateau in the aggregate length, it selects one of the above three strategies to upset the routing configuration. These strategies are detailed below.

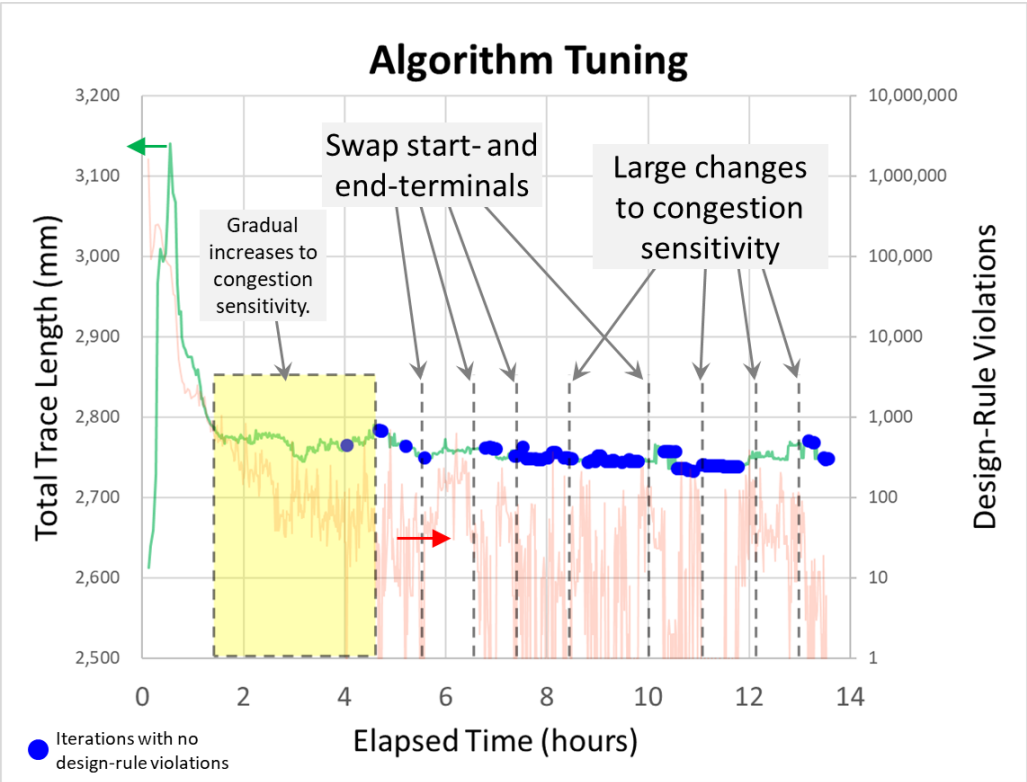

Figure 10: As a function of run-time, the total routing length (green) and number of design-rule violations (red) are plotted. Blue points represent 139 iterations with violation-free solutions. Yellow region and dashed lines denote when routing parameters were modified to reduce path costs and design-rule violations.

Acorn exchanges the start-/end-terminals to alter the routing when multiple routes have equivalent F-costs for the same net. Acorn's implementation of A* deterministically selects one of these lowest-cost routes from iteration to iteration. But when start/end-terminals are exchanged, Acorn can select a different route with equivalent F-cost.

Acorn modulates the congestion sensitivity, which is one of the factors that constitutes G_{cong}, to change the cost for nets to cross each other's paths. This sensitivity has no effect on G_{dist} and therefore has no direct effect on nets without design-rule violations.

Figure 11 provides a conceptual diagram of the aggregate F-cost from all nets as a function of the (infinity of) routing configurations. In Figure 11(a) are three local minima (A, B, C) and a global minimum (D) – the latter representing a routing configuration with design-rule violations. For zero or small values of the congestion sensitivity (small G_{cong}), the global minimum will always represent a routing configuration with design-rule violations, except for trivially simple problems. By increasing the congestion sensitivity (Figure 11(b)), Acorn increases the cost of configurations that contain design-rule violations without affecting violation-free configurations. Eventually, this will cause local minimum C, which contains no violations, to become a global minimum at which Acorn attempts to converge.

