Interconnect-driven Cell-based Migration of Integrated Circuit Layout

Research Thesis

In Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Science

Evgeny Shaphir

Submitted to the Senate of the Technion - Israel Institute of Technology

Nisan, 5769 Haifa April 2009
This Research Thesis Was Done Under the Supervision of Assoc. Prof. Shmuel Wimer, Department of Electrical Engineering, Technion and the School of Engineering, Bar-Ilan University, and Assoc. Prof. Ron Y. Pinter, Department of Computer Science, Technion, in the Department of Computer Science.
# Table of Contents

1. INTRODUCTION ...................................................................................................... 3

2. THE MIGRATION FRAMEWORK .......................................................................... 6
   2.1 The Placement-Routing Handshake ............................................................ 6

3. THE INTERCONNECT MIGRATION ALGORITHM .............................................. 12
   3.1 The Graph Model ....................................................................................... 12
   3.2 Description of the Algorithm ...................................................................... 13
      3.2.1 Construction of the Layout Graph ...................................................... 15
      3.2.2 Merge ................................................................................................. 16
      3.2.3 Reduce ............................................................................................... 19
      3.2.4 Exact Solution Construction ............................................................... 20

4. CORRECTNESS OF THE INTERCONNECT MIGRATION ALGORITHM ....... 23
   4.1 Layouts and Graphs ..................................................................................... 23
   4.2 The Algorithm's Invariants ......................................................................... 29
      4.2.1 Merge ................................................................................................. 29
      4.2.2 Reduce ............................................................................................... 39
      4.2.3 Exact Solution Construction ............................................................... 41

5. EXPERIMENTAL RESULTS .................................................................................. 49

6. DISCUSSION AND FUTURE RESEARCH ........................................................... 51

7. REFERENCES ......................................................................................................... 52
List of Figures

Figure 1: (a) A hierarchical layout comprising four levels of hierarchy and (b) the corresponding calling tree. The black-frame cells are library leaf cells................................. 7

Figure 2: Complete block layout migration flow................................................................. 9

Figure 3: Parent-son cell placement and its X and Y visibility graphs. Weights are assigned to vertices according to sons’ width and height in target process. Weights are assigned to arcs according to the space between the corresponding son cells weighted by some scaling factor. .......................................................................................................... 10

Figure 4: A one-layer layout and the associated visibility graph ..................................... 12

Figure 5: Bottom-up flow block diagram ........................................................................ 14

Figure 6: Top-down flow block diagram........................................................................... 14

Figure 7: Construction of a Flat Layout Visibility Graph – pseudo code......................... 15

Figure 8: Construction of placement arcs – pseudo code ................................................. 16

Figure 9: A one-layer layout and the associated Flat Visibility Layout Graph: gray vertices represent walls, blue and green – polygons, red arcs represent placement constraints ......................................................................................................................... 16

Figure 10: (a) The hierarchical layout and (b) the underlying hierarchy graph. The hierarchy order is A,B,C,D,E........................................................................................... 17

Figure 11: Merge phase – pseudo code ............................................................................ 18

Figure 12: Merged graph – generated by Merge on Layout Graph G .............................. 19

Figure 13: Reduce template – pseudo code ...................................................................... 20

Figure 14: Reduced graph – generated by reduceTemplateInstances reducing template B from Layout Graph G......................................................................................................................... 20

Figure 15: Exact solution derivation – pseudo code......................................................... 21
Figure 16: (a) Reduced Graph - the numbers next to vertices represent the coordinates for
the vertices that already obtained the solution. (b) Constructing an LP problem from the
graph. ................................................................................................................................ 22

Figure 17: Coordinates of an instance $i$ and polygons in vertically oriented layout. .... 24

Figure 18: (a) A layout and (b) the underlying Flat Visibility Layout Graph. ............. 25

Figure 19: (a) A layout and (b) the underlying Hierarchy Constrained Graph............... 26

Figure 19: The path in $H_t$. Similarity arcs are in red. Blue represents the path
$x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_n$ ................................................................. 31

Figure 20: The path in $H_t$. Similarity arc is red. Blue represents the paths $x_1 \rightarrow \ldots \rightarrow x_n$ and
$y_1 \rightarrow \ldots \rightarrow y_m$ .................................................................................. 35

Figure 21: The path $\pi$ in $H_t$. Blue represents the paths $x_1 \rightarrow \ldots \rightarrow x_n$ and $y_1 \rightarrow \ldots \rightarrow y_m$ .... 36

Figure 22: Destination Layout vs. Source Layout Net Delays. Encircled data points
represent nets of significant delays which have been badly scaled.............................. 49

Figure 23: Tests statistics - size of graphs and runtime .............................................. 50

**List of Tables**

Table 1: Equivalence between the properties of layouts and graphs .......................... 28

Table 2: Tests statistics – size of graphs and runtimes .............................................. 50
ABSTRACT

Time to market and cost are primary drivers to preserve designs of integrated circuits, calling for effective migration methods. Moreover, it is desirable to maintain the hierarchical nature of a design for performance, use of libraries, and maintainability. Thus, flattening a layout for migration is inappropriate both due to the huge data structures that might arise as well as the loss of the hierarchical information.

We provide a novel, cell-based method for migrating hierarchical designs, while meeting performance constraints imposed on the signals goals such as delay, power and signal integrity. Our algorithm separates the placement and interconnect considerations in a clear and effective manner. It supports a bottom-up flow with emphasis on efficient interconnect migration, thus addressing the most difficult challenge in cell-based compaction. The algorithm's input is a source layout, a target layout sizing together with a placement, the target's manufacturing technology design rules, and target design performance specifications; it uses a hierarchy of directed graphs to represent the constraints of the interconnect layout in a non-redundant fashion. The complete solution is derived top-down, formulating and solving a Linear Programming problem, treating a single level of the hierarchy at the time. It satisfies the new design rules, a good starting point for convergence, preserving hierarchies and wires' order. We prove that the algorithm either returns a legal routing or manifests contradictory constraints. Finally, we have implemented our method and applied it to a number of microprocessor designs, demonstrating a significant reduction of time complexity without sacrificing the quality of the resulting layout.
### List of Symbols and Abbreviations

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASIC</td>
<td>Application-specific Integrated Circuit</td>
</tr>
<tr>
<td>BFS</td>
<td>Breadth First Search</td>
</tr>
<tr>
<td>DFM</td>
<td>Design For Manufacturability</td>
</tr>
<tr>
<td>DFS</td>
<td>Depth First Search</td>
</tr>
<tr>
<td>FVLG</td>
<td>Flat Visibility Layout Graph</td>
</tr>
<tr>
<td>HCG</td>
<td>Hierarchy Constrained Graph</td>
</tr>
<tr>
<td>IC</td>
<td>Integrated Circuit</td>
</tr>
<tr>
<td>LP</td>
<td>Linear Programming</td>
</tr>
<tr>
<td>LPW((u, v))</td>
<td>The Weight of the Longest Path between the vertices (u) and (v)</td>
</tr>
<tr>
<td>TSA</td>
<td>Template Similarity Arc</td>
</tr>
<tr>
<td>VG</td>
<td>Visibility Graph</td>
</tr>
<tr>
<td>VLSI</td>
<td>Very Large Scale of Integration</td>
</tr>
<tr>
<td>W((u, v))</td>
<td>The Weight of arc ((u, v))</td>
</tr>
</tbody>
</table>
1. INTRODUCTION

The strong competition in the high–end microprocessor market [1,2] have lead IC manufacturers to adopt an aggressive marketing strategy in which every year brands of new processors, covering a wide spectrum of applications, are being delivered to market. VLSI process technology, on the other hand, follows Moore’s Law [3], with a new process technology every two years [4].

The gap between the one–year product cycle and the two–year technology period, together with the tremendous VLSI design effort required for full–custom high–end products, have lead to a marketing policy called Tick–Tock [5]: a new core – featuring major architectural enhancements – is designed every two years, such that new design and new technology are progressing in one–year overlap. The new core designed in the \( n^{th} \)–generation process node and being sold in low volumes, is then migrated into the \((n+1)^{st}\)–generation process node and sold in high volumes, where its performance is significantly improved in both higher speed and lower power consumption. The transition of a core across process generations mostly involves re–design of circuit and layout due to the new process technology.

The above circumstances raise the need for a very fast full–custom backend redesign cycle for which traditional design methods are inappropriate. Since layout artwork effort is very significant, layout skills are very expensive, and the design period is very short, new layout automation methods have become a must. Among those methods the automatic migration of layout, called also hard–IP reuse, was found as a major productivity booster [6]. More importantly, the preservation of the relative position of layout objects, whose careful design in \( n^{th} \)–generation required a huge investment, ensures the fast convergence and timing closure in the new design.

