Starfish: An Efficient P&R Co-Optimization Engine with A*-based Partial Rerouting

Fangzhou Wang, Lixin Liu, Jingsong Chen, Jinwei Liu, Xinshi Zang, Martin D.F. Wong
Department of Computer Science and Engineering, The Chinese University of Hong Kong
{fzwang, lxiu, jschen, jwliu, xszang}@cse.cuhk.edu.hk, mdfwong@cuhk.edu.hk

Abstract—Placement and routing (P&R) are two important stages in the physical design flow. After circuit components are assigned locations by a placer, routing will take place to make the connections. Defined as two separate problems, placement and routing aim to optimize different objectives. For instance, placement usually focuses on optimizing the half-perimeter wire length (HPWL) and estimated congestion while routing will try to minimize the routed wire length and the number of overflows. The misalignment between the objectives will inevitably lead to a significant degradation in solution quality. Therefore, in this paper, we present Starfish, an efficient P&R co-optimization engine that bridges the gap between placement and routing. To incrementally optimize the routed wire length, Starfish conducts cell movements and reconnects broken nets by A*-based partial rerouting. Experimental results on the ICCAD 2020 contest benchmark suites [1] show that our co-optimizer outperforms all the contestants with better solution quality and much shorter runtime.

I. INTRODUCTION

To solve the complex physical design problem, conventional physical design flow is divided into several stages, such as floorplanning, placement, clock tree synthesis, routing, etc. Among these, placement and routing are the most critical and intricate parts, which will affect the circuit’s power, performance, and area (PPA) significantly. However, the difficulty in optimizing placement towards some routing metrics while routing has not yet been done, can incur a significant degradation in solution quality. Besides, with the rapid growth of half-perimeter wire length (HPWL) and estimated congestion while routing has not yet been done, can incur a significant degradation in solution quality. Besides, with the rapid growth of the number of transistors, physical design at nano scale becomes much more complicated and the adverse effect of separating the two processes is magnified.

To bridge the gap between placement and routing, academic global placers [2]–[5] developed in recent years not only focus on optimizing the half-perimeter wire length (HPWL), but will also invoke a global router to obtain a congestion map to alleviate congestion. In order to further optimize the global placement solution in terms of routability, several post-processing approaches were proposed. GRPlacer [6] explores a weighted optimal region for each cell and formulates a multi-cell placement problem as a bipartite graph matching problem. The work [7] approximates the routing solution by rectilinear minimum spanning tree (RMST) and minimizes the corresponding congestion-driven routing cost.

However, both [6] and [7] mainly focus on optimizing circuit placement without a routing solution, the performance gap between placement and routing still fails to be mitigated directly. Different from [6] and [7], IPR [8] explicitly maintains a global routing solution generated by FastRoute [9]. IPR will evaluate the score of every possible destination to determine the next step movement of each cell in order to relieve congestion and reduce wire length. However, as each round of evaluation requires rerouting all the nets associated with the moved cell, IPR is rather time-consuming and is hard to deploy in practice.

With the everlasting demand for circuits of better performance, it is crucial to leverage the power of routing with cell movement in physical design to co-optimize placement and routing in a more unified and effective way.

To this end, we present Starfish\(^1\), a P&R co-optimization engine that minimizes the actual global routing wire length by cell movement and net rerouting. Meanwhile, all hard constraints under the routing with cell movement context are strictly followed (e.g., connectivity, min-routing-layer, overflow-free, etc). Our contributions can be summarized as follows.

- We propose an efficient multi-threaded P&R co-optimization engine, which integrates several novel techniques to effectively minimize the total routed wire length based on an initial P&R solution.
- Our engine integrates a rectilinear Steiner minimum tree (RSMT)-guided maze routing technique for both wire length and runtime reduction.
- We propose a lookup table-based approach for accurate wire length estimation, which takes several hard constraints (e.g., min-routing-layer, preferred routing direction) into account. With accurate point-to-point wire length estimation, a cell movement gain estimation scheme, which utilizes the existing net routing topologies, is proposed to pick good candidate destinations for cells. We further accelerate the estimation scheme by a median box-based pruning technique.
- In alignment with the proposed gain estimation scheme, we design a multi-source single-target A*-based partial rerouting algorithm to quickly connect a moved cell back to the trunk of a previous routing topology.
- Experimental results on the ICCAD 2020 contest benchmarks suite [1] show that our proposed P&R co-optimization engine outperforms all the contestants. In particular, compared with the champion of the contest, Starfish can achieve 0.9% better scores\(^2\) with 87% less runtime on average.

II. PRELIMINARIES

In the routing with cell movement problem [1], a bounded number of cells can be relocated to improve the original placement and routing solution such that the routed wire length can be further minimized without causing routing overflows. Figure 1 shows an

---

1 Starfish, or sea stars, are well recognized for their marvelously ability to regrow arms. Inspired by the similarity between arm regeneration and the process of reconnecting a moved cell back to the existing routing segments, we name the proposed engine Starfish.

2 In the contest setting, this can be considered as a significant gap, given that the differences in score between the first place and the other two top-3 participants are only 0.8% and 1.0% respectively on average.
example of routing with cell movement based on an existing placed-and-routed circuit. After cell relocation and net rerouting (shown in Figure 1(b)), a better P&R solution is achieved with shorter routed wire length. It is worth noting that, since this problem is not aimed at generating a totally different placement solution, the total number of cells that can be moved is limited (e.g. 30% of the total number of cells). In the following discussion, we denote \( C \) as the set of cells and \( N \) as the set of nets.