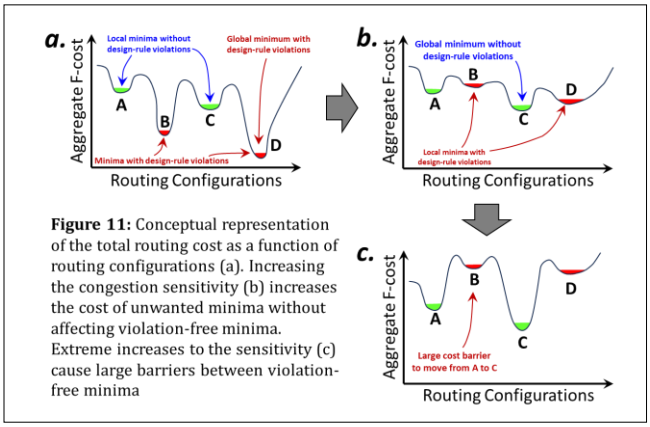

Figure 11: Conceptual representation of the total routing cost as a function of routing configurations (a). Increasing the congestion sensitivity (b) increases the cost of unwanted minima without affecting violation-free minima. Extreme increases to the sensitivity (c) cause large barriers between violation-free minima

If the congestion sensitivity is increased too much, however, Acorn has been found to converge to configurations that are obviously suboptimal. Figure 11(c) is one explanation; the cost-barrier between violation-free configurations A and C becomes so large that Acorn converges to local minimum A rather than global minimum C.

No theoretical attempt has been made to predict the optimal congestion sensitivity *a priori* for a given case. Instead, Acorn gradually increases the sensitivity early in the run (yellow region in Figure 10) before settling on an arbitrary (but deterministic) value. If persistent design-rule violations persist after at least 60 iterations, then Acorn significantly increases the sensitivity by 41% (by a factor of $\sqrt{2}$). At the end of an additional 60 iterations, Acorn compares the quality of the routing at the two sensitivity values. If the higher value resulted in equivalent or better routing metrics, the sensitivity is increased again by $\sqrt{2}$x. Such increases continue every 60 iterations until the routing metrics begin to degrade, at which point Acorn decreases the sensitivity to congestion. This coarse gradient-descent is inspired by simulated annealing (SA) algorithms in which the temperature is adjusted to modulate the likelihood of overcoming cost barriers. A large congestion sensitivity in Acorn (Figure 11(c)) is analogous to a low SA temperature; both inhibit transitions between local cost-minima in complex systems.

Acorn deposits permanent congestion in rare situations at locations that persistently exhibit a specific type of design-rule violation. The congestion is intended to repel DP pseudo-traces when violations on the associated DP traces cannot be resolved through the normal IACO process. Such persistent violations can occur when user-defined barriers constrain vias to a small region, such as the die-to-package or package-to-PCB interface where such vias are constrained in rectangular or hexagonal (staggered) arrays. The permanent congestion causes the A* pathfinder to add a pseudo-via to a layer that is less crowded with DP traces.

Finally, Acorn introduces a pseudorandom component into the routing for all iterations that exhibit design-rule violations. Specifically,

for all nets that contain design-rule violations during an iteration, Acorn randomly selects a small number of nets (1-3) to have their congestion sensitivities temporarily modified on the subsequent iteration. For some nets, the sensitivity will increase by as much as 400%; for others, it will reduce by up to 98%. Such temporary, single-iteration changes can significantly disrupt the routing, but are believed to enable convergence to routing configurations with overall lower costs, while avoiding oscillatory routing behavior, similar to [11]. Acorn behaves deterministically despite these pseudorandom effects.

4 Experimental Results

The Acorn algorithm was developed in the C programming language with use of the OpenMP libraries for multithreading and LibPng libraries to generate routing layouts for viewing in a web interface.

More than 300 small regression test cases with 1 to 30 nets were created to validate routing and design-rule checking (e.g., Figure 4(b)). Larger test cases with 356 nets were created to mimic industry routing challenges – especially in the package. Relative to typical industry designs, these large test cases have comparable routing areas, design rules for linewidths and spacings, and densities of terminals. The 356-net cases used a grid-size of 20 μm.

For these large test cases, the software was run on Linux using 16-, 32-, and 64-threaded Arm-based, compute-optimized, Amazon Graviton3 processors from Amazon Web Services with memory equal to 2 GB per thread.

4.1 IC Package Routing

To simulate the routing of a flip-chip package, 365 two-terminal nets were arranged in a netlist to route from realistic C4 sites on the top layer of the package (near the die) to solder balls in the ball-grid array on the bottom layer. The netlist included 24 differential pairs (i.e., 48 DP nets), in addition to 15 two-terminal power/ground nets with larger linewidths. Depending on the electrical performance requirements, a package of this size and wire density could easily require eight metal layers, including the bottom layer dedicated largely to solder balls. For this test case, only five layers were included because Acorn does not route power/ground planes.