The key technology involved in hard–IP reuse is layout compaction [7,8,9]. This well–known technology has been widely used for several decades to migrate small–scale VLSI layouts, such as standard–cell and custom libraries [10], data–path macros [11,12], control blocks [13], and DFM enhancements [14]. The complexity of compaction algorithms and the desire to preserve the design hierarchy has delayed the usage of full–
chip automatic compaction until recently, when some novel commercial tools have been made available to cope with large-scale fully hierarchical layouts [6].

Hierarchical compaction has been used successfully at e.g. Intel for several process generations by now, from 130nm, through 90nm and 65nm, to 45nm. Lately, lithography limitations and DFM requirements have significantly increased the complexity of layout compaction. Moreover, though hierarchical compactors fully preserve the hierarchy of a layout and each standard cell of the source layout is modified into a unique target cell complying with all its instances, a separate migration of several different layouts might create several target mutations of the same cell, each for a distinct source layout. Cell mutations are major disadvantage in full-custom design, since they avoid the usage of central cell characterization. The need for a unique common cell library is a key factor for design efficiency and productivity.

Thus, a new layout migration technology called cell-based compaction [15], has evolved. It uses a common standard cell library for the entire project, which is optimized regardless of its cell instances in the entire layout. The migration of a given functional block takes place bottom-up, starting from the leaf standard cells, building the intermediate hierarchies, up to the top of the block, which is a basic entity in full-custom design.

This thesis scales the cell-based approach to entire processor design. Its goal is to preserve the design intent and meet predefined performance goals, such that the layout obtained will be a good starting point for fast convergence. It uses a common standard cell library for the entire project, which is optimized regardless of its cell instances in the entire layout. It uses a bottom-up flow with emphasis on efficient interconnect migration, thus addressing the most difficult challenge in cell-based compaction. The input to the interconnect migration phase is the source layout, the blocks' placement in the target layout, the target's manufacturing technology design rules, and wire width and space specifications as derived from delay considerations. The migration of a given functional block takes place bottom-up, starting from the leaf standard cells, building the intermediate hierarchies, up to the top of the block, which is a basic entity in full-custom design. The complete solution is then derived top-down, formulating and solving a
Linear Programming problem, treating one level of the hierarchy at a time; it satisfies the new design rules, a good starting point for convergence, and preserving hierarchies and wires' order.

The rest of the thesis is organized as follows. Section 2 describes the general framework of the migration process and provides its placement portion. Section 3 presents the interconnect migration problem and outlines its solution by using a suitable graph theoretic model. Section 4 provides the mathematical foundations of the algorithm in use, with an outline of a rigorous proof for its correctness. Section 5 presents experimental results achieved on industrial full–custom VLSI blocks of a 45nm process technology. We conclude with some challenges for further research.
2. THE MIGRATION FRAMEWORK

2.1 The Placement-Routing Handshake

Figure 1a illustrates a hierarchical layout comprising several cells and their interconnecting wires. Various hierarchical layers are drawn by distinct colors, where black cells are library leaf cells. Wires of vertical and horizontal routing layers are drawn in different colors, while connecting vias reside at their incidence point. Wires connecting leaf cells within their parents are drawn by solid lines, while upper level wires are drawn by dotted lines. The corresponding call–tree is drawn in Figure 1b. We denote by a template the master cell that is being instantiated in the design. The occurrences of a template are denoted by instances. The physical location of the instance (its left down corner) is denoted by the instance's offset.

The transition from one process technology to its successor is a uniform scaling of all design rules by a common factor, usually nearly 0.7, thus yielding area scaling of 0.5 across process migration. Unfortunately, ideal scaling is impossible and strong nonlinearity occurs, which becomes worse in 32nm and smaller process nodes. The electrical characteristics of interconnecting wires are also scaled differently on different layers, so the width and space specifications of every net are recalculated to retain performance goals such as delay, power, signal integrity and IR drop. This is further introducing nonlinearity to the scaling. As a result, standard cells, custom leaf-cells and intermediate hierarchical blocks scale nonlinearly, and their X and Y scaling may typically be distributed from 0.6 to 0.9 according to a variety of geometric and circuit considerations that are beyond the scope of this thesis. The non-linearity is propagated all the way up to the layout's root, where the area consumed by every intermediate hierarchical block is the outcome of the area of its sons and the scaling of the interconnecting wires.
Implementation of bottom-up migration by a Depth First Search (DFS) traversal of the layout hierarchy tree is self evident. We start with migrating parent cells calling leaf cells by positioning the leaf cells of the new process technology in the same relative left-to-right and bottom-to-top order as in the source layout. Here we account also for the projected routing area in order to accommodate interconnecting wires. We then complete those by using the same routing patterns as in the source layout. Clearly, the preservation of hierarchy requires that every intermediate hierarchy is migrated only once and then used in its multiple instances. Such a bottom-up migration flow is far more efficient than and advantageous over ordinary hierarchical compaction as it deals with one block at a

Figure 1: (a) A hierarchical layout comprising four levels of hierarchy and (b) the corresponding calling tree. The black-frame cells are library leaf cells.
time, knowing the exact dimensions of all its called cells, which have already been migrated. In contrast, traditional full-size hierarchical compaction solves the constraint graph of the entire layout at once, where the identity of cell instances is maintained by imposing a huge amount of extra constraint equations.

There is a problem with the above paradigm, stemming from interlace of same–layer interconnects across various hierarchy levels. Unlike place–and–route layout implementation widely used in ASIC design where all interconnecting wires reside in same hierarchy level, full–custom design style consists of many hierarchy levels, so interleave of wires across various hierarchies is unavoidable as shown in Figs 1a and 2. The above bottom–up migration however is blind to upper level wires, which may constrain the position and size of its wires in given cell. Moreover, different instances of same cell may see different upper constraining wires, so their adherences in one instance doesn’t guarantee adherence of other instances. This problem is handled in full–size hierarchical compaction by solving simultaneously all the instance constraints, which is robust but very time consuming [16].

The entire migration flow of our process is shown in Figure 2. In addition to reading the source layout and target process technologies, width and space specifications for every wire in every net are read and subsequently derived such that the performance measures mentioned before are retained [17]. The placement and routing migration are carried out in two successive steps, with internal feedback to ensure convergence to feasible and high-quality layout realization. A purely placement-oriented migration is carried out first by estimating the area resources required for the wiring migration which follows. If the wiring subject to the given placement and the area resources is found feasible, a routing commitment takes place which assigns accurate width and position to every wire. If however a feasible solution does not exist two actions can take place. In the first the placement of the underlying block is relaxed in order to accommodate interconnect constraints, at the expense of area growth. Alternatively, the constraints are relaxed, so the final layout will include violations that will later be fixed manually at the expense of the design effort.
For the sake of completeness we’ll briefly describe the placement migration part of the bottom-up flow and how it accommodates the area resources required for the migration of routing which follows later. By the very definition of bottom-up, all the sons of a presently migrated intermediate hierarchy cell are already done; hence their size in target process is defined. A visibility graph \( G(U, E, \bar{w}, \bar{s}) \) is then derived from the source layout, defined as follows (see Figure 3). Assuming that there are \( n \) son blocks, the left and right borders of the migrated parent cell correspond to the source and target vertices, denoted by \( u_0 \) and \( u_{n+1} \), respectively. Every son cell corresponds to a vertex \( u_i \), \( 1 \leq i \leq n \).

Two vertices \( u_i \) and \( u_j \) are connected with a directed arc \( e(u_i, u_j) \) if son cells \( i \) and \( j \) are visible to each other in source layout.

**Figure 2: Complete block layout migration flow**
Thus defined is a directed acyclic planar graph having one source and one sink. A node $u \in U$ is assigned with a weight $w(u)$ equals the width of the corresponding migrated son cell in target process. An arc $e \in E$ is assigned with a weight $s(e)$ equals the distance between the corresponding cells in source layout, multiplied by some scaling factor which accounts for the process technology scaling factor and the amount of wires that are crossing the space between the blocks. The derivation of scaling factor for an arc is pretty delicate and its discussion is beyond the scope of this thesis.

A graph $H(V, F, \bar{w}, \bar{s})$ is defined similarly to represent the vertical relations between son cells and define the vertical dimension of the parent cell. In these definitions the dimensions of parent cell in target process are defined by the longest paths from source to sink in $G$ and $H$, where the location of the son cells within their parent are defined by the longest paths from sources to the corresponding vertices in $G$ and $H$. At this point the placement migration of a cell has been finished and DFS proceeds. Once the root is
reached the placement migration of the entire block is done and routing migration takes place.

