A Fast Optimal Double-row Legalization Algorithm

STEFIN HOUGARDY and MEIKE NEUWOHNER, Research Institute for Discrete Mathematics, University of Bonn, Germany
ULRIKE SCHORR, Cadence Design Systems Inc., Germany

In Placement Legalization, it is often assumed that (almost) all standard cells possess the same height and can therefore be aligned in cell rows, which can then be treated independently. However, this is no longer true for recent technologies, where a substantial number of cells of double- or even arbitrary multiple-row height is to be expected. Due to interdependencies between the cell placements within several rows, the legalization task becomes considerably harder. In this article, we show how to optimize squared cell movement for pairs of adjacent rows comprising cells of single- as well as double-row height with a fixed left-to-right ordering in time $O(n \cdot \log(n))$, where $n$ denotes the number of cells involved. Opposed to prior works, we do not artificially bound the maximum cell movement and can guarantee to find an optimum solution. Our approach also allows us to include gridding and movebound constraints for the cells. Experimental results show an average percental decrease of over 26% in the total squared movement when compared to a legalization approach that fixes cells of more than single-row height after Global Placement.

CCS Concepts: • Hardware → Placement;

Additional Key Words and Phrases: Placement, legalization, double-row-height cells

ACM Reference format:
Stefan Hougardy, Meike Neuwohner, and Ulrike Schorr. 2023. A Fast Optimal Double-row Legalization Algorithm. ACM Trans. Des. Autom. Electron. Syst. 28, 5, Article 71 (September 2023), 26 pages. https://doi.org/10.1145/3579844

1 INTRODUCTION

The Standard Placement Problem captures the task of locating hundreds of thousands or even millions of standard cells, which are usually assumed to exhibit uniform heights, within the rectangular chip area. In doing so, multiple objectives such as minimizing the total length of inter-cell electrical connections (nets) or achieving desirable timing properties have to be respected. Given the fact that even the underlying packing problem is strongly NP-hard [9], the placement task is most commonly split into the three sub-problems of Global Placement, Legalization, and Detailed Placement. Global Placement aims at finding cell locations that approximately minimize the total netlength for a certain net model and obey bounds on local packing density, but does not have to ensure internal disjointness of shapes. The Legalization step deals with resolving the remaining

This is an extended version of Reference [14], which appeared in the Proceedings of the ISPD’21.
Authors’ addresses: S. Hougardy and M. Neuwohner, Research Institute for Discrete Mathematics, University of Bonn, Lennéstr. 2, Bonn, Germany, 53113; emails: {hougardy, neuwohner}@or.uni-bonn.de; U. Schorr, Cadence Design Systems Inc., Mozartstr. 2, Munich, Germany, 85622; email: uschorr@cadence.com.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
1084-4309/2023/09-ART71 $15.00
https://doi.org/10.1145/3579844

ACM Transactions on Design Automation of Electronic Systems, Vol. 28, No. 5, Article 71. Pub. date: September 2023.
overlaps by shifting cells locally, trying to minimize either netlength or the total (squared) cell displacement. Detailed Placement usually incorporates several post-optimization routines.

Optimizing squared instead of linear cell movement during the Legalization step is desirable for several reasons. First, unlike linear cell displacement, squared cell displacement is a strictly convex function. This implies that there is a unique optimum (ignoring gridding constraints), whereas a minimum linear cell displacement can be attained by multiple placements (see Figure 1). Furthermore, if each cell is assigned a weight, that, for example, reflects its timing criticality, then the solution minimizing weighted squared cell displacement is continuous as a function of the weights (again ignoring gridding constraints). This is not the case for weighted linear displacement, even if the optimum happens to be unique. For example, if in the situation of Figure 1 the left cell is assigned a weight of $1 - \epsilon$ and the right cell receives a weight of $1 + \epsilon$, then the left solution is the unique minimizer of weighted linear displacement, whereas if the weights are swapped, the right solution is the unique optimum. In contrast to this, when optimizing squared cell movement, the optimum solutions arise from the middle one by only slightly shifting it to the left or right, respectively. Thus, optimizing squared cell movement fosters stability, which is an important feature, especially as placement is an early step in the design flow. Moreover, while the minimization of total linear cell movement is oblivious to the distribution of the displacements among the cells, optimizing squared cell displacement balances the displacements of different cells. The latter is desirable, because it particularly honors the quality of the Global Placement result (e.g., w.r.t. timing).

When only cells of single-row height are present, the Standard-cell Legalizers “Tetris” [13] and “Abacus” [22] produce good results. They process the cells one by one, ordered by the $x$-coordinates of their Global Placement positions, and place each cell at the closest free position [13] or at the end of a nearby row, choosing the one that allows for the minimum possible total cell movement [22]. Another strategy, which is employed within the BonnTools project [3, 17], uses a min-cost-flow approach to first assign the cells to zones, unblocked parts of a row [1]. Fixing the left-to-right ordering of the cells contained within each zone to the one imposed by the Global Placement locations, legal cell positions are then obtained by minimizing the total squared cell displacement (or (weighted) bounding box netlength) within each zone. The latter task is captured by the Single-row Problem, which also occurs as a sub-problem of the Abacus Legalizer. It was first studied by Kahng, Tucker, and Zelikovsky [16], who suggested the Clumping Algorithm to tackle it. While their implementation runs in $\theta(m \cdot \log^2(m))$ for unit net weights [2] (where $m$ denotes the number of nets), the fastest implementation, which is due to Suhl [23], achieves a running time of $O(m \cdot \log(m))$ even for general net weights. A similar result has been obtained in the context of scheduling [10]. When the goal is to optimize squared cell displacement, the Clumping Algorithm can easily be implemented to run in time linear in the number of cells [3].

While the mentioned approaches work well in the presence of uniform cell heights, it is not obvious how to generalize them to a setting where cells of double- or even arbitrary multiple-row height may occur. Wang et al. [25] try to adapt the Clumping Algorithm to the double-row case, but manage to guarantee optimality only in a very restricted setting. In contrast to this, Wu and Chu [26] suggest to handle cells of double-row height by, depending on the placement density,
either inflating or matching cells of single-row height to ensure uniform cell heights again. However, as was already pointed out in Reference [21], this strategy can neither handle distinct power alignment constraints nor cells covering more than two rows. Besides, both merging and inflating cells may drastically reduce the placement flexibility as well as lead to a significant area overhead.

Many other authors, therefore, settle for a dynamic programming solution instead of generalizing the Clumping Algorithm, guaranteeing a reasonable runtime by artificially bounding the maximum displacement allowed for each cell by a small number of placement sites. In exchange, they show how to make their dynamic program aware of several other desirable objective traits or incorporate a larger degree of freedom by allowing for a local reordering of cells, even between multiple rows [6, 11, 12, 21].

Other approaches comprise solving a linear complementarity problem to approximately minimize the squared cell movement and then resolving the remaining overlaps [5, 20, 27], applying integer linear programming to legalize sufficiently small regions of the chip separately [15], or making use of a cell insertion scheme [7], combined with bipartite matching and min-cost-flow-algorithms [18].

In this article, we present a fast $O(n \log n)$-time (where $n$ denotes the number of cells) algorithm minimizing the total squared displacement for cells of single- and double-row height that need to be accommodated in two adjacent rows obeying a fixed ordering of the cells covering each row. In contrast to previous dynamic programming approaches, we do not need to artificially restrict the number of available positions for each cell, which may be beneficial for regions of low density and when dealing with coarser grid sizes for double-row cells, which our algorithm can take into account. Moreover, our approach can be extended to support rectangular movebounds for the cells.

In addition to the conference version of this article [14], we show how to incorporate gridding and movebound constraints into our algorithm (Section 5). Furthermore, we discuss possible extensions to more general scenarios featuring cells of multiple-row height and the limitations of our approach (Section 6). We have also added a more detailed description and analysis of our algorithm. Finally, compared to Reference [14], we provide additional experimental results.

The rest of this article is organized as follows: In Section 2, we discuss the Single-row Problem, the Clumping Algorithm and its implementation for piecewise quadratic cost functions. In Section 3, we then introduce the Double-row Problem and show how to reduce it to the Single-row Problem in Section 4. We explain how to incorporate gridding constraints and movebound constraints into our algorithm in Section 5. In a special setting, our algorithm can be extended to also work not only for double-row height cells but also for multiple-row height cells. We explain this in Section 6. Finally, Section 7 presents our experimental results.

2 PRELIMINARIES

The following section comprises the base results our reduction from the Double-row to the Single-row Problem builds upon.

- Section 2.1 reviews the Clumping Algorithm and its analysis.
- Theorem 2.3 points out how an optimum solution to the Single-row Problem changes when the domain is restricted.
- Section 2.2 discusses an efficient implementation of the Clumping Algorithm for piecewise quadratic cost functions.

2.1 The Single-row Problem and the Clumping Algorithm

The task of the Single-row Problem is to place a set of cells of single-row height within a given row, minimizing a sum of continuous, convex objective functions on the positions of the individual
cells. In practice, these will usually model the squared horizontal cell displacement. The left-to-right ordering of the cells is fixed and the cells are not allowed to overlap. More formally, we have the following definition, which is visualized in Figure 2:

**Definition 2.1 (Single-row Problem).**

**Instance:** A tuple \((C, w, x_{\text{min}}, x_{\text{max}}, (f_i)_{i=1}^n)\) consisting of

- a set \(C := \{C_1, \ldots, C_n\}\) of cells,
- cell widths \(w : C \to \mathbb{R}^+\),
- a minimum and maximum coordinate \(x_{\text{min}}, x_{\text{max}} \in \mathbb{R}\) satisfying \(\sum_{i=1}^n w(C_i) \leq x_{\text{max}} - x_{\text{min}}\) and
- convex, continuous functions \(f_i : \mathbb{R} \to \mathbb{R}\) for \(i = 1, \ldots, n\).

**Task:** Find coordinates \((x_i)_{i=1}^n\) minimizing \(\sum_{i=1}^n f_i(x_i)\) subject to

- \(x_{\text{min}} \leq x_1\),
- \(x_i + w(C_i) \leq x_{i+1}\) for \(i = 1, \ldots, n - 1\) and
- \(x_n + w(C_n) \leq x_{\text{max}}\).

For \(i = 1, \ldots, n\), we write

\[
[f_i^-, f_i^+] := \arg\min \left\{ f_i(x), x \in \left[ x_{\text{min}} + \sum_{j=1}^{i-1} w(C_j), x_{\text{max}} - \sum_{j=i}^{n} w(C_j) \right] \right\}.
\]

The Single-row Problem can be solved by the aforementioned Clumping Algorithm [16]. The basic idea can be described as follows: We iteratively, for \(i = 1\) to \(n\), construct a feasible placement for the first \(i\) cells \(C_1, \ldots, C_i\). For \(i = 1\), we just place \(C_1\) at an optimum position for its cost function, restricted to the interval in which the cells need to be placed. For \(i \geq 2\), we again first place \(C_i\) at an optimum position for its cost function. However, this might result in a placement where \(C_i\) is not placed to the right of the preceding cell, because they either overlap, or \(C_i\) even lies to the left of its predecessor. If one of these situations occurs, then we force \(C_i\) and its predecessor to abut by merging them into one cell the width of which equals the sum of widths of the two cells and the cost function of which comprises the costs for placing both \(C_i\) and its predecessor. Now, we try to place the combined cell at an optimum position. If this does not lead to an overlap and does not violate the required left-to-right ordering of the cells, then we continue with the next cell, if one exists. Otherwise, we again merge this new cell with its predecessor, and so on. Algorithm 1 provides a formal description of the algorithm. This description is based on Reference [2].

**Theorem 2.2 ([16]).** The Clumping Algorithm finds an optimum placement.

We prove a slightly stronger statement, which we will need at a later point. To formulate it, we have to introduce the notion of a block, which we define as follows: For a cell \(C_i \in \mathcal{L}\), the block \(B(i)\) represented by \(C_i\) is defined to be the consecutive set of cells:

\[
B(i) := \{ C_j : i \leq j \leq n \land \exists C_l \in \mathcal{L} : i < l \leq j \}.
\]
ALGORITHM 1: Clumping Algorithm

**Input:** An instance of the Single-row Problem given by an ordered list \( \mathcal{L} = (C_1, \ldots, C_n) \) of cells, cell widths \( w: [C_1, \ldots, C_n] \rightarrow \mathbb{R}^+ \), a row interval \([x_{\text{min}}, x_{\text{max}}]\) and convex cost functions \((f_i)_{i=1}^n\).