A. Coordinate Plane and Grid Graph

Coordinate plane and grid graph are two basic concepts in this problem, which represent the placement region and the global routing region respectively.

1) 2D Coordinate Plane: Given the number of rows \( \text{Row} \) and the number of columns \( \text{Col} \), the coordinate plane \( P \subseteq \mathbb{Z}^2 \) contains \( |P| = \text{Row} \times \text{Col} \) coordinates and serves as the placement region. Each cell \( v_i \in C \) locates at a specific 2D coordinates \( p_i \in P \), and each 2D coordinates \( p \in P \) can usually accommodate more than one cell as long as the constraints described in Section II-D are satisfied.

2) 3D Grid Graph: Similar to traditional 3D global routers [10]–[12], a 3D grid graph \( G(V,E) \) is constructed to represent the 3D routing region. The size of \( V \subseteq \mathbb{Z}^3 \) is given by \( |V| = \text{Row} \times \text{Col} \times \text{L} \), where \( \text{L} \) is the number of metal layers. Meanwhile, \( E \) is the set of edges showing the connectivity between Gcells. There are two types of edges between Gcells. One is the wire edge, which represents the intra-layer connection between two Gcells lying on the same layer and being adjacent in the preferred routing direction of that layer. The other one is the via edge, which represents the inter-layer connection between two adjacent Gcells located at the same 2D coordinate.

B. Net Model

In the routing with cell movement context, each net possesses its own sets of routing segments to connect the pins, and the routed wire length of a net is computed as the total number of Gcells traversed by its routing segments. In addition, the routing segments of each net must satisfy a min-routing-layer (MRL) constraint, which specifies a minimum 2D routing layer. The horizontal and vertical routing segments of a net \( n \in N \) must occur on or above its min-routing-layer \( \text{mr}l(n) \).

C. Overflow Gcell Definition

To define an overflow Gcell, we first introduce Gcell capacity. The capacity of a Gcell \( v \in V \) is defined as the difference between its supply and demand, denoted as \( \text{cap}(v) = \text{supply}(v) - \text{demand}(v) \). The term \( \text{supply}(v) \) is a given value that measures the available resources for cell placement and net routing, and \( \text{demand}(v) \) is the summation of cell blockage demand, cell extra demand and routing demand. The detailed formulation of \( \text{supply}(v) \) and \( \text{demand}(v) \) are given in [1], including how the cell extra demand is calculated. When the capacity of a Gcell \( v \) is smaller than zero, that is, \( \text{cap}(v) < 0 \), \( v \) is considered as an overflow Gcell.

D. Problem Formulation

Given a placed-and-routed circuit, the routing with cell movement problem is defined to minimize the total routed wire length by cell relocation and net rerouting. The final solution should satisfy the following constraints.

1) Max-Cell-Movement: The total number of cells that can be moved is limited by a constant \( \text{MaxCellMove} \).
2) Min-Routing-Layer: Routing segments of each net must occur on or above its min-routing-layer.
3) Connectivity: No open net exists.
4) Preferred Routing Direction: The intra-layer routing segments must follow the preferred routing direction.
5) Overflow-Free: No overflow Gcell exists.

Satisfying the aforementioned constraints, the score of the final output solution is defined as:

\[
\text{Score} = \text{TWL}_{\text{input}} - \text{TWL}_{\text{output}},
\]

where \( \text{TWL}_{\text{input}} \) and \( \text{TWL}_{\text{output}} \) denote the total wire length in the given input solution and the final output solution respectively.

III. PROPOSED ALGORITHMS

A. Overview

As Figure 2 shows, our proposed algorithm can be divided into three major parts, (1) pre-processing, (2) iterative multi-threaded cell movement, and (3) post-processing.

In the pre-processing stage, we will first read in a given overflow-free routing solution. As cycles and unnecessary wire segments may exist in this initial solution, a refinement step will first be conducted to generate a clean routing topology for each net.
After refining the initial solution, a congestion-driven rip-up and reroute (RRR) algorithm will be applied to greedily improve the routing solution, thus providing a better initial solution for the coming multi-threaded cell movement.

Before cell movements are performed, movable cells will first be partitioned into several batches, where cells in the same batch are independent from each other. Two cells are independent if they are not connected by any net. We then utilize a lookup table-based estimation scheme to calculate the cell movement gains and locate good candidate destinations for each movable cell.

In the iterative multi-threaded cell movement stage, cells will be moved and reconnected by an A*-based partial rerouting algorithm that aligns with the proposed estimation scheme. In order to conduct cell movements in parallel, cells in an independent batch will be further partitioned into several mini-batches in such a way that cells in the same mini-batch do not overlap in terms of placement and routing region.

After handling an independent batch of cells, movement gains will be updated for cells that have been successfully moved or have connections with some moved cells. Note that this gain update step can be delayed until after an independent batch of cell movements is finished and can be readily parallelized, since the relocation of one cell will not affect the cell movement gains of other cells that are independent with it. A net rerouting scheme will then be applied to further optimize nets that have been partially rerouted during the previous cell movement step.

One may wonder about the cell ordering, which can have a significant impact on the solution quality. Indeed, with the two-level batching strategy, cells are not explicitly ordered in our engine. Instead, we introduce a gain threshold $\tau$ to maintain a cell ordering implicitly. A cell movement will be committed only when its wire length gain is at least $\tau$. By starting with a relatively large threshold and gradually decreasing its value, cells with higher potential of wire length reduction can be prioritized.