The next sections present an efficient and accurate solution to the hierarchical wire interlace problem, which ensures the solution of interconnect migration. It detects whether the given cell area can accommodate certain layer routing requisites imposed by the source layout, target process and performance specifications (via wire width and space). If the answer is positive, the width and position of wires is committed, where net connectivity is retained by stitching the wires of orthogonal layers to the ends of those migrated. Such stitching is possible due to a strict methodology of using layers in alternating routing directions. Then the next layer is verified. If however it is found that the area available after placement cannot accommodate legal routing in a given layer, this information is fed back to the placement algorithm for another iteration which accounts for the discovered wire congestion, or some constraints are relaxed and left for later resolution by manual work of a mask designer or by circuit fixes. This placement-routing handshaking migration continues until all congestions are resolved and a committed layout is produced. The flow in Figure 2 guarantees that the area, process technology rule and performance constraints in the target design are under control, leaving low design effort to reach convergence. The experiment in Section 5 demonstrates by how much performance constraints imposed on interconnects are met.
3. THE INTERCONNECT MIGRATION ALGORITHM

3.1 The Graph Model

The common method to represent the adjacency relation between interconnect features in a layout is by a directed graph, where vertices represent polygons, and edges – the visibility relations of polygons. For purposes of presentation, we limit our discussion throughout to one layer; still, our solution to the layout migration problem can be extended to the general case layout, with multiple-layer layout. Independent treatment of layers is possible because of the following reasons:

- The layers are treated in sequential order – thus the changes to polygons’ coordinates of the layer $n-1$ are fixed by changing the length of the polygons of layer $n$ (which is orthogonal to $n-1$)
- Statistically the changes to polygons coordinates of the layer $n-1$ are negligible relatively to the length of polygons of the layer $n$.
- The layout produced by the algorithm is not expected to be clean: final routing fixes to be done by either specific application or mask designers.

Figure 4 shows the one-layer layout with vertical routing orientation and the associated visibility graph. In this case the compaction problem is translated into the *longest path problem* in a graph [7], [8], [23]. The algorithm for finding longest paths is also used for positive cycles’ detection, thus establishing the feasibility of the layout itself.

![Figure 4: A one-layer layout and the associated visibility graph](image.png)
3.2 Description of the Algorithm

The interconnect migration flow that is presented in this section is aimed to solve the migration of routing from old technology to the new one for a single layer and satisfy the requirements as stated below.

Given the layer interconnect (set of rectangles) at source manufacturing technology and blocks’ sizing and placement (blocks’ borders coordinates) at the target manufacturing technology the flow generates a legal layer interconnect (set of rectangles) at the target manufacturing technology that satisfies:

1. Preserves the polygon order (derived from visibility relation)
2. Preserve the hierarchical structure
3. Satisfy the new sizing and placement at the target manufacturing technology
4. Satisfies the new manufacturing technology design rules

In the next section we prove the correctness of the flow subject to these requirements.

At the time the visibility graph is being built, the instances of sub–graphs of the same hierarchy are marked; this information is used later in the Merge phase to obtain a compact solution for the problem. Then, during the Reduce phase, we construct a concise description of the compaction problem at each level of the hierarchy. Once solved, the algorithm will transfer to the upper layout hierarchy levels only the data that is essential for the upper level solutions. Note that this data represents a family of solutions, which in turn will result in a related family of solutions at the upper levels.

At the end of this bottom–up propagation of solutions, all the solutions are obtained at the root of the hierarchical layout. The block diagram for the bottom–up part of the flow is presented in Figure 5.
The commitment of the exact locations of the interconnecting polygons takes place top–down (Figure 6). There, the decision of the polygons' locations at a certain level of the hierarchy is made in order to optimize some objective function, reflecting circuit and layout considerations. The specific solution thus obtained imposes constraints on the solution family of its layout children. These are then being solved independently of each other.

**Figure 5: Bottom-up flow block diagram**

The commitment of the exact locations of the interconnecting polygons takes place top–down (Figure 6). There, the decision of the polygons' locations at a certain level of the hierarchy is made in order to optimize some objective function, reflecting circuit and layout considerations. The specific solution thus obtained imposes constraints on the solution family of its layout children. These are then being solved independently of each other.

**Figure 6: Top-down flow block diagram**
3.2.1 Construction of the Layout Graph

The first phase of the top down flow is the generation of a Flat Visibility Layout Graph (FVLG) $G$ induced by the source layout (of the old manufacturing technology) and placement (of the new manufacturing technology). The FVLG will be formally defined later. The generation of the graph is performed in two main phases: (1) visibility graph construction and (2) addition of placement arcs.

The pseudo code for Phase 1 is presented in Figure 7. Every polygon or block border becomes a vertex of $G$. The arcs of $G$ represent the visibility relation between the layout features. The weights of the arcs are defined by the new technology design rules. Roughly speaking, minimum space between any two layout features is the minimum distance allowed by the manufacturing technology measured from the centers of those features.

```plaintext
1. foreach el (layout.polygons, layout.borders) {
2.     flatGraph.addVertex(el)
3. }
4. foreach el (layout.polygons, layout.borders) {
5.     foreach nb (el.visibleElementsOnRight) {
6.         /*Skip non-relevant relations*/
7.         if(el.isBorder && nb.isPolygon && nb != el.instance ||
8.             nb.isBorder && el.isPolygon && el != nbr.instance)
9.             continue;
10.        flatGraph.addArc(el, nb, minSpace);
11.    }
12. }
```

**Figure 7: Construction of a Flat Layout Visibility Graph – pseudo code**

Phase 2 adds the arcs to the graph generated by Phase 1; the arcs represent the constraints created by placement. For every instance, two arcs added to the graph: these two arcs ensure that the size of the block remains equal to the size defined by placement. The pseudo code for Phase 2 is presented in Figure 8.
3.2.2 Merge

The next step in top–down flow is the generation of a merged graph $M$ from the Layout Graph $G$; the aim is both to reduce the graph's complexity as well as to add the similarity constraints between all instances of the same template.
This is done iteratively for all the templates in reverse *hierarchical order* (defined later) from the leaf cells to the top cell, concentrating on one template at every iteration. All occurrences of some polygon in the instances of the treated template are merged into a single vertex. The weights of the arcs to/from the treated vertex are updated accordingly. This significantly reduces the size of the graph on one hand, and ensures the similarity between all instances of the same template on the other hand.

The hierarchical order between the templates is defined by the containment relation: for any two templates $a,b$ define $a \prec b \iff a \subseteq b$. The order can be obtained from the topological order of the hierarchy graph as shown in Figure 10.

The pseudo code for the procedure `mergeTemplateInstances` is presented in Figure 11. This procedure is iteratively called for all templates in $G$, as shown in Figure 5. At the beginning of the phase, the vertices of the flat graph $G$ represent the occurrences of polygons. The aim of the Merge phase is to replace all vertices that represent the occurrences of the same polygon by a single vertex. Denote by $\nu(i)$ the vertex that represents the occurrence of some template vertex $\nu$ in instance $i$. First we add a new vertex $\nu$ to the graph (Line 2); $\nu$ is a single vertex that will replace all the occurrences. For every arc $(u, \nu(i))$ entering the occurrence vertex $\nu(i)$ in instance $i$, a new arc is added to the graph, to the newly introduced template vertex $\nu$, that represents the vertex $\nu(i)$. The weight of the new arc is decreased by the offset of the instance $i$ of vertex $\nu(i)$ (Line 7). The original arc is removed. Similarly, the arcs going from the vertex $\nu(i)$ are replaced.

![Diagram](image_url)
by arcs from \( v \); the weight of those arcs are increased by the offset of the instance \( i \) (Line 10).

We prove later that the merged graph maintains the property of the original layout graph: the feasible solution exists if and only if there are no positive cycles in the graph. Thus the check of positive cycles can be done at this step on the smaller merged graph. Once positive cycles are detected, the layout does not have a feasible solution, and a false status is returned as in Figure 5. However, instead of termination, a relaxation can be done at this step: some of constraints can be removed or relaxed. This will produce a partially legal layout that is ready for further usage (e.g. performance verification analysis), depending on project requirements.

```
proc mergeTemplateInstances(graph, template) {
    1. foreach vertex (template.vertices) {
        2.   graph.addVertex(vertex);
    }
    3. foreach inst (template.instances) {
        4.   foreach tempV (template.vertices) {
            5.       instV ← instance.getInstance(templateV.name);
            6.       foreach e (instanceV.inArcs) {
                7.           graph.addArc(e.vertex1, tempV,
                                         e.weight−inst.offset);
                8.           graph.removeArc(e);
            }
            9.       foreach e (instanceV.outArcs) {
                10.          graph.addArc(tempV, e.vertex2,
                                 e.weight+inst.offset);
                11.          graph.removeArc(e);
            }
            12.       graph.removeVertex(instanceV);
        }
    }
}
```

**Figure 11: Merge phase – pseudo code**
Note that the size of the FVLG is proportional to the size of the whole layout, whereas the size of the merged graph is proportional to the sum of sizes of the design templates. In a hierarchical design the latter is significantly smaller than the FVLG we started with.