**Output:** Optimum positions \((x_i)_{i=1}^n\).

1. Add an auxiliary element \( C_0 \) to the front of \( \mathcal{L} \) and set \( x_0 \leftarrow x_{\text{min}} \) and \( w_0 \leftarrow 0 \).
2. for \( i \leftarrow 1 \) to \( n \) do
3. \[ \text{Compute } f_i^- \text{ and } f_i^+. \]
4. \[ w_i \leftarrow w(C_i) \]
5. for \( i \leftarrow 1 \) to \( n \) do
6. \[ \text{PLACE}(C_i, \mathcal{L}) \]
7. for \( i \leftarrow 1 \) to \( n \) with \( C_i \notin \mathcal{L} \) do
8. \[ x_i \leftarrow x_{i-1} + w(C_{i-1}) \]
9. return \((x_i)_{i=1}^n\)

---

ALGORITHM 2: PLACE(C_i, \mathcal{L})

1. \( C_h \leftarrow \) predecessor of \( C_i \) in \( \mathcal{L} \)
2. if \( x_h + w_h \leq f_i^+ \) then
3. \[ x_i \leftarrow \max\{x_h + w_h, f_i^-\} \]
4. else
5. \[ \text{COLLAPSE}(C_h, C_i, \mathcal{L}) \]
6. \[ \text{PLACE}(C_h, \mathcal{L}) \]

ALGORITHM 3: COLLAPSE(C_h, C_i, \mathcal{L})

1. Redefine \( f_h \) as \( x \mapsto f_h(x) + f_i(x + w_h) \) and update \( f_h^- \) and \( f_h^+ \) (w.r.t.
2. \[ [x_{\text{min}} + \sum_{j=1}^{h-1} w(C_j), x_{\text{max}} - \sum_{j=h}^{n} w(C_j)] \])
3. \[ w_h \leftarrow w_h + w_i \]
4. Remove \( C_i \) from \( \mathcal{L} \)

The blocks present at a given point during the run of the Clumping Algorithm indicate sets of cells that the algorithm forces to be placed contiguously (or has clumped together) at that time. Note that the partition into blocks can only get coarser throughout the run of the algorithm.

**Theorem 2.3.** Let \( I' := (C, w, x'_{\text{min}}, x'_{\text{max}}, (f_i)_{i=1}^n) \) be an instance of the Single-row Problem, let \( x_{\text{min}} \leq x'_{\text{min}} \leq x'_{\text{max}} \leq x_{\text{max}} \) and let \( I \) denote the instance of the Single-row Problem that arises from replacing \( x'_{\text{min}} \) and \( x'_{\text{max}} \) by \( x_{\text{min}} \) and \( x_{\text{max}} \), respectively. Then there exists an optimum solution \((x'_i)_{i=1}^n\) for \( I' \), such that for any block \( B(i) \) formed at the end of the execution of Algorithm 1 on \( I \), the cells in \( B(i) \) are placed contiguously.

**Proof.** Let \( K \) be the number of calls to \text{COLLAPSE} that are performed during the execution of Algorithm 1 on \( I \). For \( 0 \leq k \leq K \), let \( \mathcal{L}^k \) denote the intermediate state of the list \( \mathcal{L} \) after the first \( k \) calls to \text{COLLAPSE} have been performed, and let \((B^k(i))_{C_i \in \mathcal{L}^k}\) denote the corresponding intermediate set of blocks. We show the following claim by induction on \( k \):

**Claim:** There exists an optimum solution \((x^*_i)_{i=1}^n\) for \( I' \) with the following property: For any of the blocks \( B^k(i) \) present after the execution of Algorithm 1 on \( I \) has performed the first \( k \) calls to \text{COLLAPSE}, the cells in \( B^k(i) \) are placed contiguously.
Observe that applying the claim for $k = K$ yields Theorem 2.3. We now proceed with the proof of the claim. For $k = 0$, the claim is clearly true, because every cell constitutes a block on its own. Next, assume the claim is true for $k$ with $0 \leq k \leq K - 1$. We want to show that it is also true for $k + 1$. Consider the $k + 1$-st call to COLLAPSE. This call unites two blocks $B(h)$ and $B(i)$ by deleting $C_l$ from $L$. As the claim is true for $k$, we can pick an optimum solution $(x_i^*)_{i=1}^n$ for $I'$ respecting all intermediate blocks that are present after the first $k$ calls to COLLAPSE. We need to show how to modify the solution $(x_i^*)_{i=1}^n$ in such a way that additionally, the cells $C_{i-1}$ and $C_i$ abut, without increasing the costs of the solution. If we already have $x_i^{*'} + w(C_{i-1}) = x_i^*$, then we are done, so assume $x_i^* + \sum_{j=1}^{i-1} w(C_j) = x_i^{*'} + w(C_{i-1}) < x_i^*$. By construction of the algorithm, we have $f_h^- \leq x_h \leq f_h^+$, $w_h = \sum_{j=1}^{i-1} w(C_j)$ and $x_h + w_h > f_i^+$. If $x_i^* > f_i^+$, then we can shift $B(i)$ to the left until it hits max($x_{i-1}^* + w(C_{i-1}), f_i^+$). In doing so, we decrease the total cost, since the cost function $f_i$ of $B(i)$ is strictly monotonically increasing on $[f_i^+, x_{max} - \sum_{j=1}^{n-1} w(C_j)] \supseteq [f_i^+, x_{max} - \sum_{j=1}^{n-1} w(C_j)]$, a contradiction to the assumed optimality of $(x_i^*)_{i=1}^n$. This yields $x_i^* \leq f_i^+$, and, thus, $x_i^* - w_h \leq f_i^* - w_h < x_h \leq f_h^*$.

As a consequence, we can shift $B(h)$ to the right until it hits the left boundary of $B(i)$ without increasing the total cost, because the cost function $f_h$ of $B(h)$ is monotonically decreasing on the interval $[x_{min} + \sum_{j=1}^{h-1} w(C_j), f_h^+] \supseteq [x_{min} + \sum_{j=1}^{h-1} w(C_j), f_h^+]$.

**Remark.** Together with the fact that the Clumping Algorithm places each block $B(i)$ with its optimum range $[f_i^-, f_i^+]$ and hence also within $[x_{min}, x_{max} - w_i]$ (whereby $f_i$ and $w_i$ refer to the respective values after $B(i)$ has been formed), Theorem 2.3 implies that the Clumping Algorithm produces an optimum solution to the Single-row Problem, which yields Theorem 2.2.

**Theorem 2.4.** Let $I$ and $I'$ be as in Theorem 2.3 and let $(x_i^*)_{i=1}^n$ be the solution computed by a run of the Clumping Algorithm on $I$. Then an optimum solution $(x_i^{**})_{i=1}^n$ for $I'$ is given by

$$x_i^{**} = \min \left\{ x_{max}' - \sum_{j=1}^{n} w(C_j), \max \left\{ x_{min}' + \sum_{j=1}^{i-1} w(C_j), x_i^* \right\} \right\}$$

for $i = 1, \ldots, n$.

**Proof.** Feasibility follows easily from the fact that we have $x_{max}' - x_{min}' \geq \sum_{j=1}^{n} w(C_j)$ by definition of the Single-row Problem. By Theorem 2.3, it suffices to show that $(x_i^{**})_{i=1}^n$ places each block $B(i)$ arising from the run of the Clumping Algorithm on $I$ optimally. Pick such a block $B(i)$ and call its cumulated cost function to which $f_i$ is set during the course of the algorithm $f_i$. Then by definition of the Clumping Algorithm, we have $x_i^* \in [f_i^-, f_i^+]$. We distinguish the three cases

- $x_i^* < x_{min}' + \sum_{j=1}^{i-1} w(C_j)$,
- $x_i^* \in \left[ x_{min}' + \sum_{j=1}^{i-1} w(C_j), x_{max}' - \sum_{j=1}^{n} w(C_j) \right]$, and
- $x_i^{**} - \sum_{j=1}^{i-1} w(C_j) < x_i^*$.

In the first case, $x_i^{**} = x_{min}' + \sum_{j=1}^{i-1} w(C_j)$ is set to the leftmost feasible position and furthermore, $f_i$ is monotonically increasing to the right of $x_i^* < x_i^{**}$, showing that $B(i)$ is placed optimally. In the second case, $x_i^{**} = x_i^*$ is placed within the optimum range of $f_i \mid \left[ x_{min}' + \sum_{j=1}^{i-1} w(C_j), x_{max}' - \sum_{j=1}^{n} w(C_j) \right]$ and therefore in particular occupies an optimum position for this function. Finally, in the third case, we get $x_i^{**} = x_{max}' - \sum_{j=1}^{n} w(C_j)$, which is the rightmost feasible position $C_i$ may attain. Given that $f_i$ is monotonically decreasing on $[x_{min}' + \sum_{j=1}^{i-1} w(C_j), x_i^*] \subseteq [x_{min}' + \sum_{j=1}^{i-1} w(C_j), f_i^-]$, optimality follows again.

ACM Transactions on Design Automation of Electronic Systems, Vol. 28, No. 5, Article 71. Pub. date: September 2023.
Note that if all of the \( f_i \) are quadratic functions stored as triples \((a, b, c)\) of coefficients, such that \( f_i : x \mapsto a \cdot x^2 + b \cdot x + c \), the Clumping Algorithm can be implemented to run in linear time, as pointed out, for example, in Reference [3], since the computation of minima as well as shifting a quadratic function in the \( x \)-direction or adding it to another one only requires a constant number of arithmetic operations on the respective coefficients.

2.2 Implementation of the Clumping Algorithm with piecewise quadratic objective functions

Our strategy to solve the problem of minimizing squared movement within two adjacent rows containing cells of both single- and double-row height with a prescribed left-to-right ordering is based on a reduction of an instance of the latter problem to an instance of the Single-row Problem with piecewise quadratic objective functions. In the following subsection, we therefore discuss how to implement the Clumping Algorithm in this case.

**Definition 2.5 (Piecewise Quadratic Function).** For \([a, b] \subseteq \mathbb{R}\), we call a continuous function \( \varphi : [a, b] \to \mathbb{R} \) piecewise quadratic if there exist a non-negative integer \( \kappa \) and

- real numbers \( a =: x_0 < x_1 < \ldots < x_\kappa < x_{\kappa + 1} := b \), and
- quadratic functions \( (\varphi_i : \mathbb{R} \to \mathbb{R})_{i=0}^\kappa \)

such that \( \varphi i [x_i, x_{i+1}] = \varphi i \upharpoonright [x_i, x_{i+1}] \) for all \( i = 0, \ldots, \kappa \) (the symbol \( \upharpoonright \) denotes the restriction of a function to a sub-domain). The positions \( (x_i)_{i=1}^\kappa \) are called \( \text{kinks of \( \varphi \).} \)

We point out that a general piecewise quadratic function does not need to be convex. However, we require the cost functions that appear in an instance of the Double-row Problem (cf. Definition 3.1) to be convex piecewise quadratic functions.

Our goal is to achieve a running time of \( O((n + m) \log(\min\{n, m\})) \) for the Clumping Algorithm, where \( n \) denotes the number of cells and \( m \) specifies the total number of kinks occurring among all cost functions. Therefore, we suggest an implementation of the algorithm that is based on the one proposed in Reference [23] for the case of piecewise linear objective functions. The same approach has also been used in the context of scheduling [10]. In the following, we give a short overview of the data structures used as well as a brief outline of the analysis.