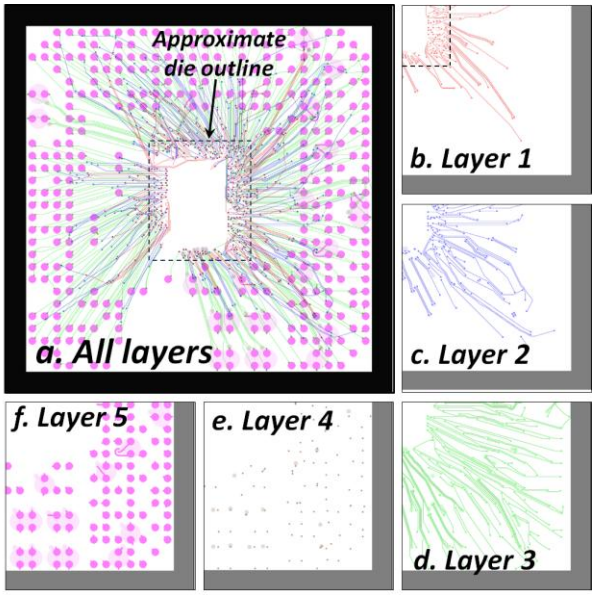

Figure 12: Lowest-cost routing configuration for a 356-net package using five routing layers. Layer 5 is the BGA layer; faint circles around BGA pairs indicate DP nets.

Figure 12(a) shows the lowest-cost routing configuration for this package, with quadrants of each layer in Figures 12(b)-(f). The lack of significant routing in the center of the package and on layers 4 and 5 was intentional; these regions were defined as high-cost zones, thereby forcing Acorn to avoid routing in these areas. In a real package, such zones could be used for power/ground planes.

The routing metrics for this test case are plotted in Figure 10. Using 16 threads, Acorn required 13.6 hours to complete 857 iterations, including 139 iterations with no design-rule violations. The first iteration without such violations was found after only 4.0 hours. However, the lowest-cost iteration was found at 10.9 hours. Subsequent iterations had comparable (but higher) costs. The aggregate path length of this lowest-cost configuration was 4.6% longer than the value derived from simply connecting each start- and end-terminal with a straight line. The number of vias in the design was the theoretical minimum for the net- and layer-counts, largely due to a large value used for $D_{vertCost}$ (15,000 μm), which was more than half the lateral size of the entire package. This choice made each via costly relative to lateral routing.

4.2 Package and PCB Routing

To assess Acorn with simultaneous package and PCB routing, three PCB layers were added to the five-layer package described in the previous section. A netlist for a fully populated PCB was not available, so this case instead investigated PCB escape-routing from the package. Unlike typical PCB escape-routing studies, however, BGA sites were not assigned to specific nets. Instead, these package-to-PCB solder-balls sites were flexible waypoints for nets to negotiate among each other. More solder-ball sites were defined than required by the netlist to enable traces to negotiate for the limited BGA waypoints.

Similar to Figure 5(b), all nets' PCB terminals for this test were located in pin-swap zones on the perimeter of the PCB. Acorn therefore attempted to find the shortest paths to this perimeter using the fewest vias without violating design rules. Like the previous test case, $D_{vertCost}$ was 15,000 μm for this run.

Acorn achieved 146 violation-free routing configurations during 1320 iterations over the course of 60 hours using 32 threads. Routing on the package layers was qualitatively similar to the previous case (Figure 12). Figure 13 shows the overall routing and the PCB routing.

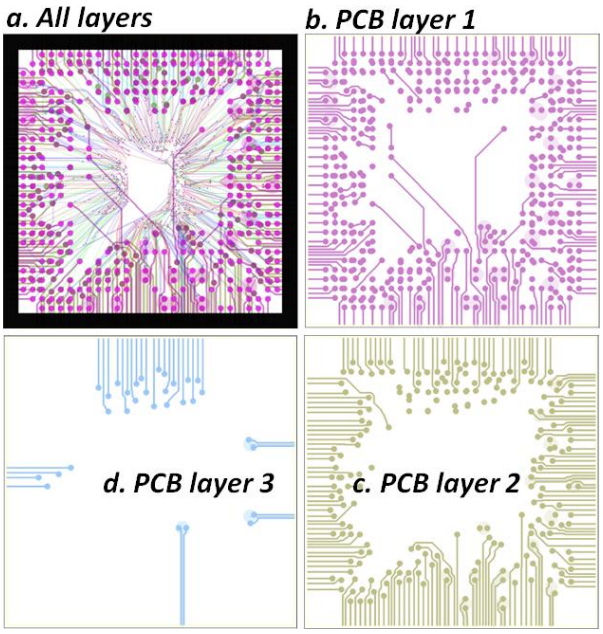

Figure 13: Lowest-cost routing configuration for a 356-net case using five package routing layers and three PCB layers. All eight layers are shown in (a), with PCB layers shown in (b)-(d). PCB terminals are located in pin-swap zones that extend across all three PCB layers at their perimeter.