![Figure 12: Merged graph – generated by Merge on Layout Graph G](image)

### 3.2.3 Reduce

The next phase of the flow is a construction of sequence of graphs called *reduced graphs* – one for every template. This phase is also performed iteratively for all the templates in reverse hierarchical order (from the leaf cells to the top cell), considering a single template at a time. At each iteration, while treating the template \( t \), all vertices that belong to \( t \) are removed from the graph. For every pair of arcs of a vertex \( v \) that is being removed, one entering \( v \) – say \((u, v)\) – and one leaving \( v\), \((v, w)\), a new arc \((u, w)\) is created with a weight that equals the sum of the weights of the two arcs (Line 5). The procedure `reduceTemplateInstances` is presented in Figure 13. During the Reduce phase this procedure is iteratively called for all the templates in the design in template hierarchical order. The graph generated by procedure `reduceTemplateInstances` after reducing template B in Merged Graph from Figure 12 is presented in Figure 14.
3.2.4 Exact Solution Construction

The next step is the derivation of the exact locations (coordinates) of the interconnect polygons. This is done by a top–down flow, iterating on templates in a hierarchical order – from the top–level block to leaf cells, see Figure 6. At each iteration, the coordinates of all the polygons of treated templates are derived. The linear programming problem is being built from the reduced graph: every variable represents the polygon coordinate and

```cpp
proc reduceTemplateInstances(graph, template) {
1. reducedGraph ← graph;
2. foreach vertex (template.vertices) {
3.     foreach inArc (vertex.inArcs) {
4.         foreach outArc (vertex.outArcs) {
5.             reducedGraph.addArc(inArc.vertex1, outArc.vertex2, inArc.weight+outArc.weight);
6.         }
7.     }
8.     reducedGraph.removeVertex(vertex);
9. }
10. return reducedGraph;
}
```

**Figure 13: Reduce template – pseudo code**

![Reduced graph](image)

**Figure 14: Reduced graph – generated by reduceTemplateInstances reducing template B from Layout Graph G**
a constraint represents the arc. The solution of the LP problem determines the location of the polygons.

Figure 15 presents the pseudo code for the LP problem construction from the reduced graph. At every iteration, the inductive property is that the polygons of the templates that precede the treated template in hierarchical order are already treated, i.e. all the polygons of those templates already derived their location. This information is used in the generation of constraints in lines 2–7: the coordinates of the polygons are stored in an array called SOL; polygons which coordinates that have not yet been calculated are represented by ∅.

```java
1. foreach e (graph.arcs) {
2.   if(SOL[e.vertex1]=∅ && SOL[e.vertex2]=∅)
3.     addConstraint(e.vertex2−e.vertex1 ≥ e.weight);
4.   else if(SOL[e.vertex1]=∅ && SOL[e.vertex2]≠∅)
5.     addConstraint(SOL[e.vertex2]−e.vertex1 ≥ e.weight);
6.   else if(SOL[e.vertex1]≠∅ && SOL[e.vertex2] ≠∅)
7.     addConstraint(e.vertex2−SOL[e.vertex1] ≥ e.weight);
}
8. LPsolver.solve();
9. foreach vertex ( getVertices(graph) )
10.   if(SOL[vertex]=∅)
11.     SOL[vertex] ← LPsolverSolution(vertex)
}
```

**Figure 15: Exact solution derivation – pseudo code**

The solution of the LP that is constructed in the previous stage contains the coordinates of the polygons in the template. In the next section we present the proof that a solution always exists and it is feasible, i.e. the layout is legal in terms of new manufacturing technology design rules. Note that the size of the LP problem is bounded quadratically by the number of template polygons; this follows directly from the fact that the number of variables is the number of template polygons and all constraints involve at most 2 variables.
Figure 16: (a) Reduced Graph - the numbers next to vertices represent the coordinates for the vertices that already obtained the solution.

(b) Constructing an LP problem from the graph.

\[ \begin{align*}
v1 - 0 & \geq 0 \\
v1 - 33 & \geq -31 \\
46 - v1 & \geq 44 \\
v2 - v1 & \geq 3 \\
v2 - 46 & \geq -31 \\
22 - v2 & \geq 6 \\
33 - v2 & \geq 7 \\
v3 - v2 & \geq 4 \\
v3 - 22 & \geq 4 \\
30 - v3 & \geq 0 \\
33 - v3 & \geq 7 \\
74 - v3 & \geq 46
\end{align*} \]
4. CORRECTNESS OF THE INTERCONNECT MIGRATION ALGORITHM

In this section we prove the following properties of the algorithm presented in section 3:

- If the layout has a feasible solution, the algorithm will return one.
- Any feasible solution can be derived using the algorithm (depending on the objective function used in the LP problem’s solution).
- The solution returned by the algorithm represents a legal layout and satisfies the requirements that were stated in Section 3.2.

This section is organized as follows. We start with a formal definition of a layout and its properties in Section 4.1. Then we define graphs that are derived from the layout and its properties. This is followed by a proof of the equivalence between graph and layout properties that are stated in Propositions 1–3. In Section 4.2 we present the definitions related to the bottom–up routing migration algorithm, and prove its invariants in Propositions 4–8. We conclude with definitions related to the exact solution derivation part of the routing migration algorithm, and prove its properties in Propositions 9 and 10. Lemmas 1 and 2 capture the main results of this chapter.

4.1 Layouts and Graphs

Definition 1: A layout $L$ is a hierarchical tree, where every node is an instance of some template, containing a set of polygons (rectangles).

Here we define only a one–layer routing as a layout (see Figure 1).

Definition 2: Given a layout $L$, the layout solution $\sigma$ of $L$ is a function $\sigma : V \rightarrow \mathbb{R}$ where $V$ is a set of polygons (rectangles) and instances' borders in $L$ and $\sigma(p)$ is the coordinate of $p$.

The coordinate of a polygon is the coordinate of its center. The coordinate (offset) of an instance is the coordinate of its left border, as shown on Figure 17.
Note that the layout solution $\sigma$ of a layout $L$ is **feasible** if it satisfies all spacing rules, i.e. for any 2 elements $u, v \in V$ we have

$$\sigma(v) - \sigma(u) \geq \text{spacing_rule}(L, u, v).$$

**Definition 3**: A feasible layout solution $\sigma$ of a layout $L$ **preserves the hierarchy structure** if for any 2 instantiations $v_1$ and $v_2$ of the same element $v$ the following holds:

$$\sigma(v_1) - \sigma(v_2) = \sigma(\text{instance}(v_1)) - \sigma(\text{instance}(v_2)).$$

We have already presented the **Flat Visibility Layout Graph** (FVLG) in the previous section. Below we give a formal definition for it.

**Definition 4**: Given a layout $L$, a graph $G(V^G, E^G)$ is a **Flat Visibility Layout Graph** if it satisfies:

1. $V^G = \{\text{polygons and instance borders in } L\}$
2. $u, v$ are polygons that are visible to each other in the layout $\Rightarrow (u, v) \in E^G$; $W(u, v)$ is defined by the layout rules.
3. $u$ is a left border of instance $i$ and $v$ is a polygon of instance $i \Rightarrow (u, v) \in E^G$; $W(u, v)$ is defined by the layout rules.
4. $v$ is a right border of instance $i$ and $u$ is a polygon of instance $i \Rightarrow (u, v) \in E^G$; $W(u, v)$ is defined by the layout rules.
5. $u, v$ are the right and left borders respectively of 2 instances in the same hierarchy that are visible to each other $\Rightarrow (u, v) \in E^G$; $W(u, v)$ is defined by the layout rules.
6. $u$ is a left border of instance $i$ and $v$ is a right border of the same instance $\Rightarrow (u, v) \in E^G$; $W(u, v) = \text{width of instance } i$. 

![Figure 17: Coordinates of an instance $i$ and polygons in vertically oriented layout.](image)
(7) $u$ is a right border of instance $i$ and $v$ is a left border of the same instance $\Rightarrow (u, v) \in E^G; W(u, v) = -$width of instance $i$.