**Representation of cost functions of cells.** We associate the quadratic function \( x \mapsto a \cdot x^2 + b \cdot x + c \) with the triple \((a, b, c)\) and store the restriction \( f_i \upharpoonright [x_{\min}, x_{\max}] \) of the piecewise quadratic cost function \( f_i \) (cf. Definition 2.1) as follows:

Let \( x_{\min} =: p_i^{m_{i+1}} < p_i^{m_i} < \ldots < p_i^1 < p_i^0 := x_{\max} \), such that \( \{p_i^0, \ldots, p_i^{m_i}\} \) is the set of kinks of \( f_i \upharpoonright [x_{\min}, x_{\max}] \) and let \( f_i \upharpoonright [p_i^{j+1}, p_i^j] \) be given by the quadratic function \( f_i^j, j = 0, \ldots, m_i \). Then, we represent \( f_i \) by the ordered list \( F_i := ((p_i^{j+1}, f_i^j))_{j=0}^{m_i} \) consisting of pairs of quadratic functions defining \( f_i \upharpoonright [x_{\min}, x_{\max}] \) on a certain interval and the left boundary of their domain. Throughout the algorithm, for each cell \( C_i \) that has already been processed and is currently placed at the position \( x_i \), we maintain the index \( j(i) \in \{0, \ldots, m_i\} \) for which \( f_i^{(j(i)+1)} < x_i \leq f_i^{(j(i))} \), respectively. \( i = m_i \) if \( x_i = x_{\min} \). Observe that if we implicitly assume all cells to be located at \( x_{\max} \) initially and further consider a cell \( C_j \in B(i) \) as being placed at \( x_i + \sum_{j=1}^{i-1} w(C_j) \), cells never move to the right during a run of the Clumping Algorithm. To see this, note that by definition of \( f_i^- \) and \( f_i^+ \), each cell is located within \([x_{\min}, x_{\max}]\) by construction. Moreover, whenever \( x_h \) is realigned after a call to \textit{COLLAPSE}(\( C_h, C_i, \mathcal{L} \)), then \( h \neq 0 \) and for \( C_h \) the predecessor of \( C_h \) in \( \mathcal{L} \), we get \( \max\{x_h + w_l, f_i^{-}\} = x_h \leq f_i^- \) and \( x_h + w_h > f_i^+ \), before \textit{COLLAPSE} is performed. Hence, after the update of \( f_h \), we have \( f_h^- \leq x_h \), implying that \( x_h \) is decreased, remains unchanged or
another call to COLLAPSE is launched. In the first case, all already processed cells \( C_i \) with \( i > h \) belong to \( B(h) \) and therefore move to the left as well.

As a consequence, the total time needed to maintain the indices \( j(i) \) can be bounded by 
\[ O(\sum_{l=1}^{n} m_l) = O(m), \]
since none of these indices is ever decreased.

**Representation of cost functions of blocks.** To realize calls to PLACE and COLLAPSE efficiently, we need some additional data, which we store for the blocks consisting of cells we have already processed. In doing so, the key observation is the fact that to implement the function PLACE, only local information on the given convex cost function is required, since for a convex real function, the question whether the interval where it attains its minimum lies to the left or right of or contains a certain coordinate can be answered by considering local monotonicity properties. In this spirit, for each block \( B(i) \), we store the following data:

- a heap \( H(i) \) that contains for each \( C_i \in B(i) \) the position \( p_i(j(i) + 1) - \sum_{h=i}^{i-1} w(C_h) \) unless \( j(i) = m_i \), and
- the quadratic function \( g_i \) defining \( f_i \) on the non-empty interval \( (\max H(i), x_i] \), where we set \( \max \emptyset := -\infty \).

We outline how to use them to implement PLACE and COLLAPSE. Consider a call to PLACE\((C_i, L)\) and remember that we implicitly assume that \( x_i = x_{\max} \) for \( 1 \leq i \leq n \) initially. Further observe that this convention ensures that throughout the algorithm, for cells \( C_h, C_i \in L \) with \( h < i \), we have \( x_h + w_h \leq x_i \). To execute PLACE, the first thing we have to decide is whether \( x_h + w_h \leq f_i^+ \). While we can compute the value of the left-hand side in constant time, \( f_i^+ \) is not necessarily known to us. However, we know that by convexity of \( f_i \), \( f_i^+ \) is the unique position in \( [x_{\min} + \sum_{j=1}^{i-1} w(C_j), x_{\max} - \sum_{j=1}^{i-1} w(C_j)] \), such that \( f_i \) is \( [x_{\min} + \sum_{j=1}^{i-1} w(C_j), x_{\max} - \sum_{j=1}^{i-1} w(C_j)] \) is monotonically decreasing to its left and strictly monotonically increasing to its right. As a consequence, if \( f_i \) is \( (\max H(i), x_i] \) (which is given by the quadratic function \( g_i \)) is monotonically decreasing, we can be sure that \( f_i^+ \geq x_i \geq x_h + w_h \). However, as long as \( f_i \) is \( (\max H(i), x_i] \) is strictly monotonically increasing, we can decrease \( x_i \) to \( \max(x_h + w_h, \max H(i)) \), and whenever this maximum is attained by \( \max H(i) \), pop all corresponding entries from the heap, increment the corresponding indices \( j(i) \) by one and insert a new heap entry unless they reach \( m_i \) and update \( g_i \). Note that if one precomputes all of the values \( \sum_{j=1}^{i-1} w(C_j) \), \( i = 1, \ldots, n \) recursively in linear time, which allows to determine \( \sum_{j=1}^{i-1} w(C_j) \) in constant time throughout the algorithm, each of these update steps takes constant time per heap entry. In each case where the maximum is not attained by \( \max H(i) \), we can infer that \( f_i^+ < x_h + w_h \) and therefore launch a call of COLLAPSE. Finally, if there is some \( z \in (\max H(i), x_i) \) where \( g_i \) changes from being monotonically decreasing to being strictly monotonically increasing, then \( z = f_i^+ \), and we are able to decide whether or not \( x_h + w_h \leq f_i^+ \) holds. In case the latter is true, we also have to determine \( \max(x_h + w_h, f_i^+) \). To this end, observe that by convexity of \( f_i \), \( f_i^\rightarrow \) is the unique coordinate in \( [x_{\min} + \sum_{j=1}^{i-1} w(C_j), x_{\max} - \sum_{j=1}^{i-1} w(C_j)] \), such that \( f_i \) is \( [x_{\min} + \sum_{j=1}^{i-1} w(C_j), x_{\max} - \sum_{j=1}^{i-1} w(C_j)] \) is monotonically decreasing to the left, and monotonically increasing to the right of \( f_i^\rightarrow \). By applying a similar strategy as before, we can therefore either compute \( f_i^\rightarrow \in (\max H(i), x_i] \) or set \( x_i \) to \( \max(x_h + w_h, \max H(i)) \geq f_i^\rightarrow \). As a consequence, we are left with discussing the implementation of COLLAPSE\((C_h, C_i, L)\). Since we do not explicitly recompute \( f_h^\rightarrow \) and \( f_h^+ \) and the updates of \( w_h \) and \( L \) can be easily performed in constant time when implementing \( L \) as a doubly linked list, we only have to take care of the redefinition of \( f_h \). To this end, note that \( g_h \) can be updated by setting \( g_h(x) \leftarrow g_h(x) + g_i(x + w_h) \) by a constant number of arithmetic operations on the respective coefficients. As far as the heap \( H(h) \) is concerned, we have to shift all entries in \( H(i) \) by \( w_h \) to the left and then merge \( H(i) \) into \( H(h) \). By employing Leftist Heaps and storing key differences instead of the actual keys (see Reference [24].
for further details), the shifting can be performed in constant and the merging in logarithmic (w.r.t. the total number of heap elements) time. A logarithmic or even constant time bound also applies for all other heap operations we perform, which comprise the creation of empty heaps, the extraction and deletion of maximum heap entries as well as the insertion of new elements. By observing that the maximum heap size is bounded by $\min(n, m)$, since each heap contains at most one entry per cell, but also at most one entry per kink, and that the total number of heap operations is $O(n + m)$, since for every (pair of) shifting and merging, we remove an entry from $L$, and every kink position is added to and removed from a heap at most once, we obtain the claimed runtime bound.

3 THE DOUBLE-ROW PROBLEM

In this section, we

- formally introduce the Double-row Problem, and
- reformulate the feasibility constraints as those of an instance of the Single-row Problem defined on the set of cells of double-row height.

As the name of the problem indicates, the task is to place a set of cells of single- and double-row height within a given rectangular window covering two rows, minimizing a sum of continuous, convex objective functions on the positions of the individual cells. The left-to-right ordering of those cells occupying a certain row is fixed and the cells are not allowed to overlap. Figure 3 visualizes an instance of the Double-row Problem. Table 1 provides an overview of the symbols used in Definition 3.1 and their intuitive meaning.

**Definition 3.1 (Double-row Problem).**

**Instance:**

- a non-empty set $C := \{C_1, \ldots, C_k\}$ of double-row cells,
- a set of cells $B := \{b_{ij}, i = 0, \ldots, k, j = 1, \ldots, m_i\}$ to be placed in the bottom row and a set of cells $T := \{t_{ij}, i = 0, \ldots, k, j = 1, \ldots, n_i\}$ to be placed in the top row, where $m_i, n_i \in \mathbb{N}_0$ for $i = 0, \ldots, k$,
- cell widths $w : C \cup B \cup T \to \mathbb{R}^+$,
- a minimum and maximum coordinate $x_{\text{min}}, x_{\text{max}} \in \mathbb{R}$, such that

$$x_{\text{min}} + \sum_{i=1}^{k} w(C_i) + \sum_{i=0}^{k} \max \left\{ \sum_{j=1}^{m_i} w(b_{ij}), \sum_{j=1}^{n_i} w(t_{ij}) \right\} \leq x_{\text{max}},$$

and

- convex, piecewise quadratic cost functions
  - $f_i : \mathbb{R} \to \mathbb{R}$ for $i = 1, \ldots, k$,
  - $g_{ij} : \mathbb{R} \to \mathbb{R}$ for $i = 0, \ldots, k, j = 1, \ldots, m_i$, and
  - $h_{ij} : \mathbb{R} \to \mathbb{R}$ for $i = 0, \ldots, k, j = 1, \ldots, n_i$. 
### Table 1. Overview of Symbols and Their Meanings

| Symbol        | Meaning                                                                 |
|---------------|-------------------------------------------------------------------------|
| \( C = \{C_1, \ldots, C_k \} \) | cells of double-row height (from left to right) and their cost functions |
| \( (f_i)_{i=1}^k \) | cells of single-row height in the bottom row and their cost functions |
| \( \mathcal{B} := \{b_{ij}, i = 0, \ldots, k, j = 1, \ldots, m_i \} \) | cells in the bottom row located to the left of \( C_1 \) (\( i = 0 \)), between \( C_1 \) and \( C_{i+1} \) (\( i = 0, \ldots, k - 1 \)) or to the right of \( C_k \) (\( i = k \)), indexed from left to right |
| \( (g_{ij})_{i=0}^k m_i \) | cells in the top row located to the left of \( C_1 \) (\( i = 0 \)), between \( C_1 \) and \( C_{i+1} \) (\( i = 0, \ldots, k - 1 \)) or to the right of \( C_k \) (\( i = k \)), indexed from left to right |
| \( \mathcal{B}_i = \{b_{ij}, j = 1, \ldots, m_i \} \) | cells of single-row height in the bottom row and their cost functions |
| \( \mathcal{T}_i := \{t_{ij}, i = 0, \ldots, k, j = 1, \ldots, n_i \} \) | cell widths |
| \( \{h_{ij}\}_{i=0}^k n_i \) | horizontal interval the cells need to be placed within |
| \( \mathcal{T}_t \) | \( \{t_{ij}, j = 1, \ldots, n_i \} \) |
| \( \mathcal{W} \) | \( w : C \cup \mathcal{B} \cup \mathcal{T} \rightarrow \mathbb{R}^+ \) |
| \( [x_{\min}, x_{\max}] \) | \( x_{\min}, x_{\max} \) and each constraint only applies if all of its variables exist. |

**Task:** Find coordinates \( (x_i)_{i=1}^k \), \( (y_{ij})_{i=0}^k m_i \) and \( (z_{ij})_{i=0}^k n_i \) minimizing

\[
\sum_{i=1}^k f_i(x_i) + \sum_{i=0}^k \left( \sum_{j=1}^{m_i} g_{ij}(y_{ij}) + \sum_{j=1}^{n_i} h_{ij}(z_{ij}) \right),
\]

subject to

- \( x_i + w(C_i) \leq x_{i+1} \) for \( i = 0, \ldots, k \),
- \( x_i + w(C_i) \leq y_{i1} \) for \( i = 0, \ldots, k \),
- \( x_i + w(C_i) \leq z_{i1} \) for \( i = 0, \ldots, k \),
- \( y_{ij} + w(b_{ij}) \leq y_{i+1,j} \) for \( i = 0, \ldots, k, j = 1, \ldots, m_i - 1 \),
- \( z_{ij} + w(t_{ij}) \leq z_{i+1,j} \) for \( i = 0, \ldots, k, j = 1, \ldots, n_i - 1 \),
- \( y_{im_i} + w(b_{im_i}) \leq x_{i+1} \) for \( i = 0, \ldots, k \), and
- \( z_{in_i} + w(t_{im_i}) \leq x_{i+1} \) for \( i = 0, \ldots, k \),

where \( x_0 := x_{\min}, w(C_0) := 0, x_{k+1} := x_{\max} \) and each constraint only applies if all of its variables exist.