In the post-processing stage, steps are taken to further reduce the routed wire length while ensuring that the output solution is legal. As the cell ordering for optimal P&R co-optimization is non-trivial, to achieve better results, we allow more than MaxCellMove cells to be moved in the cell movement stage, and design a scheme to carefully move cells that bring fewer gains back to their original locations. Finally, several rounds of wire length-driven rerouting will be performed to explore potential gains based on the final layout.

In Section III-B, we will discuss an RSMT-guided maze routing approach, which is adopted in all net rerouting stages except the partial rerouting after cell movement. In Section III-C, details about the proposed cell movement and partial rerouting scheme will be covered. After that, we introduce the post-processing techniques in Section III-D.

### B. RSMT-guided Maze Routing

In the proposed flow, we utilize maze routing to improve the existing routing solution under a fixed cell placement. The routing function will be called many times throughout the entire co-optimization process. For instance in Figure 2, this RSMT-guided maze routing will be invoked in the initial rerouting of the pre-processing stage, the selective net rerouting of the multi-threaded cell movement stage, and the wire length-driven rerouting of the post-processing stage. Therefore, an efficient method with good quality is essential.

Existing global routers like MGR [11] and CUGR [12] utilize multi-level routing to speed up the routing process, where a set of route guides will be generated to help pruning the routing solution space. With a similar idea in mind, we adopt an RSMT-guided maze routing approach, which is both faster and can achieve shorter wire length compared with a naive congestion-driven maze routing where routing is simply restricted in the net bounding box.

#### 1) Routing Region Generation: The guides will be generated based on a coarsen-grained grid graph. In our implementation, each block of 5 x 5 Gcells forms a coarsened grid. As Figure 3(a) shows, an RSMT topology will first be generated by FLUTE [13] in an instant according the 2D coordinates of the pins. For an edge whose two end-points are not on the same row or column, the RSMT does not tell which L-shape to use, and we can either go horizontally or vertically first. Therefore, the entire rectangular region that covers the two points will be considered as part of the routing region to provide more flexibility.

In this way, a smaller routing region like Figure 3(b) will be generated. It is worth noting that for the coming fine-grained maze routing, the region will be expanded vertically to form a 3D subgraph, where the routing layer range will be initially determined by the min-routing-layer and the pin locations of the net. Based on that, we will expand the layer range both upwards and downwards to increase the routing flexibility. As shown in Figure 3(c), with the help of route guides, a fine-grained routing solution with short wire length can be efficiently retrieved in a much smaller solution space.

#### 2) Cost Scheme: In the fine-grained maze routing, we assign a cost to each Gcell, taking into account congestion. The cost function $f(v)$ for a Gcell $v$ is described by

$$f(v) = 1 + \frac{con(v) + op(v)}{\alpha},$$

$$con(v) = \frac{1 + exp(cap(v))}{1},$$

$$op(v) = inf x \mathbb{1}(cap(v) \leq 0),$$

where $\mathbb{1} \in \{0, 1\}$ is an indicator function. We can see that $f(v)$ consists of three parts: the wire length cost (which is always one), the congestion cost, and the overflow penalty. The congestion cost $con(v)$ is designed to (1) increase rapidly when the capacity is close to zero and (2) incur little cost when the routing resource is abundant. Here $\alpha$ is a constant corresponding to the weight of the congestion cost. Lastly, if $cap(v) \leq 0$, that is, overflow has occurred or adding just one more demand will cause overflow, a large penalty will be invoked by $op(v)$.

#### 3) Greedy Rip-up and Reroute: Based on the RSMT route guides and the cost scheme, a greedy rip-up and reroute (RRR) is implemented to improve an existing routing solution in terms of congestion and wire length. The greedy RRR process is described as follows. Nets will first be sorted by the HPWL in ascending order. To

---

**Fig. 3:** Illustration for RSMT-guided maze routing. (a) An RSMT is generated according to the 2D layout by FLUTE [13]. (b) Route guides are generated according to the Steiner tree topology. (c) Fine-grained maze routing is conducted in the pruned solution space.
guarantee an overflow-free solution, nets will be rerouted one by one. The new routing solution of a net will be accepted if no overflow occurs and the corresponding wire length is shorter than the old one.

C. Cell Movement with A*-based Partial Rerouting

In this section, a multi-threaded cell movement scheme is proposed. In this process, we need to identify the cells to be moved and their potential destinations. To achieve this, we devised a lookup table-based approach to estimate quickly wire length gain after cell relocation utilizing the existing net routing segments, and an A*-based partial rerouting algorithm for fast routing tree reconstruction.

1) Lookup Table-based Wire Length Estimation: We first explain how routed wire length can be accurately measured by a lookup table-based approach under the min-routing-layer and preferred routing direction constraints. For two points $p_1 = (r_1, c_1, l_1)$ and $p_2 = (r_2, c_2, l_2)$, without any constraint, as shown in Figure 4(a), the routing distance between them can roughly be estimated as the summation of $|r_1 - r_2|$, $|c_1 - c_2|$, and $|l_1 - l_2|$. However, when the two routing layer-related constraints are involved, the calculation becomes less straightforward. Figure 4(b) shows one shortest routing path to connect a two-pin net when $mrl = 2$. Starting from A, the route needs to reach M2 first because of the min-routing-layer constraint. Vertical in-layer routing will then be conducted. Due to the existence of preferred routing directions, there must be an additional pair of up-and-down movements in order to conduct horizontal routing on M3. Calculation of such minimum routing distance may look complicated and can be affected by various factors.