(8) $u$ is a left border of instance $i$ and $v$ is a left border of instance $j$ such that $j$ is a direct son of $i$ $\Rightarrow (u, v) \in E^G; W(u, v)$ is defined by the placement of $j$ in $i$.

(9) $u$ is a right border of instance $i$ and $v$ is a right border of instance $j$ such that $j$ is a direct son of $i$ $\Rightarrow (u, v) \in E^G; W(u, v)$ is defined by the placement of $j$ in $i$.

Definition 5: Let $G(V^G, E^G)$ be a FVLG for a given layout $L$. The graph $H(V^H, E^H)$ is a **Hierarchy Constrained Graph** (HCG) of the layout $L$ if it satisfies:

- $V^H = V^G$
- $E^H = E^G \cup \{(u, v) | u$ and $v$ are different instantiations of the same polygon, $W(u, v) = \text{offset between the two instances that contain } u$ and $v$ respectively$\}$.

Definition 6: The arc $(u, v)$ is a **template similarity arc** of the template $t$ if $u$ and $v$ are different instantiations of the same polygon of template $t$.

Denote by $LPW(G, u, v)$ the weight of the longest path between vertices $u$ and $v$ in $G$. If there is an arc between the vertices $u$ and $v$ in $G$, the weight of the arc is denoted by $W(G, u, v)$, or simply $W(u, v)$ when the graph is clear from the context. For a path $u \rightarrow \ldots \rightarrow v$ we denote by $W(G, u \rightarrow \ldots \rightarrow v)$ the weight of the path; in this case the graph $G$ can also be omitted if it is clear from the context.
Definition 7: Given a FVLG $G(V^G, E^G)$, the graph solution $\tau$ for $G$ is a function $\tau : V \rightarrow \mathbb{R}$. A graph solution $\tau$ is feasible if for any two vertices $u, v$ we have:

$$LPW(G, u, v) \leq \tau(v) - \tau(u).$$

Since we identify between the layout elements (polygons and cell borders) and graph vertices, we can also identify between the layout solution and the graph solution.

Propositions 1–3 below capture the equality between the properties of layouts and the associated graphs, as summarized in Table 1.

**Proposition 1**: A layout $L$ has a feasible solution $\iff$ its FVLG $G$ has a feasible solution. Moreover, $\sigma$ is a feasible solution for $L \iff \sigma$ is a feasible solution for $G$.

**Proof**:

$\Rightarrow$: Let $\sigma$ be a feasible solution for the layout $L$. Let $G$ be FVLG for $L$. We will show that $\sigma$ is also a feasible solution for $G$.

Let $u, v$ be 2 vertices in $G$. It is required to show that $LPW(G, u, v) \leq \sigma(v) - \sigma(u)$. We will show this by induction on the number of vertices $N$ in the path $u \rightarrow \ldots \rightarrow v$.

If $N=1$, there is an arc $u \rightarrow v$ in $G$. It was created by rules 2…8 of Definition 4 and the weight defined by the polygon’s width, its placement or the layout rule. Since the layout
is legal, all layout rules, width and placement constraints hold and thus \( \sigma(v) - \sigma(u) \geq W(u,v) \); this means that for any path of length 1 (single arc) we have: \( \text{LPW}(G, u, v) \leq \sigma(v) - \sigma(u) \).

Now we assume the correctness for \( N \) and prove for \( N+1 \). Let \( u \rightarrow \ldots \rightarrow w \rightarrow v \) be a path from \( u \) to \( v \) of length of \( N+1 \); the induction hypothesis implies: \( \text{LPW}(G, u, w) \leq \sigma(w) - \sigma(u) \) and \( \text{LPW}(G, w, v) \leq \sigma(v) - \sigma(w) \). For every longest path in \( G \) of length \( \leq N+1 \) the induction hypothesis implies that \( \text{LPW}(G, u, v) \leq \sigma(v) - \sigma(u) \); thus \( \sigma \) is a feasible graph solution for \( G \).

\( \Leftarrow \): Let \( \sigma \) be a feasible solution for \( G \). Clearly \( \text{LPW}(G, u, v) \leq \sigma(v) - \sigma(u) \Rightarrow W(u,v) \leq \sigma(v) - \sigma(u) \); thus all the layout rules hold by construction of \( G \). Hence \( \sigma \) is a feasible layout solution for \( L \).

Q.E.D.

**Proposition 2:** A layout \( L \) has a feasible solution that preserves the hierarchy structure \( \iff \) its HCG has a feasible solution. Moreover, \( \sigma \) is a feasible solution for \( L \) that preserves the hierarchy structure \( \iff \sigma \) is a feasible solution for \( H \).

**Proof:**

\( \Rightarrow \): Let \( \sigma \) be a feasible solution that preserves the hierarchy structure for the layout \( L \). Let \( H \) be a hierarchy constrained graph for \( L \). We will show that \( \sigma \) is also a feasible solution for \( G \).

The only change from Proposition 1 is that we need to show the basis of the induction: for any arc \( (u,v) \) between two instantiations of the same element \( W(u,v) \leq \sigma(v) - \sigma(u) \). By the construction of \( H \): \( W(u,v)=\) offset between the two instances that contain \( u \) and \( v \), respectively. \( \sigma \) preserves the hierarchy structure \( \Rightarrow \sigma(v) - \sigma(u)= \) offset between the 2 instances that contain \( u \) and \( v \), respectively; thus \( W(u,v) = \sigma(v) - \sigma(u) \);

\( \Leftarrow \): Let \( \sigma \) be a feasible solution for the HCG of the layout \( L \). From Proposition 1 it follows that \( \sigma \) is also a feasible solution for \( L \). We will show that \( \sigma \) also preserves the hierarchy structure. We need to show that for any 2 instantiations \( u, v \) of the same
element the following holds: \( \sigma(v) - \sigma(u) = \sigma(\text{instance}(v)) - \sigma(\text{instance}(u)) \). By the construction of \( H \) the following holds:

1. \( \sigma(\text{instance}(v)) - \sigma(\text{instance}(u)) = W(u, v) \leq \sigma(v) - \sigma(u) \)
2. \( \sigma(\text{instance}(u)) - \sigma(\text{instance}(v)) = W(v, u) \leq \sigma(u) - \sigma(v) \) \( \Rightarrow \)
   \( \sigma(\text{instance}(v)) - \sigma(\text{instance}(u)) \geq \sigma(v) - \sigma(u) \)

Thus from (1) and (2), \( \sigma(\text{instance}(v)) - \sigma(\text{instance}(u)) = \sigma(v) - \sigma(u) \).

Q.E.D.

**Proposition 3**: Given a HCG \( H \) of the layout \( L \), a feasible solution exists for \( H \iff \) no positive cycles exist in \( H \).

**Proof**:  
\( \Rightarrow \): Let \( \sigma \) be a feasible solution for \( H \). Proof by contradiction: assume that \( u \rightarrow \ldots \rightarrow v \rightarrow u \) is a positive cycle. Since \( \sigma \) is feasible, \( \sigma(v) - \sigma(u) \geq \text{LPW}(H, u, v) \); also \( \sigma(u) - \sigma(v) \geq \text{LPW}(H, v, u) \); Thus the total length of the given positive cycle is \( \text{LPW}(H, u, v) + \text{LPW}(H, v, u) \leq \sigma(v) - \sigma(u) + \sigma(u) - \sigma(v) = 0 \); Contradiction.

\( \Leftarrow \): For every \( v \in V \) define \( \sigma(v) = \text{LPW}(H, \text{top_left_border}, v) \). Let \( u, v \in H \).

We need to show that \( \text{LPW}(H, u, v) \leq \sigma(v) - \sigma(u) \).

\( \text{LPW}(H, \text{top_left_border}, v) \geq \text{LPW}(H, \text{top_left_border}, u) + \text{LPW}(H, u, v) \)

\( \sigma(v) \geq \sigma(u) + \text{LPW}(H, u, v) \).

\( \text{LPW}(H, u, v) \leq \sigma(v) - \sigma(u) \).

Q.E.D.

**Table 1: Equivalence between the properties of layouts and graphs**

<table>
<thead>
<tr>
<th>Layout</th>
<th>Legal</th>
<th>Legal and preserves hierarchy structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph</td>
<td>No positive cycles in FVLG</td>
<td>No positive cycles in HCG</td>
</tr>
</tbody>
</table>
4.2 The Algorithm's Invariants

We continue with a presentation of the main invariants of the Merge and Reduce phases. Once the hierarchical order for the templates is defined, \(w.l.o.g.\) we can rename the templates of the layout to \(1,...,N\) starting from the leaves to the top level template.