For \( i = 0, \ldots, k \), we define \( \mathcal{B}_i := \{b_{ij}, j = 1, \ldots, m_i \} \) and \( \mathcal{T}_i := \{t_{ij}, j = 1, \ldots, n_i \} \).

**Proposition 3.2.** Given a tuple \( (x_i^*)_{i=1}^k \) and an instance of the Double-row Problem as defined above, there exists a feasible solution to the Double-row Problem with \( x_i = x_i^* \) for \( i = 1, \ldots, k \) if and only if

\[
x_i^* + w(C_i) + \max \left\{ \sum_{j=1}^{m_i} w(b_{ij}), \sum_{j=1}^{n_i} w(t_{ij}) \right\} \leq x_{i+1}^* \text{ for } i = 0, \ldots, k,
\]

where \( x_0^* := x_0 := x_{\min}, w(C_0) := 0 \) and \( x_{k+1}^* := x_{k+1} := x_{\max} \). We call such a tuple \( (x_i^*)_{i=1}^k \) feasible.

**Proof.** Necessity follows by adding up the respective constraints from the problem definition. However, for a feasible tuple \( (x_i^*)_{i=1}^k \), we obtain a solution to the Double-row Problem by setting \( y_{ij} := x_i^* + w(C_i) + \sum_{l=1}^{j-1} w(b_{il}) \) and \( z_{ij} := x_i^* + w(C_i) + \sum_{l=1}^{j-1} w(t_{il}) \). \( \square \)
ALGORITHM 4: Algorithm for the Double-row Problem

**Input:** an instance \((C, B, T, w, x_{\text{min}}, x_{\text{max}}, (f_i)_{i=1}^k, (g_i)_{i=0}^k, (h_i)_{i=0}^k)\) of the Double-row Problem

**Output:** an optimum solution \(( (x_1^*)_{i=1}^k, (y_1^*)_{i=0}^k, (z_1^*)_{i=0}^k)\)

1. for \(i \leftarrow 0 \) to \(k\)
   2. \((y_{ij})_{j=1}^{m_i} \leftarrow \text{Clumping Algorithm} ((B_i, w | g_i, [x_{\text{min}}, x_{\text{max}}], (g_i)_{j=1}^{m_i})\)
   3. \((z_{ij})_{j=1}^{n_i} \leftarrow \text{Clumping Algorithm} ((T_i, w | T_i, [x_{\text{min}}, x_{\text{max}}], (h_i)_{j=1}^{n_i})\)

4. Define \(x_{\text{min}}' \leftarrow x_{\text{min}} + \max \left\{ \sum_{j=1}^{m_i} w(b_{ij}), \sum_{j=1}^{n_i} w(t_{ij}) \right\}\)

5. Define \(w': C \rightarrow \mathbb{R}^+\) by \(w'(C_i) \leftarrow w(C_i) + \max \left\{ \sum_{j=1}^{m_i} w(b_{ij}), \sum_{j=1}^{n_i} w(t_{ij}) \right\}\)

6. Define cost functions \((F_i)_{i=1}^k\) according to Lemma 4.1

7. \((x_i^*)_{i=1}^k \leftarrow \text{Clumping Algorithm} (C, w', [x_{\text{min}}', x_{\text{max}}'], (F_i)_{i=1}^k)\)

8. Define \((y_{ij})_{i=0}^{m_i} \leftarrow \text{by } y'_{ij} \leftarrow \min \left\{ x_{i+1}^* - \sum_{j=1}^{m_i} w(b_{ij}), \max \left\{ x_{i+1}^* + w(C_i) + \sum_{j=1}^{l-1} w(b_{ij}), y_{ij} \right\} \right\}\)

9. Define \((z_{ij})_{i=0}^{n_i} \leftarrow \text{by } z'_{ij} \leftarrow \min \left\{ x_{i+1}^* - \sum_{j=1}^{n_i} w(t_{ij}), \max \left\{ x_{i+1}^* + w(C_i) + \sum_{j=1}^{l-1} w(t_{ij}), z_{ij} \right\} \right\}\)

10. Return \(( (x_1^*)_{i=1}^k, (y_1^*)_{i=0}^k, (z_1^*)_{i=0}^k)\)

**Remark.** Note that a tuple \((x_i^*)_{i=1}^k\) is feasible if and only if it defines a feasible solution to the instance of the Single-row Problem with cell set \(C\), cell widths

\[
w'(C_i) := w(C_i) + \max \left\{ \sum_{j=1}^{m_i} w(b_{ij}), \sum_{j=1}^{n_i} w(t_{ij}) \right\},
\]

and enclosing x-interval \([x_{\text{min}}', x_{\text{max}}']\) given by

\[
x_{\text{min}}' := x_{\text{min}} + \max \left\{ \sum_{j=1}^{m_i} w(b_{ij}), \sum_{j=1}^{n_i} w(t_{ij}) \right\},
\]

and \(x_{\text{max}}' := x_{\text{max}}\).

4 REDUCTION TO THE SINGLE-ROW PROBLEM

In this section, we show how to reduce the Double-row Problem to the Single-row one, which results in Algorithm 4 for the Double-row Problem. In the following, we outline the steps necessary for the reduction in more detail. As we have already seen how to deal with the subject of feasibility, it remains to transfer costs from the single-row cells to the double-row ones, i.e., to determine the minimum cost of a feasible extension of a feasible tuple \((x_i^*)_{i=1}^k\) and to express it as \(\sum_{i=1}^k f'_i(x_i^*)\) for some piecewise quadratic objective functions \(f'_i\).

- We examine the structure of an optimum extension of a feasible tuple to coordinates for the single-row height cells.
- Lemma 4.1 expresses the total cost of such an extension, up to a constant, as a sum \(\sum_{i=1}^k F_i(x_i^*)\).
- We show that each of the functions \(F_i\) is convex and piecewise quadratic and linearly bound the total number of kinks.
- We then derive our main result stated in Theorem 4.3.

ACM Transactions on Design Automation of Electronic Systems, Vol. 28, No. 5, Article 71. Pub. date: September 2023.
Consider the coordinates \((\bar{y}_{ij})_{i=0}^{k} m_i \) and \((\bar{z}_{ij})_{i=0}^{k} n_i \) arising from runs of the Clumping Algorithm on the instances of the Single-row Problem given by \((\mathcal{B}_i, w \mid \mathcal{B}_i, x_{min}, x_{max}, (g_{ij})_{i=1}^{m_i})\) and \((\mathcal{T}_i, w \mid \mathcal{T}_i, x_{min}, x_{max}, (h_{ij})_{i=1}^{n_i})\) for \(i = 0, \ldots, k\). Note that once a feasible tuple \((x^*_i)_{i=1}^{k}\) of coordinates for the double-row cells has been fixed, coordinates \((y_{ij})_{i=0}^{k} m_i \) and \((z_{ij})_{i=0}^{k} n_i \) extend them to a feasible solution of the Double-row Problem if and only if for each \(i \in \{0, \ldots, k\}\), \((y_{ij})_{i=1}^{m_i}\) and \((z_{ij})_{i=1}^{n_i}\) constitute feasible solutions of the instances of the Single-row Problem given by \((\mathcal{B}_i, w \mid \mathcal{B}_i, x^*_i + w(C_i), (g_{ij})_{i=1}^{m_i})\) and \((\mathcal{T}_i, w \mid \mathcal{T}_i, x^*_i + w(C_i), (h_{ij})_{i=1}^{n_i})\), respectively, where again \(x^*_0 := x_{min}, w(C_0) := 0\) and \(x^*_{k+1} := x_{max}\). Note that these instances are feasible by feasibility of \((x^*_i)_{i=1}^{k}\). But now, since for each \(i = 0, \ldots, k\), we have \(x_{min} \leq x^*_i + w(C_i) \leq x^*_{i+1} \leq x_{max}\), Theorem 2.4 tells us that an optimum extension \((y_{ij})_{i=0}^{k} m_i \) and \((z_{ij})_{i=0}^{k} n_i \) of \((x^*_i)_{i=1}^{k}\) is given by

\[
y_{ij}^* = \min \left\{ x^*_{i+1} - \sum_{l=j}^{m_i} w(b_{il}), \max \left\{ x^*_i + w(C_i) + \sum_{l=1}^{j-1} w(b_{il}), \bar{y}_{ij} \right\} \right\}
\]

and

\[
z_{ij}^* = \min \left\{ x^*_{i+1} - \sum_{l=j}^{n_i} w(t_{il}), \max \left\{ x^*_i + w(C_i) + \sum_{l=1}^{j-1} w(t_{il}), \bar{z}_{ij} \right\} \right\}.
\]

This allows us to express the total cost of the solution in terms of the coordinates \((x^*_i)_{i=1}^{k}\). Figure 4 illustrates the new cost functions we obtain.

**Lemma 4.1.** Let \((\bar{y}_{ij})_{i=0}^{k} m_i \) and \((\bar{z}_{ij})_{i=0}^{k} n_i \) as before and define

\[
F_i : x \mapsto f_i(x)
\]

\[
+ \sum_{j=1}^{m_i} g_{i-1j} \left( \min \left\{ x - \sum_{l=j}^{m_{i-1}} w(b_{i-1l}), \bar{y}_{i-1j} \right\} \right)
\]

\[
+ \sum_{j=1}^{m_i} g_{ij} \left( \max \left\{ x + w(C_i) + \sum_{l=1}^{j-1} w(b_{il}), \bar{y}_{ij} \right\} \right)
\]

\[
+ \sum_{j=1}^{n_{i-1}} h_{i-1j} \left( \min \left\{ x - \sum_{l=j}^{n_{i-1}} w(t_{i-1l}), \bar{z}_{i-1j} \right\} \right)
\]

\[
+ \sum_{j=1}^{n_i} h_{ij} \left( \max \left\{ x + w(C_i) + \sum_{l=1}^{j-1} w(t_{il}), \bar{z}_{ij} \right\} \right)
\]

and \(c := \sum_{i=1}^{k-1} \sum_{j=1}^{m_i} g_{ij}(\bar{y}_{ij}) + \sum_{i=1}^{k-1} \sum_{j=1}^{n_i} h_{ij}(\bar{z}_{ij})\). Then for a feasible tuple \((x^*_i)_{i=1}^{k}\), the total cost of an optimum solution to the Double-row Problem with \(x_i = x^*_i\) for \(i = 1, \ldots, k\) amounts to \(\sum_{i=1}^{k} F_i(x^*_i) - c\).