Despite that, the routing distance can still be estimated efficiently with the help of a pre-calculated lookup table. We denote the minimum routing distance (MRD) needed to connect two points, $p_1$ and $p_2$, with a minimum routing layer $mrl$ by a function $mrd(p_1, p_2, mrl)$. In Figure 4(b), we can see that when calculating $mrd(p_1, p_2, mrl)$, $|\Delta r| = |r_1 - r_2|$ and $|\Delta c| = |c_1 - c_2|$ always contribute to the total routing distance. It is the capturing of via usage under different situations that makes the routing distance estimation complicated. An accurate estimation of via usage can be obtained with the following information: (1) whether the two points are on the same row, (2) whether the two points are on the same column, (3) the layer of $p_1$, (4) the layer of $p_2$, (5) the $mrl$ value, and (6) the preferred routing direction for each layer in the current design. A via-usage lookup table $vlut \in \mathbb{N}^{2 \times 2 \times 2 \times 2 \times 2}$ can then be pre-calculated in $O(L^5)$ time by enumerating all the possibilities. Since $L$ can be at most 16 in all the given cases, the space and preprocessing overhead is negligible.

With the above definitions, the formulation of $mrd$ is as follows,

$$mrd(p_1, p_2, mrl) = |\Delta r| + |\Delta c| + vlut(zero(\Delta r), zero(\Delta c), l_1, l_2, mrl), \quad (3a)$$

$$zero(x) = \begin{cases} 1 & \text{if } x = 0, \\ 0 & \text{if } x \neq 0. \end{cases} \quad (3b)$$

Back to the case in Figure 4(b), the $mrd$ value can be quickly computed as $[1 - 2] + [1 - 4] + vlut(0, 0, 1, 2, 2) = 1 + 3 + 3$.

2) Cell Movement Gain Estimation: With accurate point-to-point wire length estimation, we further discuss the cell movement gain estimation that utilizes the existing net routing topologies, which will be critical for selecting a set of good candidate destinations for the placement optimization purpose.

For a multi-pin net $n$, let $C_n$ be the set of cells connected by $n$ and $G(n)$ be the set of visited Gcells in the current routing solution.

![Fig. 4: Illustration for the minimum routing distance (MRD) calculation. (a) Without any constraints, MRD can be estimated as the summation of $|\Delta r| + |\Delta c| + |\Delta l|$. (b) With the preferred routing direction and $mrl$, $|\Delta r| + |\Delta c|$ can be one invariant in the complicated MRD calculation.](image)

![Fig. 5: Illustration for the remaining segments and the removed segments. Note that the remaining segments of Net 1 and Net 2 both form a single connected component.](image)

Under the circumstance that one of its cells $i \in C_n$ is moved, we denote the remaining segments (RS) as $rs(n,i)$, which is the smallest subset of $G(n)$ such that all the cells in $C_n \setminus i$ are still connected without violating any routing constraints. Figure 5 gives an illustration for the remaining segments when cell $C$ is moved. Further, we denote the removed routing segments due to the cell movement as $rm(n,i) = G(n) \setminus rs(n,i)$. In this way, the rest of the net $rs(n,i)$ can still form a single connected component, and the routing tree can be easily reconstructed by connecting the moved cell to any one of the Gcells in $rs(n,i)$. Note that both $rs(n,i)$ and $rm(n,i)$ can be quickly retrieved by a search on the routing tree, starting from the Gcell containing the pin of the moved cell $i$.

Next we (1) conduct the cell movement gain estimation and (2) select candidate destinations for each cell. Consider a single cell $i$, let $N_i$ be the set of nets associated with cell $i$. As Algorithm 1 shows, for gain estimation, we first get a rectangular candidate region (denoted as $candi Region$ in line 1) where the wire length gains will be calculated for each location $(r, c)$ in $candi Region$ and stored in a gain map denoted as $gain Map$. The gain map will be initialized to zero at the beginning (line 2) and updated from time to time. We will later discuss how this candidate region can be obtained.

As line 3 shows, we will iterate through all the nets in $N_i$ and accumulate the cell movement gains for each location $(r, c)$ in $gain Map$ (line 25). When handling a net $n$, we first find the pin in $n$ that belongs to cell $i$ to get the layer information of the point to be connected (line 4-5). With the remaining segments $rs(n,i)$ and removed segments $rm(n,i)$ retrieved from the routing tree of $n$, the wire length reduction $wlcm$ due to the removal of cell $i$ will simply be the size of $rm(n,i)$ (line 6-8). In line 9-11, the boolean variable $same Loc$ indicates if all the Gcells in the remaining segments have the same 2D coordinate, which can make a significant difference in
Algorithm 1 Cell Movement Gain Estimation