4.2.1 Merge

As presented in the top level flow in Figure 5, the procedure \(\text{mergeTemplateInstances}\) is iteratively applied to the FVLG.

Definition 8: An \(i\)-Merged Layout Graph is a FVLG after exactly \(i\) calls to the procedure \(\text{mergeTemplateInstances}\). The \(N\)-Merged Layout Graph is called the Merged Layout Graph.

Definition 9: Let \(G(V^G, E^G)\) be a FVLG for a given layout \(L\), and \(t\) some template in \(L\). The graph \(H=(V^H, E^H)\) is a \(t\)-Hierarchy Constrained Graph of the layout \(L\) if it satisfies:

- \(V^H = V^G\)
- \(E^H = E^G \cup \{(u, v)\mid (u, v) \text{ is a template similarity arc of } t\}\)

Definition 10: Let \(G(V^G, E^G)\) be a FVLG for a given layout \(L\). The graph \(H=(V^H, E^H)\) is an \(i\)-Hierarchy Constrained Graph \(0 \leq i \leq N\) of the layout \(L\) if it satisfies:

- \(V^H = V^G\)
- \(E^H = E^G \cup \{(u, v)\mid \forall j \ 1 \leq j \leq i \ (u, v) \text{ is a template similarity arc of } j\}\)

Note that the 0-hierarchy constrained graph of layout \(L\) is a FVLG of \(L\); the \(N\)-hierarchy constrained graph of layout \(L\) is a HCG of \(L\) (where \(N\) is the number of templates in \(L\)).

\(\forall i\) the \(i\)-Hierarchy Constrained Graph \(H_i(V_i, E_i)\) of layout \(L\) and a the HCG \(H(V^H, E^H)\) of \(L\) satisfy: \(V_i=V^H, E_0\subseteq\ldots\subseteq E_{i-1}\subseteq E_i=E^H\).

Propositions 4 and 5 capture the main invariant of the Merge phase. They claim that at every step of Merge, the weight of the longest path between a pair of vertices in the \(i\)-Merged Layout Graph equals to that in the \(i\)-Hierarchy Constrained Graph.

We prove the relation \(\leq\) between the weights of the longest paths in Merged Layout Graph and Hierarchy Constrained Graph in Proposition 4a and relation \(\geq\) in Proposition 4b; we conclude the equality in proposition 4.
Proposition 4a: Given a layout graph $G(V_G, E_G)$, some template $t$, a $t$–hierarchy constrained graph $H_h(V_{H_h}, E_{H_h})$ of $G$, and a merged graph $M(V_M, E_M)$ generated from $G$ by the procedure $\text{mergeTemplateInstances}$ applied to template $t$. Let $u, v \in V_M$ so that $u, v \in t$. Let $i$ be an instance of $t$ and $u(i), v(i) \in V_{H_h}$ are instantiations of $u, v$, respectively, in instance $i$. The following holds: $\forall u, v \in t \forall i \ LPW(M, u, v) \leq LPW(H_h, u(i), v(i))$.

Proof:
We will pick one of the longest paths between $u$ and $v$ in $M$ and find a path in $H_t$ of the same length. Let $\pi = u \rightarrow x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_n \rightarrow v$ be the path that we picked. We will construct the path $\pi'$ in $H_t$ and show that its length is at least equal to that of $\pi$.

We will show the correctness by induction on the number of template vertices in the path, i.e. $m = |\{x_j \in \pi | 1 \leq j \leq n, x_j \in t\}|$.

Basis: $m = 0$: $\forall j x_j \notin t$.

There are 2 cases: $x_1 = v$ (the path is $u \rightarrow v$) and $x_1 \neq v$.

Case $x_1 = v$. The arc $(u(i), v(i))$ exists in $G$ and thus it exists in $H_t$. The weight of the arc, while being added to the merged graph $M$, was changed twice: once by adding the offset of instance $i$ (line 10 of $\text{mergeTemplateInstances}$) since the arc is an output of $u(i) \in i$, and then by subtracting the offset of instance $i$ (line 7 of $\text{mergeTemplateInstances}$) since the arc is an input of $v(i) \in i$; thus the weight of the arc in $G$ is the same and the weight of arc $(u(i), v(i))$ in $H_t$ is at least the same.

Case $x_1 \neq v$. By the construction of $M$, there exists an instance $i'$ so that arc $(u(i'), x_1) \in E_G$, and the weight of the arc changed exactly once by subtracting the offset of instance $i$ (line 7 of $\text{mergeTemplateInstances}$) since the arc is an output of $u(i') \in i'$: $W(u, x_1) = W(u(i'), x_1) + \text{offset}(i')$.

By the construction of $M$, there exists an instance $i''$ so that arc $(x_n, v(i'')) \in E_G$, and the weight of the arc changed exactly once by adding the offset of instance $i''$ (line 10 of $\text{mergeTemplateInstances}$) since the arc is an input of $v(i'') \in i''$: $W(x_n, v) = W(x_n, v(i'')) - \text{offset}(i'')$.

Thus $\forall j x_j \notin t$ the path $x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_n$ exists in $G$, and all the arcs in the path are unchanged.

Note that $H_t$ contains the template similarity arcs of $t$. 
Hence \((u(i), u(i')) \in H_i\) and \((v(i''), v(i)) \in H_i;\)
\[
W(u(i), u(i')) = (\text{offset}(i') - \text{offset}(i)), \quad W(v(i''), v(i)) = (\text{offset}(i) - \text{offset}(i'')).
\]
In case \(i' = i\), \(W(u(i), u(i)) = (\text{offset}(i) - \text{offset}(i')) = 0\), the same holds if \(i'' = i\).

![Figure 20: The path in \(H_i\). Similarity arcs are in red.](image)

Blue represents the path \(x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_n\)

Let us calculate the weight of path \(u(i) \rightarrow u(i') \rightarrow x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_n \rightarrow v(i'') \rightarrow v(i)\) in \(H_i;\)
\[
W(u(i), u(i')) + W(u(i'), x_i) + \text{LPW}(x_i \rightarrow x_2 \rightarrow \ldots \rightarrow x_n) + W(x_n, v(i'')) + W(v(i''), v(i))
\]
\[
= (\text{offset}(i') - \text{offset}(i)) + (W(u, x_i) - \text{offset}(i')) + \text{LPW}(x_i \rightarrow x_2 \rightarrow \ldots \rightarrow x_n) + (W(x_n, v) + \text{offset}(i'')) + (\text{offset}(i) - \text{offset}(i'')) = W(u, x_i) + \text{LPW}(x_i \rightarrow x_2 \rightarrow \ldots \rightarrow x_n) + W(x_n, v).
\]
This is exactly the weight of the longest path in \(M\).

Until now we proved the basis of the induction, we continue with the step.

We assume the correctness for every path \(u \rightarrow x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_n \rightarrow v\) that satisfies
\[
|\{x_j | 1 \leq j \leq n, x_j \in t\}| < m,
\]
and prove for \(m \geq 1\).

Let \(\pi = u \rightarrow x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_n \rightarrow v\) be the longest path in \(M\) so that \(|\{x_j \in t | 1 \leq j \leq n, x_j \in t\}| = m\).

Since \(m > 0\), there exists \(w \in t\) so that the path is \(\pi = u \rightarrow x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow w \rightarrow \ldots \rightarrow x_n \rightarrow v\).

Clearly the path \(\pi_1 = u \rightarrow x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow w\) is the longest path in \(M\) between \(u\) and \(w\) (otherwise we can find a longer path in \(M\) between \(u\) and \(v\)). The Same holds for \(\pi_2 = w \rightarrow \ldots \rightarrow x_n \rightarrow v\).

Thus the induction hypothesis holds for both paths \(\pi_1\) and \(\pi_2\) meaning
\[
\text{LPW}(M, u, w) \leq \text{LPW}(H_i, u(i), w(i))
\]
\[
\text{LPW}(M, w, v) \leq \text{LPW}(H_i, w(i), v(i))
\]
Thus \( \text{LPW}(M, u, v) = \text{LPW}(H_t, u(i), w(i)) + \text{LPW}(M, w, v) \leq \text{LPW}(H_t, u(i), v(i)) \)

Q.E.D.