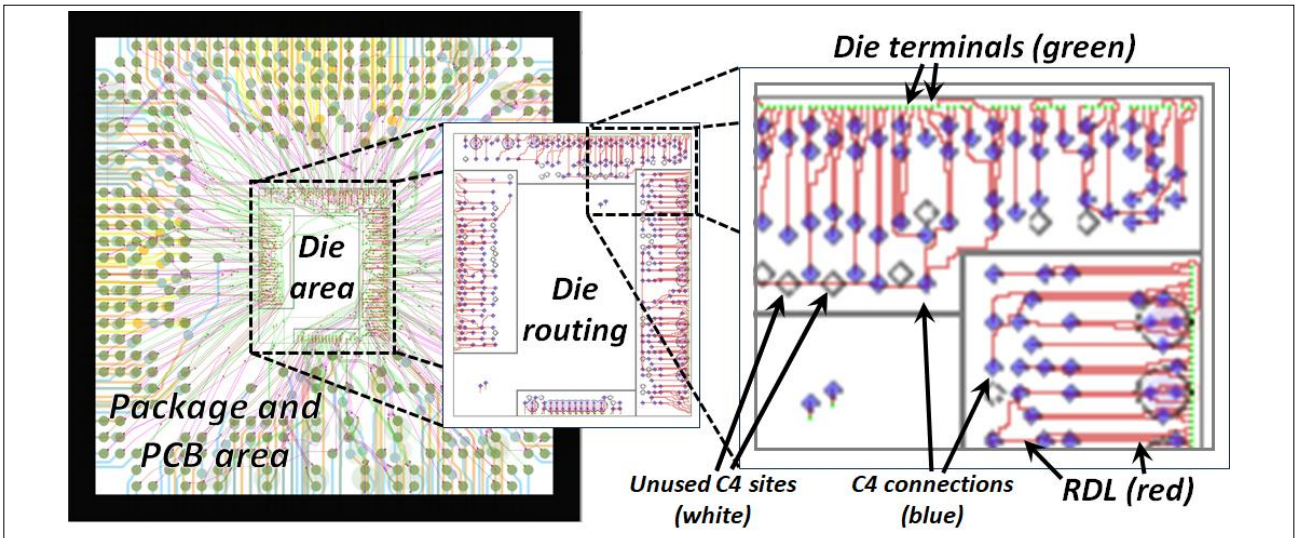

Figure 14: Lowest-cost routing configuration for a 356-net case using one RDL layer, five package routing layers, and three PCB layers. All nine layers are shown at left. Insets show the die-level terminals (green), routing (red) and the die-to-package C4 connections (blue). Some C4 sites were not used for routing (white).

4.3 Die, Package, PCB Routing

Simultaneous die, package, and PCB routing was tested by adding a die-level RDL layer to the case described in the previous section. 356 terminals of silicon I/O buffers were assigned to realistic locations on the die perimeter. Acorn routed RDL traces with a pitch of 16 µm to C4 sites, representing the die-to-package electrical interface. More die-to-package C4 sites were defined than this net-count to enable RDL traces to negotiate for the limited C4 waypoints.

Similar to Figure 5(b), all nets' PCB terminals for this test were located in pin-swap zones on the perimeter of the PCB. Acorn therefore attempted to find the shortest paths to this perimeter using the fewest vias without violating design rules. $D_{vertCost}$ was 5,000 µm for this run.

Acorn achieved 139 violation-free routing configurations during 1850 iterations over the course of 91 hours using 64 threads. Figure 14 shows the overall routing. For the package and PCB layers, the routing was qualitatively similar to the previous cases (Figures 12 and 13). The die-level routing is highlighted in the insets of Figure 14.

5 Conclusions

As a proof-of-concept, the algorithm described herein exhibits potential for co-design across the die, package, and PCB domains. It provides multiple, viable, routing configurations for industry-like design cases, albeit with run-times that extend to hours or even days. The solution proposed herein is not expected to provide an optimal solution within any given domain; its utility instead is its ability to span multiple domains without dependencies on other software tools – proprietary or otherwise.

Acknowledgements

The author wishes to thank former engineering colleagues at NXP Semiconductors for ideas that inspired and shaped the present work.