Input: Cell i, Set of nets \( N_i \) associated with cell i.
Output: A map \( \text{gainMap} \) recording the estimated gains when cell i is moved to different locations.
1: \( \text{candiRegion} \leftarrow \text{getCandidateRegion}(i); \)
2: Initialize each entry of \( \text{gainMap} \) to 0;
3: foreach \( n \in N_i \) do
4: \( \text{Let } p_{in} \in n \text{ be a pin of cell } i; \)
5: \( l \leftarrow \text{getLayer}(p_{in}); \)
6: \( r_s(n, i) \leftarrow \text{getRemainingSegments}(n, i); \)
7: \( r_m(n, i) \leftarrow \text{getR\_ RemovedSegments}(n, i); \)
8: \( \text{wl}_{\text{rem}} \leftarrow \text{getR\_ RemovedSegments}(n, i); \) // Get removed wire length.
9: \( \text{sameLoc} \leftarrow \text{true}; \)
10: if \( r_s(n, i) \) contains Gcells with different 2D coordinates then
11: \( \text{sameLoc} \leftarrow \text{false}; \) // For higher estimation accuracy.
12: foreach \( dest = (r, c) \in \text{candiRegion} \) do
13: \( \text{Let } p = (r, c, l); \)
14: \( \text{cost}_{\text{min}} \leftarrow \infty; \)
15: \( \text{foreach } \text{Gcell } p' = (r', c', l') \in rs(n, i) \) do
16: if \( r = r' \) and \( c = c' \) and \( \text{sameLoc} = \text{false} \) then
17: if \( l < \text{mrl}(n) \) and \( l' < \text{mrl}(n) \) then
18: \( \text{cost}_{\text{cur}} \leftarrow \min(l' - l, 0); \)
19: \( \text{cost}_{\text{min}} \leftarrow \min(\text{cost}_{\text{min}}, \text{cost}_{\text{cur}}); \)
20: else
21: \( \text{cost}_{\text{min}} \leftarrow \min(\text{cost}_{\text{min}}, \text{mrd}(p, p', \text{mrl}(n))); \)
22: else
23: \( \text{cost}_{\text{min}} \leftarrow \min(\text{cost}_{\text{min}}, \text{mrd}(p, p', \text{mrl}(n))); \)
24: \( \text{wl}_{\text{gain}} \leftarrow \text{wl}_{\text{rem}} - \text{cost}_{\text{min}}; \)
25: \( \text{gainMap}[r][c] \leftarrow \text{gainMap}[r][c] + \text{wl}_{\text{gain}}; \)
26: return

Algorithm 2 Candidate Destination Selection

Input: Cell i, Gain estimation map \( \text{gainMap} \) for cell i.
Output: Candidate destinations with movements gain for cell i.
1: \( \text{candiDest} \leftarrow \{\}; \) // A collection of (gain, destination) pairs.
2: \( \text{candiRegion} \leftarrow \text{getCandidateRegion}(i); \)
3: foreach \( dest = (r, c) \in \text{candiRegion} \) do
4: if \( \text{gainMap}[r][c] > 0 \) then
5: \( \text{candiDest}.\text{append}(gain, dest); \)
6: Sort \( \text{candiDest} \) by \( \text{gain} \) in descending order.
7: return

the later estimation process.

In line 12-24, we calculate the wire length gain of net \( n \) when the cell is moved to different positions in \( \text{candiRegion} \). Consider a possible destination \( (r, c) \), the corresponding point to be connected back to \( rs(n, i) \) will be \( p = (r, c, l) \) (line 13). We then iterate through all the Gcells \( p' \in rs(n, i) \) to find the minimum wire length needed for the reconnection (line 15-23). In most cases, the minimum value can be easily updated with the \( \text{mrd} \) function, which is based on a lookup table (line 23). However, some special case handling is needed for higher estimation accuracy (line 16-19) as explained below.

An example is shown in Figure 6. In this example, we consider moving cell A to Gcell p and try to estimate the minimum routing distance for the reconnection. This is a case that the destination position overlaps with part of the remaining segments in 2D. In the presence of min-routing-layer M3, if we simply measure the \( \text{mrd} \) between p and each \( q_k \) in the remaining segments, \( \text{mrd}(p, q_k, 3) = 2 \) will be considered as the minimum wire length needed to connect pin A with the remaining parts. However, the actual additional wire length needed is 1 only, which corresponds to the usage of Gcell p. The over-estimation by \( \text{mrd}(p, q_k, 3) \) is resulted from the ignorance of the fact that Gcell \( q_2 \) has already been included in the remaining segments when we are considering \( (p, q_i) \) pair. The checkings and steps in line 16-19 can help to improve the estimation accuracy under similar situations, which is essential as the min-routing-layer can be as high as ten in some benchmarks.

After cell movement gain estimation and updates to gain map, we can conduct candidate destination selection as shown in Algorithm 2. For each position in the candidate region, if the estimated wire length gain is greater than zero, it will be added to the candidate destination collection \( \text{candiDest} \) along with its gain (line 3-6). Sorting will be conducted based on the gains such that positions with higher potential of reducing wire length will be tried first.

3) Candidates Pruning by Median Box: At the beginning of this cell movement gain estimation stage, we will locate the candidate region. For efficiency, it is not good to take the whole \( R \times C \) layout as the candidate region. Consider current cell i, let \( N_i \) be the set of nets connecting to it, and \( C_i \) be the set of cells connected to i. An intuitive idea will be to consider the smallest bounding box that covers all the cells in \( C_i \) and cell i itself as the region, which will be referred to as the cell bounding box. However, with this approach, the bounding box can still be large when \( C_i \) spans a wide area, which dramatically slows down the estimation process.