**Proposition 4b**: Given are a layout graph \( G(V^G, E^G) \), some template \( t \), a \( t \)-hierarchy constrained graph \( H_t(V^H, E^H) \) of \( G \), and a merged graph \( M(V^M, E^M) \) generated from \( G \) by the procedure `mergeTemplateInstances` applied to template \( t \). Let \( u, v \in V^M \) so that \( u, v \in t \). Let \( i \) be an instance of \( t \) and \( u(i), v(i) \in V^H \) are instantiations of \( u, v \), respectively, in instance \( i \). The following holds:

\[ \forall u, v \in t \forall i \quad \text{LPW}(M, u, v) \geq \text{LPW}(H_t, u(i), v(i)). \]

**Proof**:

We will pick one of the longest paths between \( u(i) \) and \( v(i) \) in \( H_t \) and find a path in \( M \) of at least the same length. Let \( u(i) \rightarrow x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_n \rightarrow v(i) \) be the longest path in \( H_t \).

We shall show the claim by induction on the number of template vertices included in the longest path, \( m = |\{x_j| 1 \leq j \leq n, x_j \in t\}| \)

**Basis**: \( m = 0 \).

There are 2 cases: \( x_j = v \) (the path is \( u(i) \rightarrow v(i) \)) and \( x_j \neq v \).

**Case** \( x_j = v \). By the construction of \( M \) the arc \( (u, v) \) exists in \( G \) and thus it exists in \( M \). The weight of the arc, while being added to the merged graph was changed twice: once by adding the offset of instance \( i \) (line 10 of `mergeTemplateInstances`) since the arc is an output of \( u(i) \in i \) and then by subtracting the offset of instance \( i \) (line 7 of `mergeTemplateInstances`) since the arc is an input of \( v(i) \in i \). Thus the weight of the arc in \( M \) is at least the same as in \( G \). Since \( (u(i), v(i)) \) is not a template similarity arc, it is same in \( G \) and \( H_t \).

**Case** \( x_j \neq v \). By the construction of \( M \), the weight of arc \( (u(i), x_i) \in E^G(\text{line 10 of mergeTemplateInstances}) \) since the arc is an output of \( u(i) \in i \): \( W(u, x_i) = W(u(i), x_i) + \text{offset}(i) \).

By the construction of \( M \), the weight of arc \( (x_i, v(i)) \) changed exactly once by subtracting the offset of instance \( i \) (line 7 of `mergeTemplateInstances`) since the arc is an input of \( v(i) \in i \): \( W(x_n, v) = W(x_n, v(i)) - \text{offset}(i) \).

Since \( \forall j \quad x_j \notin t \) the number of template similarity arcs is 0, the path \( u(i) \rightarrow x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_n \rightarrow v(i) \) exists in \( G \) and belongs to the same instance. By the
construction of $M$, all the paths from $u(i)$ to $v(i)$ exist in $M$. Since $\forall j \ x_j \not\in t$ the weights of arcs $(x_j, x_{j+1})$ are unchanged.

Let us calculate the weight of the path $u \rightarrow x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_n \rightarrow v$ in $M$:

$$W(M, u \rightarrow x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_n \rightarrow v) = W(M, u, x_1) + W(M, x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_n) + W(M, x_n, v) = (W(G, u(i), x_i) + \text{offset}(i)) + W(G, x_i \rightarrow x_2 \rightarrow \ldots \rightarrow x_n) + (W(G, x_n, v) - \text{offset}(i)) = W(G, u(i), x_i) + W(G, x_i \rightarrow x_2 \rightarrow \ldots \rightarrow x_n) + W(G, x_n, v(i))$$

This is exactly the weight of the path in $H_t$.

Until now the basis of the induction was proven; we continue with the step.

Assume the correctness for every path $u(i) \rightarrow x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_n \rightarrow v(i)$ so that $|\{x_j| 1 \leq j \leq n, x_j \in t\}| < m$, and prove for $m \geq 1$.

Let $\pi = u(i) \rightarrow x_i \rightarrow x_2 \rightarrow \ldots \rightarrow x_n \rightarrow v(i)$ be the longest path $\pi$ in $H_t$ so that $|\{x_j| 1 \leq j \leq n, x_j \in t\}| = m$.

Let $(x_k, x_{k+1})$ be the first template similarity arc in the path. Clearly, $x_k \in i'$ and $x_{k+1} \in i''$ are instantiations of the same vertex $w$ in instances $i'$ and $i''$, respectively, which are – in turn – instances of $t$.

Let us introduce a new path $\pi'$ in $H_t$ with at least the same weight as $\pi$.

$$u(i) \rightarrow x_1 \rightarrow \ldots \rightarrow x_k \rightarrow w(i) \rightarrow x_{k+1} \rightarrow \ldots \rightarrow x_n \rightarrow v(i)$$

$$W(H_b, x_k \rightarrow w(i) \rightarrow x_{k+1}) = W(H_b, x_k, w(i)) + W(H_b, w(i), x_{k+1}) = \text{offset}(i) - \text{offset}(i') + \text{offset}(i') - \text{offset}(i) = W(H_b, x_k, x_{k+1})$$

Now divide the path $\pi'$ into 2 paths: $u(i) \rightarrow x_1 \rightarrow \ldots \rightarrow x_k \rightarrow w(i)$ and $w(i) \rightarrow x_{k+1} \rightarrow \ldots \rightarrow x_n \rightarrow v(i)$. Both of them satisfy $|\{x_j| 1 \leq j \leq n, x_j \in t\}| < m$.

Clearly path $u \rightarrow x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow w(i)$ is a longest path in $M$ between $u$ and $w$ (otherwise we can find a longer path in $M$ between $u$ and $v$). The same holds for $w(i) \rightarrow \ldots \rightarrow x_n \rightarrow v$.

Thus induction hypothesis holds for both paths $u \rightarrow x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow w$ and $w \rightarrow \ldots \rightarrow x_n \rightarrow v$, meaning

$$\text{LPW}(M, u, w) \geq \text{LPW}(H_t, u(i), w(i))$$

$$\text{LPW}(M, w, v) \geq \text{LPW}(H_t, w(i), v(i))$$

$$\text{LPW}(M, u, v) \geq \text{LPW}(M, u, w) + \text{LPW}(M, w, v) \geq \text{LPW}(H_t, u(i), v(i))$$

Q.E.D.
**Proposition 4**: Given are a layout graph $G(V^G, E^G)$, some template $t$, a $t$–hierarchy constrained graph $H(t^{Ht}, E^{Ht})$ of $G$, and a merged graph $M(V^M, E^M)$ generated from $G$ by the procedure $\text{mergeTemplateInstances}$ applied to template $t$. Let $u, v \in V^M$ so that $u, v \in t$. Let $i$ be an instance of $t$ and $u(i), v(i) \in V^{Ht}$ are instantiations of $u, v$, respectively, in instance $i$. The following holds: $\forall u, v \in t \forall i \ LPW(M, u, v) = LPW(H_t, u(i), v(i))$.

**Proof**: Follows directly from Propositions 4a and 4b.

Proposition 4 claims the equality between the weights of the longest paths in *Merged Layout Graph* and *Hierarchy Constrained Graph* between a pair of vertices $u, v \in V^M$ so that $u, v \in t$ for a template $t$ treated by procedure $\text{mergeTemplateInstances}$. In Proposition 5 we prove the same property for a pair of vertices $u, v \in V^M$ so that $u, v \notin t$ for a template $t$ treated by procedure $\text{mergeTemplateInstances}$.

We prove the relation $\leq$ between the weights of the longest paths in *Merged Layout Graph* and *Hierarchy Constrained Graph* in Proposition 5a and relation $\geq$ in Proposition 5b; we conclude the equality in proposition 5.

**Proposition 5a**: Given are a layout graph $G(V^G, E^G)$, some template $t$, a $t$–hierarchy constrained graph $H(t^{Ht}, E^{Ht})$ of $G$, and a merged graph $M(V^M, E^M)$ generated from $G$ by procedure $\text{mergeTemplateInstances}$ applied to template $t$. Let $u, v \in V^M$ so that $u, v \notin t$. The following holds: $\forall u, v \notin t \ LPW(M, u, v) \leq LPW(H_t, u, v)$.

**Proof**: Let $u \rightarrow x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_k \rightarrow v$ be the longest path in $M$ between $u$ and $v$.

There are 2 cases:

1. $\forall j \ 1 \leq j \leq k \ x_j \notin t \Rightarrow$ the same path exists in $G$ and all the arcs in the path are unchanged (see $\text{mergeTemplateInstances}$).