**Proof.** Recall that an optimum extension \((y_{ij})_{i=0}^{k} m_i \) and \((z_{ij})_{i=0}^{k} n_i \) of \((x^*_i)_{i=1}^{k}\) is given by Equations (1) and (2). This means that proving Lemma 4.1 boils down to verifying that Equation (8) holds, since the left-hand side equals the cost of an optimum solution to the Double-row Problem with \(x_i = x^*_i\) for \(i = 1, \ldots, k\):

\[
\sum_{i=1}^{k} f_i(x^*_i) + \sum_{i=0}^{k} \left( \sum_{j=1}^{m_i} g_{ij}(y_{ij}^*) + \sum_{j=1}^{m_i} h_{ij}(z_{ij}^*) \right) = \sum_{i=1}^{k} F_i(x^*_i) - c.
\]
Fig. 4. How the new cost function $F_i$ (drawn in black, kinks indicated by black dots) (see Lemma 4.1) of the double-row cell $C_i$ arises. First, $C_i$ has to pay for its own objective, which is depicted in green. As far as cells of single-row height in the two adjacent gaps are concerned, we imagine placing them at there optimum positions computed by applying the Clumping Algorithm to the 4 single-row sub-instances arising for the upper and lower row in the left and right gap, respectively. Now, when moving $C_i$ to the right, at some point, the right boundary of $C_i$ starts hitting the left boundary of $b_{i1}$ or $t_{i1}$, respectively, causing them to be displaced as well. The corresponding coordinates are indicated by the right dashed blue line for $b_{i1}$, and by the right dashed red line for $t_{i1}$. To the right of the respective coordinates, we add the costs for displacing $b_{i1}$ and $t_{i1}$, whereas to the left of these coordinates, we only invoke constant costs (dotted) for placing them at their optimum positions. The respective cost functions are drawn in red ($t_{i1}$) and blue ($b_{i1}$). We get a similar cost contribution for the other cells in $B_l$ and $T_l$, which are not depicted here. When shifting $C_i$ far enough to the left, we have to pay for shifting $b_{i-1m_{i-1}}$ and $t_{i-1n_{i-1}}$ to the left as well, which is indicated in the left half of the picture. Again, we get a similar cost contribution for the other cells in $B_{l-1}$ and $T_{l-1}$, which are not depicted here.

Our strategy to prove Equation (8) is to expand the right-hand side $\sum_{i=1}^{k} F_i(x_i^*) - c$ by plugging in the definitions of the functions $(F_i)^k$ and of the constant $c$, and to then prove the following three statements:

(a) For each $i = 1, \ldots, k$, the sum of all terms of the form $f_i(\cdot)$ that occur on the left-hand side is equal to the sum of all terms of the form $f_i(\cdot)$ that occur on the right-hand side.

(b) For each $i = 0, \ldots, k$, and $j = 1, \ldots, m_i$ the sum of all terms of the form $g_{ij}(\cdot)$ that occur on the left-hand side is equal to the sum of all terms of the form $g_{ij}(\cdot)$ that occur on the right-hand side.

(c) For each $i = 0, \ldots, k$, and $j = 1, \ldots, n_i$ the sum of all terms of the form $h_{ij}(\cdot)$ that occur on the left-hand side is equal to the sum of all terms of the form $h_{ij}(\cdot)$ that occur on the right-hand side.

By taking a closer look at the definition of the $F_i$ and of the constant $c$, the first statement (a) actually follows immediately, since the only term involving the cost function $f_i$ for $i = 1, \ldots, k$ on both the left- and the right-hand side is “$f_i(x_i^*)$,” which is added once on either side.

As (b) and (c) can be verified analogously, we restrict ourselves to the proof of (b) in the following. For $i = 0$ and $j = 1, \ldots, m_0$, we need to show that $g_{0j}(\min(x_1^* - \sum_{t=j}^{n_0} w(b_{0t}), y_{0j})) = g_{0j}(y_{0j}^*)$. This follows from

$$x_0^* + w(C_0) + \sum_{l=1}^{j-1} w(b_{0l}) = x_{\text{min}} + \sum_{l=1}^{j-1} w(b_{0l}) \leq y_{0j},$$

ACM Transactions on Design Automation of Electronic Systems, Vol. 28, No. 5, Article 71. Pub. date: September 2023.
implying that
\[ y_{ij}^* = \min \left\{ x_i^* - \sum_{l=j}^{m} w(b_{il}), \max \left\{ x_i^* + w(C_i) + \sum_{l=1}^{j-1} w(b_{il}), \bar{y}_{ij} \right\} \right\} = \min \left\{ x_i^* - \sum_{l=j}^{m} w(b_{il}), \bar{y}_{ij} \right\}. \]

For \( i = k \) and \( j = 1, \ldots, m_k \), we have to show that \( g_{kj}(\max\{x_k^* + w(C_k) + \sum_{l=1}^{j-1} w(b_{kl}), \bar{y}_{kj}\}) = \bar{y}_{kj}(y_{kj}^*) \). The latter follows, since
\[ \bar{y}_{kj} \leq x_{max} - \sum_{l=j}^{m_k} w(b_{kl}) = x_{k+1}^* - \sum_{l=j}^{m_k} w(b_{kl}) \]

and
\[ x_k^* + w(C_k) + \sum_{l=1}^{j-1} w(b_{kl}) \leq x_{max} - \sum_{l=j}^{m_k} w(b_{kl}) = x_{k+1}^* - \sum_{l=j}^{m_k} w(b_{kl}) \]

yield
\[ y_{kj}^* = \min \left\{ x_{k+1}^* - \sum_{l=j}^{m_k} w(b_{kl}), \max \left\{ x_k^* + w(C_k) + \sum_{l=1}^{j-1} w(b_{kl}), \bar{y}_{kj} \right\} \right\} = \max \left\{ x_k^* + w(C_k) + \sum_{l=1}^{j-1} w(b_{kl}), \bar{y}_{kj} \right\}. \]

Finally, for \( i \in \{1, \ldots, k-1\} \) and \( j \in \{1, \ldots, m_i\} \), we have to verify that
\[ g_{ij}(\min\{x_{i+1}^* - \sum_{l=j}^{m_i} w(b_{il}), \bar{y}_{ij}\}) + g_{ij}(\max\{x_i^* + w(C_i) + \sum_{l=1}^{j-1} w(b_{il}), \bar{y}_{ij}\}) - g_{ij}(\bar{y}_{ij}) = g_{ij}(y_{ij}^*). \]

By feasibility, we have
\[ x_i^* + w(C_i) + \sum_{l=1}^{j-1} w(b_{il}) \leq x_{i+1}^* - \sum_{l=j}^{m_i} w(b_{il}). \quad (9) \]

If we additionally have \( \bar{y}_{ij} \leq x_{i+1}^* - \sum_{l=j}^{m_i} w(b_{il}) \), then \( \min\{x_{i+1}^* - \sum_{l=j}^{m_i} w(b_{il}), \bar{y}_{ij}\} = \bar{y}_{ij} \). Moreover, \( \bar{y}_{ij} \leq x_{i+1}^* - \sum_{l=j}^{m_i} w(b_{il}) \) and Inequality (9) yield
\[ y_{ij}^* = \min \left\{ x_{i+1}^* - \sum_{l=j}^{m_i} w(b_{il}), \max \left\{ x_i^* + w(C_i) + \sum_{l=1}^{j-1} w(b_{il}), \bar{y}_{ij} \right\} \right\} = \max \left\{ x_i^* + w(C_i) + \sum_{l=1}^{j-1} w(b_{il}), \bar{y}_{ij} \right\}, \]

so
\[ g_{ij}(\max\{x_i^* + w(C_i) + \sum_{l=1}^{j-1} w(b_{il}), \bar{y}_{ij}\}) + g_{ij}(\min\{x_{i+1}^* - \sum_{l=j}^{m_i} w(b_{il}), \bar{y}_{ij}\}) - g_{ij}(\bar{y}_{ij}) = g_{ij}(y_{ij}^*) + g_{ij}(\bar{y}_{ij}) - g_{ij}(\bar{y}_{ij}) = g_{ij}(y_{ij}^*). \]

The case where \( x_{i+1}^* - \sum_{l=j}^{m_i} w(b_{il}) < \bar{y}_{ij} \) can be handled similarly.
Up to the constant $c$, which only depends on the given instance of the Double-row Problem, but not on the tuple $(x^j_i)^{k}_{i=1}$, we can hence express the cost of an optimum solution extending a feasible tuple $(x^j_i)^{k}_{i=1}$ as a sum of the cost functions $(F_i)^{k}_{i=1}$ applied to the individual coordinates.

Let $\sum$ be the sum of these cost functions.

**Lemma 4.2.** Each of the functions $(F_i)^{k}_{i=1}$ defined in Lemma 4.1 is piecewise quadratic and convex. The total number of kinks of the cost functions $(F_i)^{k}_{i=1}$ can be bounded by $2 \cdot (|T|) + N$, where $N$ denotes the total number of kinks present in the cost functions of the single- and double-row cells.

**Proof.** As the class of piecewise quadratic functions is closed under linear shifting and replacement by a constant function to the left or right of a certain coordinate (ensuring continuity), each of the summands (3) to (7) induces a piecewise quadratic function in $x$. Hence, each of the functions $(F_i)^{k}_{i=1}$ is piecewise quadratic as a sum of piecewise quadratic functions.

Concerning the total number of kinks, we first note that each kink of a sum of piecewise quadratic functions must be a kink of one of the summands. Hence, for $i \in \{1, \ldots, k\}$, each kink of $F_i$ is of one of the following types:

- A kink of $f_i$,
- A kink $x$ of $g_{i-1j}$ with $x < y_{i-1j}$, or $y_{i-1j}$, $j = 1, \ldots, m_{i-1}$,
- A kink $x$ of $g_{ij}$ with $x > y_{ij}$, or $y_{ij}$, $j = 1, \ldots, m_i$,
- A kink $x$ of $h_{i-1j}$ with $x < z_{i-1j}$, or $z_{i-1j}$, $j = 1, \ldots, n_{i-1}$,
- A kink $x$ of $h_{ij}$ with $x > z_{ij}$, or $z_{ij}$, $j = 1, \ldots, n_i$.

In particular, each kink of each of the cost functions of the single- and double-row cells induces a kink of at most one of the functions $(F_i)^{k}_{i=1}$ and additionally, each of the coordinates $(y_{ij})^{m_i}_{i=0j=1}$ and $(z_{ij})^{n_i}_{i=0j=1}$ induces a kink of at most two of the functions $(F_i)^{k}_{i=2}$. This gives a bound of $2 \cdot (|T|) + N$ on the total number of kinks.

Finally, we would like to show that each of the functions $(F_i)^{k}_{i=1}$ is convex. To this end, it suffices to prove that each of the summands (3)–(7) induces a convex function. This is clear for Summand (3), and it remains to show it for Summands (4) and (5), since Summands (6) and (7) can be handled analogously. Moreover, as the proof for Summand (5) is very similar to the proof for Summand (4), we only provide the latter one.

Let $L^b_{i-1}$ denote the list of cells arising from the run of the Clumping Algorithm on the aforementioned instance of the Single-row Problem with cell set $B_{i-1}$. Given that for $b_{i-1j} \in L^b_{i-1}$, the cells in the block $B(i - 1, j)$ starting at $b_{i-1j}$ are placed contiguously, we can rewrite Summand (4) as $\Sigma b_{i-1j} \in L^b_{i-1} G_{i-1j}(\min\{x - \Sigma_{i=1}^{m_i} w(b_{i1}, y_{i-1j})\}$, where $G_{i-1j}$ denotes the cumulated cost function of the block represented by $b_{i-1j}$. Recall that by definition of the Clumping Algorithm, $y_{i-1j}$ occupies a minimum position of $G_{i-1j}$ for $b_{i-1j} \in L^b_{i-1}$. Given that for a continuous, convex function $f : [a, b] \rightarrow \mathbb{R}$ and $x_0 \in \text{argmin}\{f(x), x \in [a, b]\}$, the function mapping $x \in [a, b]$ to $f(\min\{x, x_0\})$ is convex, it follows that Summand (4) defines a convex function in $x$ (see Figure 5).

This completes our reduction from the Double to the Single-row Problem and it remains to discuss the runtime it requires. Note that the positions $(y_{ij})^{m_i}_{j=0i=1}$ and $(z_{ij})^{n_i}_{j=0i=1}$ can be computed in total time $O((|B| + |T| + N) \cdot \log(|B| + |T|))$, where again $N$ denotes the total number of kinks of the all cost functions appearing in the given instance of the Double-row Problem.

A time of $O((|C| + |B| + |T| + N) \cdot \log(|C| + |B| + |T| + N))$ then suffices to build up and solve the instance of the Single-row Problem on the set of double-row cells to which we reduce, and optimum coordinates for the single-row cells can be deduced from the computed positions for the cells in $C$ in linear time. Putting everything together, we can therefore formulate the following theorem:
Fig. 5. For a continuous, convex function \( f : [a, b] \to \mathbb{R} \) and \( x_0 \in \operatorname{argmin}\{f(x), x \in [a, b]\} \), the functions \( x \mapsto f(\max\{x, x_0\}) \) and \( x \mapsto f(\min\{x, x_0\}) \) are convex.