Therefore, we introduce a median box-based approach to generate candidate region, which effectively prunes the total number of candidates without affecting the solution quality. In the proposed approach, an expanded optimal region will be taken as a candidate region, which is defined based on a concept called optimal region (OR). We define optimal region for cell i (\( OR_i \)) as the set of locations that minimize \( \sum_{n \in N_i} \text{hpwl}(n) \) when cell i is being moved. Here \( \text{hpwl}(n) \) refers to the half-perimeter wire length of net n. As mentioned in [14], \( OR_i \) can be computed based on the median of the bounding box boundaries of the nets in \( N_i \). For a net \( n \in N_i \), we denote the lower-left and upper-right corner of its bounding box with cell i removed as \( (r_1(n), c_1(n)) \) and \( (r_2(n), c_2(n)) \) respectively. Let \( Rs = \{r_1(n_1), r_2(n_1), r_1(n_2), r_2(n_2), ..., r_1(n_{|N_i|}), r_2(n_{|N_i|})\} \) be an array of all these row indices. Similarly, we have an array \( Cs \) storing the column indices. \( OR_i \) will then be the rectangular region with \( (r_1^{opt}, c_1^{opt}) \) as its lower-left corner and \( (r_2^{opt}, c_2^{opt}) \) as its upper-right corner, where \( r_1^{opt}, c_1^{opt}, r_2^{opt}, c_2^{opt} \) are the medians of \( Rs \) and \( Cs \) respectively. Note that both \( Rs \) and \( Cs \) will have an even number of elements. It is possible that \( r_1^{opt} = r_2^{opt} \) or \( c_1^{opt} = c_2^{opt} \). In that case, \( OR \) will be a single line or even a single point. Figure 7(a) shows the optimal region for a single cell A in a case that there are only two nets \{A, B, E\}. A\{A, C, D\} containing A. With cell A excluded, the optimal region (green region) can be calculated using the bounding boxes of the two nets. However, when all the positions in \( OR_i \) are congested, moving the cell to a location nearby may still
lead to a considerable amount of wire length reduction. As shown in Figure 7(b), by expanding the optimal region by a margin, an expanded optimal region is obtained and will serve as the candidate region. In Section IV, we will show the effectiveness of this pruning technique and explore a suitable width for the candidate margin.

4) A*-based Partial Rerouting: With a set of good cell movement candidate destinations, routing is needed to rebuild net connection and verify if the cell can be successfully moved to the new location, that is, no overflow is caused and the actual wire length gain through this relocation is large enough. In alignment with the proposed estimation scheme, and to quickly connect a cell back to the remaining segments of its associated nets, a multi-source single-target A*-based partial rerouting algorithm is proposed (Algorithm 3). Consider a net $n$, given its remaining segments $rs$, and a pin to be reconnected $p = (r, c, l)$, we try to find a least-cost path from $rs$ to pin $p$. In the algorithm, we assume that $p \not\in rs$, or else the remaining segments can be directly used as the new routing solution. As two critical elements in an A* search, the routing cost and the heuristic cost of a Gcell $v$ are modeled by $f(v)$ from Equation (2) and $mrd(v, p, mrl(n))$ from Equation (3a) respectively.

In the proposed A* search, we maintain a map $\text{cameFrom}$ for routing path traceback, a map $r\text{Score}$ for the minimum routing cost from any point to $rs$, and a map $f\text{Score}$ for the heuristic score based on function $f$ and $mrd$ (line 1-4). We treat all the Gcells in $rs$ as sources, updating the corresponding entries in $r\text{Score}$ and $f\text{Score}$ (line 5-8), and conduct a typical A* search (line 9-22). Once the pin $p$ is found, we construct the routing solution following the $\text{cameFrom}$ map (line 11-13). Routing failure will be reported if the pin cannot be reached due to the preferred routing direction or min-routing-layer constraints (line 23-24). Since the given placement has been optimized globally, cells will not be moved far away from its original location in most of the cases. Therefore, the proposed path searching algorithm can be highly efficient in this routing with cell movement context.

With the proposed A*-based partial rerouting algorithm, the overall flow of a cell movement is described in Algorithm 4. Before we move a cell $i$, the total wire length of the previous routing solution will be recorded first (line 1). We then rip up the cell from the current placement (line 2). To reclaim part of the resource used by the previous routing solution and prepare for the future partial rerouting, the remaining and removed segments will be retrieved for each net associated with cell $i$ (line 3-6). After that, we will consider every position $\text{dest}$ in the pre-calculated candidate destinations $\text{candiDest}_i$, starting from the one with the largest $\text{gain}$ (line 8-18). After partial rerouting, the actual gain in wire length $\text{gain}_i$ will be calculated as the total wire length change before and after the cell movement (line 12-14). A cell movement will be accepted when there is no overflow and $\text{gain}_i$ is greater than or equal to the gain threshold (line 15-18). If the cell movement fails for all candidate destinations, the cell will be put back to its original position with related routing solution restored (line 19-21). Note that in line 9-10, a pruning based on the estimated gain is conducted to speed up the process without affecting the solution quality. As the cell movement estimation scheme does not consider routing congestion, $\text{gain}$ is an upper bound of the actual wire length gain $\text{gain}_i$. Therefore, if $\text{gain} < \tau$, the cell movement will be rejected immediately. Since the

Algorithm 3 A*-based Partial Rerouting

**Input:** Net to be partially rerouted $n$, Remaining segments $rs$. To be connected pin $p = (r, c, l)$.

**Output:** A new routing solution for net $n$.

1. $\text{openSet} \leftarrow \{\}$; $\text{cameFrom} \leftarrow$ empty map; $r\text{Score} \leftarrow$ map with all $\infty$; $f\text{Score} \leftarrow$ map with all $\infty$; $\text{Heuristic score based on } f$ and $mrd$.
2. $\text{openSet}.add(p')$; $r\text{Score}(p') \leftarrow 0$; $f\text{Score}(p') \leftarrow mrd(p', p, mrl(n))$.
3. *foreach* $Gcell p' \in rs$ *do*