REFERENCES

[1] Mahendrasing Patil, Amit Brahme, Michael Shust, Keven Coates, Shubhada Thatte, Sreekanth Soman, Kamal Kumar, "Chip-package-board co-design for Complex System-on-Chip (SoC)," 19th Topical Meeting on Electrical Performance of Electronic Packaging and Systems, 2010, pp. 273-276.

[2] Humair Mandavia, Kazunari Koga, Ralf Brüning, Nikola Kontic, "System I/O Optimization with SoC, SiP, PCB Co-Design," 2015 European Microelectronics Packaging Conference (EMPC), Friedrichshafen, Germany, 2015, pp. 1-4.

[3] Thomas Brandtner, Klaus Pressel, Natalia Floman, Michael Schultz, Michael Vogl, "Chip/Package/Board Co-Design Methodology Applied to Full-Custom Heterogeneous Integration," 2020 IEEE 70th Electronic Components and Technology Conference (ECTC), 2020, pp. 1718-1727.

[4] Guy Theraulaz, Eric Bonabeau, "A Brief History of Stigmergy," Artificial Life. 1999 Spring, 5(2), pp. 97-116.

[5] Larry McMurchie, Carl Ebeling, "PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs," Third International ACM Symposium on Field-Programmable Gate Arrays, Napa Valley, CA, USA, 1995, pp. 111-117.

[6] Russell Tessier, "Negotiated A* Routing for FPGAs," Proceedings of The Fifth Canadian Workshop on Field-Programmable Devices, 1998 (FPD98).

[7] Pak K. Chan, Martine D.F. Schlag, "Acceleration of an FPGA Router," in 1997 IEEE Workshop on FPGAs for Custom Computing Machines, pp. 175-181, 1997.

[8] Pak K. Chan, Martine D.F. Schlag, "New Parallelization and Convergence Results for NC: A Negotiation-Based FPGA Router," FPGA 2000: pp. 165-174.

[9] Ting-Chou Lin, Devon Merrill, Yen-Yi Wu, Chester Holtz, Chung-Kuan Cheng, "A Unified Printed Circuit Board Routing Algorithm with Complicated Constraints and Differential Pairs," ASPDAC '21: Proceedings of the 26th Asia and South Pacific Design Automation Conference, January 2021, pp. 170–175.

[10] Joše Capela Dias, Penousal Machado, Daniel Castro Silva, Pedro Henriques Abreu, "An Inverted Ant Colony Optimization Approach to Traffic," Engineering Applications of Artificial Intelligence 00 (2014) pp. 1–20.

[11] Tri-Hai Nguyen, Jason J. Jung, "ACO-based Traffic Routing Method with Automated Negotiation for Connected Vehicles," Complex & Intelligent Systems (2023) 9, pp. 625–636.

[12] Jia-Wei Fang, Kuan-Hsien Ho, Yao-Wen Chang, "Routing for Chip-Package-Board Co-Design Considering Differential Pairs," 2008 IEEE/ACM International Conference on Computer-Aided Design, 2008, pp. 512-517.

[13] Jia-Wei Fang, Yao-Wen Chang, "Area-I/O Flip-Chip Routing for Chip-Package Co-Design," 2008 IEEE/ACM International Conference on Computer-Aided Design, San Jose, CA, USA, 2008, pp. 518-522.

[14] Jia-Wei Fang, Martin D. F. Wong, Yao-Wen Chang, "Flip-Chip Routing with Unified Area-I/O Pad Assignments for Package-Board Co-Design," 2009 46th ACM/IEEE Design Automation Conference, San Francisco, CA, USA, 2009, pp. 336-339.

[15] Jia-Wei Fang, Yao-Wen Chang, "Area-I/O Flip-Chip Routing for Chip-Package Co-Design Considering Signal Skews," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 29, no. 5, May 2010, pp. 711-721.

[16] Jeffrey McDaniel, Daniel Grissom, Philip Brisk, "Multi-terminal PCB Escape Routing for Digital Microfluidic Biochips using Negotiated Congestion," 2014 22nd International Conference on Very Large Scale Integration (VLSI-SoC), Playa del Carmen, 2014, pp. 1-6.

[17] Kenny Daniel, Alex Nash, Sven Koenig, Ariel Felner, "Theta*: Any-Angle Path Planning on Grids," Journal of Artificial Intelligence Research 39 (2010), pp. 533-579.

[18] Peter E. Hart, Nils J. Nilsson, Bertram Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," in IEEE Transactions on Systems Science and Cybernetics, vol. 4, no. 2, July 1968, pp. 100-107.