**Theorem 4.3.** The Double-row Problem with convex, piecewise quadratic functions with a total amount of \( N \) kinks can be solved in time \( O((|C| + |B| + |T| + N) \cdot \log(|C| + |B| + |T| + N)) \).

## 5 Gridding and Movebound Constraints

In this section, we discuss some additional features that might be desirable in practice and that can be incorporated into the described reduction from the Double-row Problem to the Single-row Problem as well as into the implementation of the Clumping Algorithm relatively easily. As they would nevertheless make the above computations even more lengthy and tedious, we decided not to include them in the main part to avoid shadowing the key ideas.

- In Section 5.1, we show how to deal with horizontal grids the coordinates of the cells need to snap into.
- Section 5.2 shows how to make our algorithm respect movebounds, i.e., sub-intervals the (left) \( x \)-coordinate of a cell needs to be contained in.

### 5.1 Integrality and Gridding of Coordinates

As already mentioned in the introduction, on practical instances, cells do not only have to align with the cell rows vertically, but they also have to snap into a certain grid horizontally, depending on the way they are flipped in the \( x \)- and \( y \)-direction. The resulting feasible (horizontal) positions at which cells can be placed are usually referred to as placement sites, the distance between two horizontally adjacent placement sites is called the *width* of the placement site. Often, standard cells can be placed at every placement site and moreover, the widths of the cells are integer multiples of the placement site width. In such a setting, the Clumping Algorithm automatically\(^1\) places the cells on-grid, that is, on placement sites. However, on instances that, for example, feature power staple pattern that only repeat every second placement site, it might only be possible to place a cell at every second placement site. Moreover, if the cell width is an odd multiple of the placement site width, depending on the way the cell is flipped, then we can either only place it on placement sites with an odd, or only on placement sites with an even index. These requirements are not met by the Clumping Algorithm out of the box. Hence, we use this section to explain how to incorporate *gridding constraints* of the following form into our implementation of the Clumping Algorithm for *piecewise quadratic objective functions*: In addition to the data \( C = \{C_1, \ldots, C_n\} \), \( w : C \to \mathbb{R}^+ \), \( x_{\text{min}}, x_{\text{max}} \) and \((f_i)_{i=1}^n\) that an instance of the Single-row Problem (cf. Definition 2.1) contains, we provide a *grid delta* \( \delta \in \mathbb{N} \) and integral *anchor coordinates* \((a_i)_{i=1}^n\). We then restrict the set of feasible solutions \((x_i)_{i=1}^n\) to those where \( x_i \) can be written as \( x_i = a_i + j_i \cdot \delta \) with \( j_i \in \mathbb{Z} \).

---

1Assuming that \( x_{\text{min}} \) and \( x_{\text{max}} \) are placement sites.
for all \( i = 1, \ldots, n \). We call the positions of the form \( a_i + j_i \cdot \delta \) with \( j_i \in \mathbb{Z} \) on-grid (for cell \( C_i \)), see Figures \( 6(a) \) and \( 6(b) \).

For example, to require that all cells need to be placed on placement sites, we would set \( \delta \) to be the width of a placement site and \( a_i = 0 \) for all \( i = 1, \ldots, n \). In a setting where cells are only allowed to be placed on every second placement site, we set \( \delta \) to be twice the width of a placement site. For a cell \( C_i \) that can only be placed at positions that are an even multiple of the placement site width, we set \( a_i := 0 \), whereas for a cell \( C_i \) that can only be placed at positions that are an odd multiple of the placement site width, we let \( a_i \) equal the placement site width.

We now explain how to enforce on-grid coordinates during the Clumping Algorithm. For convenience, we can first increase \( x_{\text{min}} \) to the next on-grid position for \( C_i \) and decrease \( x_{\text{max}} \) to the largest coordinate where the right boundary of \( C_n \) can lie if \( C_n \) is placed on-grid and within the interval \([x_{\text{min}}, x_{\text{max}}]\). Then, for each of the cells \( C_i \) with \( 1 \leq i \leq n-1 \), we place the left boundary of \( C_i \) at an arbitrary on-grid position and increase the width of \( C_i \) until the right boundary of \( C_i \) snaps into the grid for \( C_{i+1} \) (see Figure 6). This ensures that whenever \( C_i \) is placed at an on-grid position \( x_i \) and we place \( C_{i+1} \) at \( x_{i+1} := x_i + w(C_i) \),\(^2\) then \( C_{i+1} \) lies on-grid as well. Moreover, modifying the widths in this way does not destroy any feasible solution.

As far as the cost functions are concerned, we would like all kinks and leftmost minima \( f_i^- \) (see Definition 2.1 and line 3 of Algorithm 2) to be located at on-grid positions. Recall that we are assuming piecewise quadratic cost functions in this section, since that is what we need for our reduction from the Double-row Problem. While for kinks, we can ensure that they are on-grid in a pre-processing step, as an on-grid kink for \( C_{i+1} \) transforms into an on-grid kink for \( C_i \) when shifting it to the left by \( w(C_i) \), this does not work out for the minima, since, for example, the sum of

---

\(^2\)This happens in line 8 of Algorithm 1 if cells \( C_i \) and \( C_{i+1} \) were clumped together.
two quadratic functions with even integral minima \((\delta = 2)\) does not necessarily attain its minimum at an even integral position. We will therefore first describe how to deal with kink positions and then face the subject of minima.

To this end, observe that the set of optimum on-grid solutions does not change if each cost function \(f_i\) is replaced by a new convex, piecewise quadratic cost function \(f'_i\) that agrees with \(f_i\) on every on-grid position, simply because the definition of optimality is oblivious of what happens at off-grid coordinates. We obtain \(f'_i\) from \(f_i\) as follows: For every off-grid kink \(p\), denote the next on-grid position to the left by \(p_{\text{left}}\) and the next on-grid position to the right by \(p_{\text{right}}\). We remove \(p\) from the set of kinks, add \(p_{\text{left}}\) and \(p_{\text{right}}\) to the set of kinks and replace \(f_i\) on the interval \([p_{\text{left}}, p_{\text{right}}]\) by the linear interpolation between the two points \((p_{\text{left}}, f_i(p_{\text{left}}))\) and \((p_{\text{right}}, f_i(p_{\text{right}}))\). These modifications preserve convexity and do not change the function value at any on-grid position. All off-grid kinks are removed, and the total number of kinks is at most doubled. An example for the construction described above is given in Figure 7.

We still need to clarify how to treat minima that do not lie on grid. But by the symmetry of quadratic functions, it suffices to modify the computation of \(f_i^-/f_i^+\) for the case \(a > 0\) by simply rounding to the closer on-grid coordinate, respectively, rounding down/up if the minimum position has equal distance to the next on-grid coordinate to the left and to the right. Note that this corresponds to implicitly replacing the segment of the quadratic function between the on-grid coordinates adjacent to the minimum location by the respective chord, maintaining convexity.

Fig. 7. Modification of the cost function to achieve on-grid kink positions.
This way, we can ensure that we always position cells/blocks at on-grid coordinates. Due to the modified widths, placing a block at an on-grid location with respect to the first cell automatically complies with the grids of all cells contained in the block.

Similar ideas as above can be used to also incorporate gridding constraints into the reduction from the Double- to the Single-row Problem. We can even generalize it in a way that allows the cells of double-row height to have a (larger) grid delta that is any integer multiple of the grid delta for the single-row cells. Such a situation sometimes occurs on practical instances.

5.2 Movebounds

Another property of practical instances that we did not discuss so far are rectangular movebounds in which the lower left corners of the respective cells need to be contained. In the application cases of the Single- and Double-row Problem where the assignment to rows is already fixed, the movebounds become intervals from which feasible $x$-coordinates for the cells need to be chosen. For the Single-row Problem, a variant where movebounds need to be respected but no further objective is given is discussed in Reference [23]. To incorporate movebounds into our algorithm for the Double-row Problem, it is helpful to first strengthen them in such a way that for each position within the movebound of a cell, there exists a feasible solution where this position is attained. The latter can be achieved by propagating upper movebounds from right to left and lower movebounds from left to right (see Figure 8). In doing so, no feasible solution is excluded.

For the definition of $f_j^-$ and $f_j^+$ (see Definition 2.1), we first restrict $f_j$ to the strengthened movebound. This ensures that placing a cell $C_j$ at $f_j^-$ or attaching it to a block always leads to a feasible location. Hence, one only has to make sure that one does not move a cell out of its movebound when shifting a block to the left. Due to the strengthened movebounds, we never violate any upper movebound as long as the first cell of a block is placed within its movebound. Moreover, as one can show that whenever two blocks conflict, there exists an optimum solution in which they abut in a similar way as in the analysis of Algorithm 1, we know that we never need to undo any clumping step. Therefore, when merging two blocks, we can safely set the lower movebound of the left block to the maximum of its previous value and the lower movebound of the right block minus the width of the left one. After this update, it suffices to ensure that the first cell of each block stays within its movebound.

If both movebounds and gridding should be incorporated, then one has to pay attention to the order in which the required modifications are applied. That is, one first has to restrict the movebounds to the interval $[x_{\min}, x_{\max}]$ and then further to the convex hull of the contained grid coordinates, and compute the new cell widths, before applying the movebound strengthening step.

6 EXTENSION TO MULTI-ROW CELLS

In this section, we first briefly discuss the $k$-row Problem, a generalization of the Double-row Problem to a setting that features, for a fixed parameter $k \geq 2$, both cells that are $k$ rows high, and cells...
of single-row height. More precisely, we consider a setting where we are given \( k \) rows, a set of cells covering all \( k \) rows and a set of cells of single-row height. The row assignment as well as the order of the cells within each row are fixed (see Figure 9). The task is to minimize the sum of objective functions in the \( x \)-coordinates of the individual cells (e.g., the sum of squared distances to given target positions). By an analogous approach as in Section 4, we can reduce the \( k \)-row Problem to the Double-row Problem and obtain Theorem 6.1. The same of course also holds when the given instance can be split into independent sub-instances of the aforementioned type.

**Theorem 6.1.** The \( k \)-row Problem with piecewise quadratic objective functions can be solved in time \( O((\# \text{ cells } + \# \text{kinks}) \cdot \log(\# \text{ cells} + \# \text{kinks})) \), where \( \# \text{ cells} \) denotes the total number of cells, and \( \# \text{kinks} \) is the total number of kinks occurring in the piecewise quadratic objective functions.

In a more general setting, where cells covering at least three different numbers of rows, or interleaving cells of multiple-row height (that is pairs of cells for which neither of the ranges of rows covered by the respective cell is contained in the other) occur, a natural strategy might be to reduce this problem to the known one by sweeping a window covering two adjacent rows over the instance from bottom to top, lifting it by one row in each step. In doing so, one can try to iteratively transfer the costs of the cells only present in the bottom row to those covering both rows as in the reduction from the Double-row Problem to the Single-row Problem. (Note that if there are no cells covering both rows, our instance splits into two independent sub-instances, a lower one ending with the bottom row, and an upper instance starting at the top one.) In each step, the total number of kinks only increases by twice the number of cells occupying the bottom, but not the top row by an analogous proof as in Lemma 4.1. Hence, the total number of kinks can be bounded by the number of kinks of all original cost functions of cells we have already encountered plus twice the number of cells that lie below the current window at any point, ensuring a running time of \( O(\# \text{rows} \cdot (\# \text{cells} + \# \text{kinks}) \cdot \log(\# \text{cells} + \# \text{kinks})) \).

However, there is a problem with this approach that does not seem trivial to fix. While in the case of the Double-row Problem, the feasibility constraints for a tuple of coordinates for the double-row cells nicely transform into disjointness constraints for adjacent double-row cells, forcing a cell to stay to the left of its successor/to the right of its predecessor, as well as the requirement of the cells being placed within a certain \( x \)-interval, the situation gets more complicated in the presence of interleaving cells. As depicted in Figure 10, these might lead to distance constraints between non-neighbors, that are not implied by the set of constraints imposed for pairs of neighboring cells and therefore exceed what our current formulation of the Single-row Problem can handle. In particular, the existence of these constraints also implies that the sub-instances arising in the gaps between the double-row cells of our current window cannot always be treated independently anymore. Hence, a crucial requirement for our way of transferring costs to the double-row cells is not met.
The task of either figuring out how to overcome the difficulties mentioned above or coming up with an entirely new strategy to tackle Placement Legalization in the presence of (interleaving) cells of multiple-row height therefore remains an open question requiring further research.