   4. $\text{openSet}.add(p')$; $\text{Treat all Gcells in } rs$ as sources.
5. $r\text{Score} = \text{rScore}_\text{new}$.
6. $f\text{Score} = \text{fScore}_\text{new}.$
7. if $r\text{Score}_\text{new} < \text{rScore}_\text{old}$ then
6. $\text{cameFrom}(u) \leftarrow w$;
8. $\text{rScore}(v) \leftarrow \text{rScore}_\text{new};$ $\text{Update the minimum cost.}$
9. $\text{fScore}(v) \leftarrow \text{fScore}_\text{new} + mrd(v, p, mrl(n))$;
10. *if* $v \notin \text{openSet}$ *then*
11. $\text{return}$
12. *Construct the routing solution with \text{cameFrom};*
13. *return*
14. *foreach* neighbor $v$ of $u$ *do*
15. $r\text{Score}_\text{new} \leftarrow r\text{Score}(u) + f(v);$ $f(v)$: cost for a Gcell.
16. if $r\text{Score}_\text{new} < r\text{Score}_\text{old}$ then
17. $\text{cameFrom}(v) \leftarrow w$;
18. $\text{rScore}(v) \leftarrow r\text{Score}_\text{new};$ $\text{Update the minimum cost.}$
19. $\text{fScore}(v) \leftarrow \text{fScore}_\text{new} + mrd(v, p, mrl(n))$;
20. *if* $v \notin \text{openSet}$ *then*
21. *return*
22. *openSet.add(v);
23. *Report routing failure;*
24. *return*

Algorithm 4 Single Cell Movement

**Input:** Cell $i$, Set of nets $N_i$ associated with cell $i$, Sorted candidate destinations with movement gains $\text{candiDest}_i$, Gain threshold $\tau$.

**Output:** New placement for cell $i$ and update routing solutions.

1. $\text{wl}_{\text{old}} \leftarrow \sum_{n \in N_i} \text{wl}(n);$ $\text{Original total wire length.}$
2. Rip up cell $i$ from current placement result;
3. *foreach* net $n \in N_i$ *do*
4. $\text{rs}(n, i) \leftarrow \text{getRemainingSegments}(n, i);$
5. $\text{rm}(n, i) \leftarrow \text{getRemovedSegments}(n, i);$
6. Rip up the routing resource usage by rm$(n, i);$
7. *moveSuccess* $\leftarrow$ false;
8. *foreach* $(\text{gain}, \text{dest}) \in \text{candiDest}_i$ *do*
9. *if* $\text{gain} < \tau$ *then *
10. Break out of the loop;
11. Move cell $i$ to $\text{dest}$;
12. Perform partial rerouting for all $n \in N_i$, store the results in $N'_i$;
13. $\text{wl}_{\text{new}} \leftarrow \sum_{n' \in N'_i} \text{wl}(n')$;
14. $\text{gain}_i \leftarrow \text{wl}_{\text{old}} - \text{wl}_{\text{new}}$;
15. *if* $\text{gain}_i \geq \tau$ and no overflow occurs *then*
16. Accept the cell movement and update routing solution;
17. *moveSuccess* $\leftarrow$ true;
18. *Break*;
19. *if* moveSuccess $= \text{false}$ *then*
20. *Put cell* $i$ back to its previous location;
21. *Recover old routing solution for all* $n \in N_i$;
22. *return*
TABLE I: Experimental Results on the ICCAD 2020 Benchmark

<table>
<thead>
<tr>
<th>Case ID</th>
<th>TWLinput</th>
<th>Score</th>
<th>RT (s)</th>
<th>Case ID</th>
<th>TWLinput</th>
<th>Score</th>
<th>RT (s)</th>
<th>Case ID</th>
<th>TWLinput</th>
<th>Score</th>
<th>RT (s)</th>
<th>Case ID</th>
<th>TWLinput</th>
<th>Score</th>
<th>RT (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>case3</td>
<td>32600</td>
<td>11425</td>
<td>34</td>
<td>case4</td>
<td>4680681</td>
<td>204681</td>
<td>222</td>
<td>case5</td>
<td>1763627</td>
<td>695219</td>
<td>903</td>
<td>case6</td>
<td>7188481</td>
<td>272127</td>
<td>311</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>case4B</td>
<td>4886698</td>
<td>218257</td>
<td>2299</td>
<td>case5B</td>
<td>1721530</td>
<td>664347</td>
<td>886</td>
<td>case6B</td>
<td>7340802</td>
<td>274809</td>
<td>3406</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.000</td>
<td>1.000</td>
<td></td>
<td>0.992</td>
<td>0.368</td>
<td>0.990</td>
<td>2.224</td>
<td></td>
<td>1.000</td>
<td>0.133</td>
<td>0.397</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Avg. ratio = 1.000

* The score and runtime statistics of the top-3 winners are provided by the contest organizer with Intel Xeon E7-4820 CPU (2.00 GHz, 8 cores) and 128 GB memory. TWLinput denotes the total wire length in the given input solution. RT is short for runtime. $R_{diff} = \frac{Score}{TWL_{input}}$. The average ratio for score is obtained by taking the average of normalized scores against the 1st place, same for the runtime.

5) Selective Net Rerouting: In Figure 5, we can see that in the original routing solution of Net 2, a C-shaped detour was made to connect cell C. After the relocation of cell C, this unnecessary detour, still exists in the new routing solution as it was treated as parts of the remaining segments. To handle such cases, as shown in Figure 2, we will conduct one round of selective net rerouting, which only selects the nets associated with the successfully moved cells in the previous round of cell movement, and performs greedy RRR (end of Section III-B) for them. In this way, new routing topologies can be generated if the existing ones can be further improved.