7 EXPERIMENTAL RESULTS

We implemented the proposed algorithm in the C++ programming language and embedded it into the legalization framework described in Reference [1]. More precisely, we first run the legalization algorithm from Reference [1], which legalizes all cells of more than single-row height via a greedy projection approach and then proceeds by assigning all cells of single-row height to so-called zones, unblocked segments of cell rows, through a min-cost-flow algorithm. Within each zone, the left-to-right ordering is inferred from the Global Placement positions. While the algorithm from Reference [1] proceeds by optimizing squared cell movement only within each zone making use of the Clumping Algorithm, we instead apply the Double-row Algorithm to the instances of the Double-row Problem arising from the given left-to-right ordering in every second pair of rows, treating all cells of more than double-row height as blockages. As the benchmark instances we performed our experiments on did not contain interleaving cells of double-row height, i.e., cells of double-row height only sharing one row as depicted in Figure 10, we did not have to deal with the challenges interleaving cells would raise.

All experiments were performed single-threaded on an Intel Xeon 3.3 GHz CPU with 384 GB RAM. We use two different sets of benchmarks. The first set of benchmarks consists of all 16 instances from the ICCAD-2017 CAD Contest on Multi-deck Standard-cell Legalization [8]. These instances contain movable cells that are up to four rows high. The percentage of double-row cells in these instances is 4.2% on average. The second set of benchmarks that we use contains 19 instances generated by the authors of [7] by modifying instances from the ISPD 2015 Detailed Routing-driven Placement Contest [4]. The original ISPD 2015 benchmark contains only cells of single-row height. Chow, Pui, and Young [7] doubled the height of some cells (either the sequential ones or a randomly chosen subset of one-tenth of all cells) while at the same time halving their width. All movable cells in these instances are single-row or double-row cells. The percentage of double-row cells in these instances is 6.3% on average. For all benchmark instances, we omit fence region constraints as well as soft constraints, but stick to the required power-rail alignment.

As most prior works optimize linear instead of squared cell movement, we conduct two experiments to compare our algorithm with other works. The first one aims at establishing the competitiveness of our legalization approach when compared to recent works on the matter of mixed-cell-height legalization. The second experiment displays the effectiveness of the Double-row Algorithm in improving squared cell movement.

For the first experiment, we employ our proposed legalization method to minimize linear movement during the Double-row Algorithm. Observe that this is possible, since for each cell, once its


Table 2. Comparison between the Average Linear Cell Movement in Terms of Horizontal Placement Sites for the 16 instances from the ICCAD-2017 CAD Contest on Multi-deck Standard-cell Legalization [8]

| Instance       | Δ HPWL [5] % | Av. Linear Movement (Sites) [5] | Max. Linear Movement (Sites) [5] | CPU (sec) |
|----------------|--------------|---------------------------------|---------------------------------|-----------|
|                | [20] [1] Ours | [20] [1] Ours                   | [20] [1] Ours                   |           |
| des_perf_1     | 16.21%       | 10.86, 6.97, 6.66, 6.66         | 200.82, 48.95, 57.22, 57.22     | 11.23     |
| des_perf_a_md1 | 3.27%        | 6.71, 5.94, 5.85, 5.79          | 607.30, 607.30, 607.30, 607.30   | 2.30      |
| des_perf_a_md2 | 3.35%        | 6.77, 5.93, 6.08, 6.07          | 403.86, 403.86, 403.86, 403.86   | 2.19      |
| des_perf_b_md1 | 1.75%        | 5.17, 4.77, 4.78, 4.72          | 79.34, 38.45, 48.19, 45.19       | 2.01      |
| des_perf_b_md2 | 2.05%        | 5.74, 5.25, 5.38, 5.31          | 198.74, 39.76, 50.68, 50.68      | 2.31      |
| edit_dist_1_md1| 1.47%        | 6.22, 5.79, 5.75, 5.69          | 109.34, 95.45, 67.55, 67.55      | 3.49      |
| edit_dist_a_md2| 1.17%        | 6.02, 5.51, 5.57, 5.51          | 164.00, 164.00, 164.00, 164.00   | 2.59      |
| edit_dis_a_md3 | 2.69%        | 9.11, 7.08, 6.96, 6.93          | 233.00, 233.00, 233.00, 233.00   | 5.91      |
| fft_2_md2      | 11.21%       | 8.84, 7.54, 7.89, 7.76          | 102.94, 73.60, 55.95, 60.55      | 0.70      |
| fft_a_md2      | 0.98%        | 5.03, 4.86, 4.74, 4.70          | 345.50, 345.50, 343.48, 346.50   | 0.69      |
| ff_a_md3       | 1.08%        | 4.73, 4.55, 4.43, 4.42          | 109.62, 109.62, 102.59, 102.59   | 0.63      |
| pici_bridge32_a_md1 | 3.61% | 6.01, 5.64, 5.83, 5.76          | 72.48, 63.76, 63.76, 63.76       | 0.61      |
| pici_bridge32_a_md2 | 8.33% | 9.43, 7.14, 7.55, 7.45          | 186.08, 121.35, 121.35, 121.35   | 0.53      |
| pici_bridge32_b_md1 | 2.55% | 6.35, 6.01, 5.79, 5.72          | 322.71, 332.71, 313.99, 313.99   | 0.52      |
| pici_bridge32_b_md2 | 2.80% | 5.92, 5.53, 5.43, 5.42          | 640.12, 430.04, 430.04, 430.04   | 0.50      |
| pici_bridge32_b_md3 | 3.63% | 6.74, 6.10, 6.13, 6.12          | 398.57, 398.57, 398.58, 398.58   | 0.51      |
| average        | 4.13%        | 6.85, 5.91, 5.93, 5.88          | 260.90, 219.12, 216.57, 216.64   | 2.30      |

row assignment is fixed, the distance to its Global Placement location constitutes a piecewise linear and hence in particular piecewise quadratic function. However, we point out that minimizing linear movement is not the main purpose of our algorithm and that in particular, the assignment to zones is designed to optimize squared instead of movement. Hence, the subsequent comparison should be regarded as proof that our algorithm, even though not explicitly devised to do so, can compete with state-of-the-art legalization concerning linear cell movement. We compare the average linear cell movement achieved by our algorithm to the results obtained by Reference [5] and the state-of-the-art paper [20] as reported in Reference [20] as well as the legalization approach from Reference [1]. Table 2 shows the results for the 16 instances from the ICCAD-2017 CAD Contest on Multi-deck Standard-cell Legalization [8]. It displays the relative increase of the half-perimeter wire length after Global Placement (Δ HPWL), the average linear cell movement (measured in horizontal placement sites), the maximum linear cell movement (again measured in horizontal placement sites) and the runtime in CPU seconds for the algorithms in References [1, 5, 20] and the algorithm suggested in this article (Ours). Concerning the average cell movement, which we are mainly interested in for this comparison, the column labeled “Ours/[20]” contains the percentages the average cell movement obtained by “Ours” constitutes of the average cell movement reported by Reference [20]. The final row labeled “average” displays the average of all prior values in the respective column. In particular, the respective entry in the column “Ours/[20]” refers to the average of the above percentages. One can see that on average, our proposed algorithm achieves comparable results to the algorithm in Reference [20], which in turn produces considerably better results than Reference [5] when it comes to average cell movement. However, the deviation between the different instances is relatively high: While there are some on which our algorithm significantly outperforms the method from Reference [20] (including those where no cells of triple- and quadruple-row height are present), the converse is true for several other test cases. One possible explanation for this might be the fact that the greedy legalization
of cells of more than double-row height only works well if they are sufficiently spaced out in the Global Placement solution, which is true for only some of the given benchmarks. When it comes to running time, maximum movement, and increase in HPWL, our algorithm can be seen to yield comparable or even better results.

The results for the linear movement on 19 instances from the modified ISPD 2015 benchmark [7] are shown in Table 3. Here, we compare our algorithm (column “Ours”) against the results from References [1, 5, 7, 19, 25]. The first column of this table shows the instance name, the next six columns show the total linear movement needed by the indicated algorithms. To simplify the comparison, we show in the last row the average movement relative to the movement needed by our algorithm. As can be seen from these numbers the average linear movement of our algorithm is comparable to state of the art legalizers even though our primary objective is not the linear but the squared movement of the cells.

In the second experiment, we test the effectiveness of the Double-row Algorithm in improving squared cell movement compared to existing algorithms. We compared our new algorithm to the method proposed in Reference [1]. In Table 4, we show the results for 19 instances from the modified ISPD 2015 benchmark [7] while Table 5 contains the corresponding results for the 16 IC-CAD 2017 instances of the CAD Contest on Multi-deck Standard-cell Legalization [8]. Both tables have the same columns. The first column contains the instance name, the next columns contain the HPWL after Global Placement, the total number of movable cells, and the number of single-row and double-row cells and the percentage of double-row cells. The horizontal squared cell movement of the algorithm from Reference [1] (column “[1]”) and our algorithm (column “Ours”) is shown in the next two columns. The last column contains the ratio between these two numbers. The last row in both tables contains the average of the percentages the squared cell movement

| Instance         | [7]-Imp  | [25]   | [5]     | [19]   | [1]    | Ours    |
|------------------|---------|--------|---------|--------|--------|---------|
| des_perf_1       | 279,545 | 474,789| 242,622 | 188,719| 282,695| 249,656 |
| des_perf_a       | 81,452  | 73,057 | 72,561  | 71,049 | 77,234 | 74,815  |
| des_perf_b       | 81,540  | 72,429 | 71,888  | 70,959 | 78,800 | 75,727  |
| edit_dist_a      | 59,814  | 60,971 | 62,961  | 57,264 | 64,635 | 61,780  |
| fft_1            | 54,501  | 53,389 | 46,121  | 38,938 | 61,328 | 50,495  |
| fft_2            | 25,697  | 21,018 | 20,979  | 20,381 | 24,880 | 23,140  |
| fft_a            | 19,613  | 18,150 | 18,304  | 17,897 | 19,626 | 18,822  |
| fft_b            | 28,461  | 21,234 | 21,671  | 20,852 | 23,564 | 22,236  |
| matrix_mult_1    | 80,235  | 73,682 | 71,793  | 61,992 | 82,905 | 75,228  |
| matrix_mult_2    | 75,810  | 65,959 | 65,876  | 58,250 | 75,920 | 68,695  |
| matrix_mult_a    | 46,001  | 40,736 | 40,298  | 39,683 | 41,855 | 40,510  |
| matrix_mult_b    | 40,059  | 37,243 | 37,215  | 36,658 | 37,442 | 36,576  |
| pci_bridge32_a   | 27,832  | 26,674 | 26,289  | 25,960 | 28,508 | 27,495  |
| pci_bridge32_b   | 27,864  | 26,160 | 26,028  | 26,120 | 28,492 | 27,679  |
| superblue11_a    | 1,786,342| 1,983,090| 1,742,941| 1,595,907| 1,954,143| 1,862,013|
| superblue12      | 2,015,678| 1,995,140| 1,963,403| 1,713,915| 2,092,026| 1,984,647|
| superblue14      | 1,599,810| 1,497,490| 1,566,966| 1,330,885| 1,756,323| 1,670,587|
| superblue16_a    | 1,173,106| 1,147,530| 1,135,186| 1,055,668| 1,364,443| 1,293,105|
| superblue19      | 806,529  | 808,164| 781,928 | 705,509| 868,752| 822,387 |
| average          | 0.98     | 1.00   | 0.94    | 0.84   | 1.06   | 1.00    |