D. Post-processing with Greedy Cell Put-back

After the multi-threaded cell movement process, several post-processing techniques are applied to further reduce the total routed wire length while ensuring a legal output solution. In the proposed flow, we allow more than $MaxCellMove$ cells to be moved. The total number of moved cells exceeds $MaxCellMove$, a greedy cell put-back process will be applied to carefully move cells that bring fewer gains back to their original locations without causing overflow. This is done as follows. We will start with a gain threshold (1 in our implementation), and try to put each moved cell back to its initial position. Note that the gain threshold is negative because the wire length is expected to increase in this put-back process. However, if the wire length increases by too much, that is, the wire length gain is smaller than the gain threshold, or overflow occurs, we will not move a cell back to its original position to avoid degrading the solution quality for too much.

After iterating through all the cells once, the gain threshold will be reduced to allow more cells to move back in the next iteration. This process will stop once the total number of moved cells is no greater than $MaxCellMove$. After this cell put-back process, several rounds of wire-length-driven rerouting, which is essentially the aforementioned greedy RRR with little congestion cost, will be performed to further reduce the total wire length.

IV. EXPERIMENTAL RESULTS

Our P&R co-optimization engine is implemented in the C++ language and all the experiments were conducted on a Linux machine with 2.90 GHz Intel Xeon CPU and 756 GB memory. Consistent with the contest, eight threads are used by default. We empirically test the proposed flow on the ICCAD 2020 contest benchmark suites [1] and compare our results with the top-3 winners. Ablation studies are also conducted to show the effectiveness of our proposed techniques with respect to (1) solution quality and (2) runtime.

A. Comparison with the Top-3 Winners

The benchmark consists of ten cases, including two toy cases (case1, case2). In Table II, statistics of each cases are shown, including the number of cells, the number of nets, the maximum number of cells that can be moved, and the size of the layout.

TABLE II: Benchmark Statistics

<table>
<thead>
<tr>
<th>Case ID</th>
<th>#Cells</th>
<th>#Nets</th>
<th>MCM</th>
<th>#Cells ($Row \times Col \times L$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>case1</td>
<td>8</td>
<td>6</td>
<td>2</td>
<td>5×5×3</td>
</tr>
<tr>
<td>case2</td>
<td>6</td>
<td>6</td>
<td>3</td>
<td>4×4×3</td>
</tr>
<tr>
<td>case3</td>
<td>2735</td>
<td>2644</td>
<td>820</td>
<td>27×33×7</td>
</tr>
<tr>
<td>case4</td>
<td>204206</td>
<td>179996</td>
<td>61261</td>
<td>277×277×12</td>
</tr>
<tr>
<td>case5</td>
<td>96682</td>
<td>92546</td>
<td>29004</td>
<td>104×103×16</td>
</tr>
<tr>
<td>case6</td>
<td>352269</td>
<td>332080</td>
<td>105680</td>
<td>237×236×16</td>
</tr>
<tr>
<td>case3B</td>
<td>2604</td>
<td>2563</td>
<td>781</td>
<td>29×29×7</td>
</tr>
<tr>
<td>case4B</td>
<td>207347</td>
<td>183137</td>
<td>62204</td>
<td>277×277×12</td>
</tr>
<tr>
<td>case5B</td>
<td>96689</td>
<td>92559</td>
<td>29006</td>
<td>104×103×16</td>
</tr>
<tr>
<td>case6B</td>
<td>352234</td>
<td>332045</td>
<td>105670</td>
<td>237×236×16</td>
</tr>
</tbody>
</table>

* MCM denotes the $MaxCellMove$.

Quantitative results are presented in Table I. Column “Score”, as formulated in Equation (1), measures the wire length improvement between the given input solution and the final output solution. Column “$R_{diff}$”, defined as $R_{diff} = \frac{Score}{TWL_{input}} = 1 - \frac{TW_{input}}{TW_{output}}$, denotes the total wire length reduction rate. From the table, we can observe that our proposed co-optimizer can effectively reduce the total routed wire length based on a given P&R solution. Compared with the ICCAD 2020 contest winners, our co-optimizer stands out with better scores and much shorter runtime in all the cases. Specifically, compared with the first place, we achieves 0.9% better scores with 87% less runtime on average. In the contest setting, this can be considered as a significant gap, given that the differences in score between the first place and the other two top-3 participants are only 0.8% and 1.0% respectively on average. Moreover, results from column “$R_{diff}$” shows that with the proposed routing with cell movement scheme, our co-optimizer can achieve 39.7% total wire length reduction on average.
In this paper, we propose Starfish, an efficient P&R co-optimization engine, to bridge the gap between placement and routing. We demonstrate that the proposed flow can effectively reduce the total routed wire length by routing with cell movement. Experimental results on the ICCAD 2020 benchmark suites show that, with an accurate cell movement gain estimation scheme and several efficient routing algorithms, our co-optimizer outperforms all the contestants with better solution quality and much shorter runtime.

V. CONCLUSIONS

In this paper, we propose Starfish, an efficient P&R co-optimization engine, to bridge the gap between placement and routing. We demonstrate that the proposed flow can effectively reduce the total routed wire length by routing with cell movement. Experimental results on the ICCAD 2020 benchmark suites show that, with an accurate cell movement gain estimation scheme and several efficient routing algorithms, our co-optimizer outperforms all the contestants with better solution quality and much shorter runtime.

VI. ACKNOWLEDGEMENT

The authors would like to thank Prof. Evangeline F.Y. Young for her guidance and countless valuable discussions.
REFERENCES