A Fast Optimal Double-row Legalization Algorithm

Table 3. Comparison between the Average Linear Cell Movement in Terms of Horizontal Placement Sites for 19 instances from the Modified ISPD 2015 Benchmark [7]
Table 4. Comparison between the Horizontal Squared-cell Movement Resulting from the Legalization Algorithm Described in Reference [1] and Our Algorithm for the 19 Instances from the Modified ISPD 2015 Benchmark [7]

| Instance             | GP HPWL (m) | Movable Cells | Squared Cell Movement |
|----------------------|-------------|---------------|-----------------------|
|                      |             | Total         | Single | Double | #double | Ours     | Others  |
| des_perf_1           | 1.433       | 112,644       | 103,842 | 8,802  | 7.81%   |          |         |
| des_perf_a           | 2.573       | 108,288       | 99,775  | 8,513  | 7.86%   |          |         |
| des_perf_b           | 2.131       | 112,644       | 103,842 | 8,802  | 7.81%   |          |         |
| edit_dist_a          | 5.252       | 127,413       | 121,913 | 5,500  | 4.32%   |          |         |
| fft_1                | 0.456       | 32,281        | 30,297  | 1,984  | 6.15%   |          |         |
| fft_2                | 0.463       | 32,281        | 30,297  | 1,984  | 6.15%   |          |         |
| fft_a                | 0.750       | 30,625        | 28,718  | 1,907  | 6.23%   |          |         |
| fft_b                | 0.952       | 30,625        | 28,718  | 1,907  | 6.23%   |          |         |
| matrix_mult_1        | 2.391       | 155,325       | 152,427 | 2,898  | 1.87%   |          |         |
| matrix_mult_2        | 2.584       | 155,325       | 152,427 | 2,898  | 1.87%   |          |         |
| matrix_mult_a        | 3.772       | 149,650       | 146,837 | 2,813  | 1.88%   |          |         |
| matrix_mult_b        | 3.299       | 146,435       | 143,695 | 2,740  | 1.87%   |          |         |
| pci_bridge32_a       | 0.460       | 29,517        | 26,268  | 3,249  | 11.01%  |          |         |
| pci_bridge32_b       | 0.980       | 28,914        | 25,734  | 3,180  | 11.00%  |          |         |
| superblue11_a        | 42.915      | 925,616       | 861,314 | 64,302 | 6.95%   |          |         |
| superblue12          | 39.110      | 1,286,948     | 1,172,586 | 114,362 | 8.89%   |          |         |
| superblue14          | 27.950      | 612,243       | 564,769 | 47,474 | 7.75%   |          |         |
| superblue16_a        | 31.330      | 680,450       | 625,419 | 55,031 | 8.09%   |          |         |
| superblue19          | 20.722      | 506,097       | 478,109 | 27,988 | 5.53%   |          |         |
| average              |             |               |         |        |         | 73.51%  |         |

Table 5. Comparison between the Horizontal Squared-cell Movement Resulting from the Legalization Algorithm Described in Reference [1] and Our Algorithm on the 16 ICCAD 2017 Instances of the CAD Contest on Multi-deck Standard-cell Legalization

| Instance             | GP HPWL (m) | Movable Cells | Squared Cell Movement |
|----------------------|-------------|---------------|-----------------------|
|                      |             | Total         | Single | Double | #double | Ours     | Others  |
| des_perf_1           | 1.217       | 112,644       | 112,644 | 0      | 0.00%   |          |         |
| des_perf_a_md1       | 2.160       | 108,288       | 103,589 | 4,699  | 4.34%   |          |         |
| des_perf_a_md2       | 2.177       | 108,288       | 105,030 | 1,086  | 1.00%   | 3,393,738 | 2,764,507 | 95.14%   |
| des_perf_b_md1       | 2.106       | 112,644       | 106,782 | 5,862  | 5.20%   |          |         |
| des_perf_b_md2       | 2.137       | 112,644       | 101,908 | 6,781  | 6.02%   |          |         |
| edit_dist_1_md1      | 4.004       | 130,661       | 118,005 | 7,994  | 6.12%   |          |         |
| edit_dist_1_md2      | 5.103       | 127,413       | 115,066 | 7,397  | 6.12%   |          |         |
| edit_dist_3_md1      | 5.328       | 127,413       | 119,616 | 2,599  | 2.04%   | 8,105,841 | 6,756,369 | 98.83%   |
| edit_dist_3_md2      | 0.444       | 32,281        | 28,930  | 2,117  | 6.56%   |          |         |
| fft_2_md2            | 1.092       | 30,625        | 27,431  | 3,080  | 6.79%   |          |         |
| fft_a_md3            | 0.949       | 30,625        | 28,609  | 672    | 2.19%   |          |         |
| pci_bridge32_a_md1   | 0.454       | 29,517        | 26,680  | 2,832  | 6.07%   | 425,596,92 | 397,307 | 93.37%   |
| pci_bridge32_a_md2   | 0.565       | 29,517        | 25,239  | 2,090  | 7.08%   |          |         |
| pci_bridge32_b_md1   | 0.660       | 28,914        | 26,184  | 1,726  | 6.07%   | 894,343,73 | 859,498 | 96.10%   |
| pci_bridge32_b_md2   | 0.574       | 28,914        | 28,038  | 292    | 1.01%   |          |         |
| pci_bridge32_b_md3   | 0.583       | 28,914        | 27,452  | 292    | 1.01%   |          |         |
| average              |             |               |         |        |         | 95.28%  |         |

ACM Transactions on Design Automation of Electronic Systems, Vol. 28, No. 5, Article 71. Pub. date: September 2023.
of our algorithm constitutes of the one the algorithm in Reference [1] achieves. As can be seen, we get an average improvement of 4.72%, respectively, 26.49% for the ICCAD 2017 benchmark, respectively, the modified ISPD 2015 instances.

8 CONCLUSION

In this article, we have presented a fast algorithm to minimize squared (or linear) cell displacement for pairs of cell rows comprising cells of both single- and double-row height with predefined target locations and a fixed left-to-right ordering. Even though the surrounding legalization framework is designed to optimize squared instead of linear cell displacement, our results are competitive when compared to state-of-the-art works on mixed-cell-height legalization. Moreover, experimental results comparing the squared cell displacement when fixing all cells of double-row height and when employing the Double-row Algorithm, respectively, clearly speak in favor of its effectiveness.

ACKNOWLEDGMENTS

We thank the reviewers for their efforts and their numerous helpful comments that have greatly improved the readability of our article.

REFERENCES

[1] U. Brenner. 2013. BonnPlace Legalization: Minimizing Movement by Iterative Augmentation. IEEE Trans. Comput.-Aided Design Integr. Circ. Syst. 32, 8 (2013), 1215–1227.
[2] U. Brenner and J. Vygen. 2000. Faster Optimal Single-row Placement with Fixed Ordering. In Proceedings of the Design, Automation and Test in Europe. 117–121.
[3] U. Brenner and J. Vygen. 2004. Legalizing a Placement with Minimum Total Movement. IEEE Trans. Comput.-Aided Design Integr. Circ. Syst. 23, 12 (2004), 1597–1613.
[4] I. Bustany, D. Chinnery, J. Shinnerl, and V. Yutis. 2015. ISPD 2015 benchmarks with fence regions and routing blockages for detailed-routing-driven placement. In Proceedings of the International Symposium on Physical Design (ISPD’15). 157–164.
[5] J. Chen, Z. Zhu, W. Zhu, and Y. Chang. 2017. Toward Optimal Legalization for Mixed-Cell-Height Circuit Designs. In Proceedings of the 54th Annual Design Automation Conference (DAC’17). 6.
[6] Y. Cheng, D. Huang, W. Mak, and T. Wang. 2018. A Practical Detailed Placement Algorithm under Multi-Cell Spacing Constraints. In Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD’18). 8.
[7] W. Chow, C. Pui, and E. Young. 2016. Legalization Algorithm for Multiple-Row Height Standard Cell Design. In Proceedings of the 53rd Annual Design Automation Conference (DAC’16). 1–6.
[8] N. Darav, I. Bustany, A. Kennings, and R. Mamidi. 2017. ICCAD-2017 CAD Contest in Multi-deck Standard-cell Legalization and Benchmarks. In Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD’17). 867–871. Retrieved from http://iccad-contest.org/2017/Problem_C_default.html.
[9] M. Garey and D. Johnson. 1978. “Strong” NP-Completeness Results: Motivation, Examples, and Implications. J. ACM 25, 3 (1978), 499–508.
[10] M. Garey, R. Tarjan, and G. Wilfong. 1988. One-Processor scheduling with symmetric earliness and tardiness penalties. Math. Oper. Res. 13, 2 (1988), 330–348.
[11] C. Han, K. Han, A. Kahng, H. Lee, L. Wang, and B. Xu. 2017. Optimal Multi-Row Detailed Placement for Yield and Model-Hardware Correlation Improvements in Sub-10nm VLSI. In Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD’17). 667–674.
[12] C. Han, A. Kahng, L. Wang, and B. Xu. 2019. Enhanced Optimal Multi-Row Detailed Placement for Neighbor Diffusion Effect Mitigation in Sub-10 nm VLSI. IEEE Trans. Comput.-Aided Design Integr. Circ. Syst. 38, 9 (2019), 1703–1716.
[13] D. Hill. 2002. Method and system for high speed detailed placement of cells within an integrated circuit design. U.S. Patent 6370673. (2002).
[14] Stefan Hougardy, Meike Neuwohner, and Ulrike Schorr. 2021. A fast optimal double-row legalization algorithm. In Proceedings of the International Symposium on Physical Design (ISPD’21). ACM, 2330. DOI:https://doi.org/10.1145/3439706.3447044
[15] C. Hung, P. Chou, and W. Mak. 2017. Mixed-cell-height Standard-cell Placement Legalization. In Proceedings of the Great Lakes Symposium on VLSI. 149–154.
[16] A. Kahng, P. Tucker, and A. Zelikovsky. 1999. Optimization of Linear Placements for Wirelength Minimization with Free Sites. In Proceedings of the Asia and South Pacific Design Automation Conference. 241–244.
[17] B. Korte, D. Rautenbach, and J. Vygen. 2007. BonnTools: Mathematical Innovation for Layout and Timing Closure of Systems on a Chip. Proc. IEEE 95 (2007), 555–572.

[18] H. Li, W. Chow, G. Chen, E. Young, and B. Yu. 2018. Routability-driven and fence-aware legalization for mixed-cell-height circuits. In Proceedings of the 55th Annual Design Automation Conference (DAC’18). 1–6.

[19] Haocheng Li, Wing-Kai Chow, Gengjie Chen, Bei Yu, and Evangeline F. Y. Young. 2022. Pin-Accessible legalization for mixed-cell-height circuits. IEEE Trans. Comput.-Aided Design Integr. Circ. Syst. 41, 1 (2022), 143–154. DOI: https://doi.org/10.1109/TCAD.2021.3053223

[20] X. Li, J. Chen, W. Zhu, and Y. Chang. 2019. Analytical mixed-cell-height legalization considering average and maximum movement minimization. In Proceedings of the International Symposium on Physical Design (ISPD’19). 27–34.

[21] Y. Lin, B. Yu, X. Xu, J. Gao, N. Viswanathan, W. Liu, Z. Li, C. Alpert, and D. Pan. 2016. MrDP: Multiple-row Detailed Placement of Heterogeneous-sized Cells for Advanced Nodes. In Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD’16). 1–8.

[22] P. Spindler, U. Schlichtmann, and F. Johannes. 2008. Abacus: Fast Legalization of Standard-cell Circuits with Minimal Movement. In Proceedings of the International Symposium on Physical Design (ISPD’08). 47–53.

[23] U. Suhl. 2010. Row-Placement in VLSI Design: The Clumping Algorithm and a generalization. Diploma thesis, University of Bonn, Research Institute for Discrete Mathematics.

[24] R. Tarjan. 1983. Data Structures and Network Algorithms. SIAM.

[25] C. Wang, Y. Wu, J. Chen, Y. Chang, S. Kuo, W. Zhu, and G. Fan. 2017. An Effective Legalization Algorithm for Mixed-cell-height Standard Cells. In Proceedings of the 22nd Asia and South Pacific Design Automation Conference. 450–455.

[26] G. Wu and C. Chu. 2015. Detailed Placement Algorithm for VLSI Design with Double-row Height Standard Cells. IEEE Trans. Comput.-Aided Design Integr. Circ. Syst. 35 (2015), 1569–1573.

[27] Z. Zhu, X. Li, Y. Chen, J. Chen, W. Zhu, and Y. Chang. 2018. Mixed-Cell-Height Legalization Considering Technology and Region Constraints. In Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD’18). 8.

Received 14 August 2022; revised 22 December 2022; accepted 25 December 2022