2. $\exists j \ 1 \leq j \leq k \ x_j \in t$. Denote the first vertex in the path that belong to $t$ by $a$ and the last vertex in the path that belong to $t$ by $b$. So the path is

$u \rightarrow x_1 \rightarrow \ldots \rightarrow x_n \rightarrow a \rightarrow \ldots \rightarrow b \rightarrow y_1 \rightarrow \ldots \rightarrow y_m \rightarrow v, \ \forall j \ x_j, y_j \notin t$

By the construction of $M$, there exists an instance $i'$ so that arc $(x_n, a(i')) \in E^G$, and the weight of the arc changed exactly once by subtracting the offset of instance $i'$ (line 7 of
mergeTemplateInstances) since the arc is an input of \( v(i') \in i' \): \( W(x_n, a) = W(x_n, a(i')) - \text{offset}(i') \).

By the construction of \( M \), there exists an instance \( i'' \) so that arc \((b(i'), y_1) \in E^G \), and the weight of the arc changed exactly once by adding the offset of instance \( i'' \) (line 10 of mergeTemplateInstances) since the arc is an output of \( b(i'') \in i'' \): \( W(b, y_1) = W(b(i''), y_1) + \text{offset}(i'') \).

Using Proposition 4a there exists a path in \( H \), \( a(i') \Rightarrow \ldots \Rightarrow b(i') \) so that \( \text{LPW}(M, a, b) \leq \text{LPW}(H, a(i'), b(i')) \).

In addition, there is an arc \((b(i'), b(i'')) \in H \): \( W(b(i'), b(i'')) = (\text{offset}(i'') - \text{offset}(i')) \).

Now let us calculate the weight of the path

\[
\begin{align*}
\text{LPW}(H \cup u \Rightarrow x_1 \Rightarrow \ldots \Rightarrow x_n \Rightarrow a(i') \Rightarrow \ldots \Rightarrow b(i') \Rightarrow b(i'') \Rightarrow y_1 \Rightarrow \ldots \Rightarrow y_m \Rightarrow v) \text{ in } H \text{ :}
\end{align*}
\]

\[
\text{LPW}(H, u, v) \geq \text{LPW}(H, u, x_n) + W(H, x_n, a(i')) + \text{LPW}(H, a(i'), b(i')) + W(H, b(i''), y_1) + \text{LPW}(H, y_m, v) \geq
\]

\[
\text{LPW}(M, u, x_n) + W(M, x_n, a) + \text{offset}(i') + \text{LPW}(M, a, b) + (\text{offset}(i'') - \text{offset}(i')) + W(M, u, y_1) - \text{offset}(i'') + \text{LPW}(M, y_m, v) = \text{LPW}(M, u, v).
\]

Q.E.D.

**Proposition 5b**: Given are a layout graph \( G(V^G, E^G) \), some template \( t \), a \( t \)-hierarchy constrained graph \( H_t(V^{H_t}, E^{H_t}) \) of \( G \), and a merged graph \( M(V^M, E^M) \) generated from \( G \)
by procedure $\text{mergeTemplateInstances}$ applied to template $t$. Let $u, v \in t^M$ so that $u, v \notin t$.

The following holds: $\forall u, v \notin t \ \text{LPW}(M, u, v) \geq \text{LPW}(H_t, u, v)$.

**Proof:** Let $\pi = u \rightarrow x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_k \rightarrow v$ be the longest path in $H_t$ between $u$ and $v$.

There are 2 cases:

1. $\forall j \ 1 \leq j \leq k \ \ x_j \notin t$ $\Rightarrow$ the same path exists in $G$ and all the arcs in the path are unchanged (see $\text{mergeTemplateInstances}$).

2. $\exists j \ 1 \leq j \leq k \ x_j \in t$. Denote the first vertex in the path that belongs to $t$ by $a(i')$ and the last vertex in the path that belongs to $t$ by $b(i'')$; $a(i')$ and $b(i'')$ belong to the instances of $t$ $i'$ and $i''$ respectively. So the path is

$$\pi = u \rightarrow x_1 \rightarrow \ldots \rightarrow x_n \rightarrow a(i') \rightarrow \ldots \rightarrow b(i'') \rightarrow y_1 \rightarrow \ldots \rightarrow y_m \rightarrow v, \ x_j, y_j \notin t$$

Since there are arcs $b(i'') \rightarrow b(i')$ and $b(i') \rightarrow b(i'')$ in $H_t$, let us introduce a new path $\pi'$ in $H_t$ with the same weight as $\pi$:

$$\pi' = u \rightarrow x_1 \rightarrow \ldots \rightarrow x_n \rightarrow a(i') \rightarrow \ldots \rightarrow b(i'') \rightarrow b(i') \rightarrow b(i'') \rightarrow y_1 \rightarrow \ldots \rightarrow y_m \rightarrow v, \ x_j, y_j \notin t$$

$W(b(i''), b(i')) + W(b(i'), b(i'')) = \text{offset}(i') - \text{offset}(i'') + \text{offset}(i'') - \text{offset}(i') = 0$

By the construction of $M$, there exists an arc $(x_n, a) \in E^M$, and the weight of the arc changed exactly once by subtracting the offset of instance $i'$ (line 7 of $\text{mergeTemplateInstances}$) since the arc is an input of $v(i') \in i'$: $W(x_n, a) = W(x_n, a(i')) - \text{offset}(i')$.  

---

**Figure 22:** The path $\pi$ in $H_t$

**Blue represents the paths** $x_1 \rightarrow \ldots \rightarrow x_n$ and $y_1 \rightarrow \ldots \rightarrow y_m$
By the construction of $M$, there exists an instance $i''$ so that arc $(b, y_1) \in E^T$, and the weight of the arc changed exactly once by adding the offset of instance $i''$ (line 10 of \texttt{mergeTemplateInstances}) since the arc is an output of $b(i'') \in i''$. $W(b, y_1) = W(b(i''), y_1) + \text{offset}(i'')$.

Using Proposition 4b there exists a path in $H_t$ $a \rightarrow \ldots \rightarrow b$ so that $\text{LPW}(M, a, b) \geq \text{LPW}(H_t, a(i'), b(i'))$.

Let us calculate the weight of the path in $M$:

$$\pi' = u \rightarrow x_1 \rightarrow \ldots \rightarrow x_n \rightarrow a(i') \rightarrow \ldots \rightarrow b(i'') \rightarrow b(i') \rightarrow y_1 \rightarrow \ldots \rightarrow y_m \rightarrow v :$$

$$\text{LPW}(H_t, u, v) = \text{LPW}(H_t, u, x_n) + W(H_t, x_n, a(i')) + \text{LPW}(H_t, a(i'), b(i')) + W(H_t, b(i'), b(i'')) + W(H_t, b(i''), y_1) + \text{LPW}(H_t, y_1, v) \leq$$

$$\text{LPW}(M, u, x_n) + W(M, x_n, a) + \text{offset}(i') + \text{LPW}(M, a, b) + \text{offset}(i'') - \text{offset}(i') + W(M, b, y_1) - \text{offset}(i'') + \text{LPW}(M, y_1, v) \leq \text{LPW}(M, u, v).$$

Q.E.D.

**Proposition 5**: Given are a layout graph $G(V^G, E^G)$, some template $t$, a $t$–hierarchy constrained graph $H_t(V^{Ht}, E^{Ht})$ of $G$, and a merged graph $M(V^M, E^M)$ generated from $G$ by procedure \texttt{mergeTemplateInstances} applied to template $t$. Let $u, v \in M$ so that $u, v \notin t$. The following holds: $\forall u, v \notin t \quad \text{LPW}(M, u, v) = \text{LPW}(H_t, u, v)$.

**Proof**: Follows directly from Propositions 5a and 5b.

Proposition 6 follows from Propositions 4 and 5 by induction on the number of iterations.

**Proposition 6**: Given are a HCG $H$ of the layout $L$ and a merged graph $M$ generated by the Merge phase. Let $u, v \in M$ so that $u, v \in t$ for some template $t$. Let $i$ be an instance of $t$ and $u(i), v(i) \in H$ are instantiations of $u$ and $v$, respectively, in instance $i$. The following holds: $\forall u, v \in t \quad \forall i \quad \text{LPW}(M, u, v) = \text{LPW}(H, u(i), v(i))$.

**Proof**: We will prove by induction on number $n$ of calls to \texttt{mergeTemplateInstances} as presented in Figure 5.

Basis: for $n=0$, $M=H$ and we are done.

Induction hypothesis: Let $M'$ be the $n$-merged graph after $n$ iterations of loop 3-5.
IH1. For any template \( t \), if \( t \) has already been treated by \textit{mergeTemplateInstances}, then 
\[ \forall u, v \in t \forall i \text{ LPW}(M^n, u, v) = \text{LPW}(H, u(i), v(i)). \]

Where \( i \) is an instance of template \( t \) and \( u(i), v(i) \in H \) are instantiations of \( u \) and \( v \), respectively, in instance \( i \).

IH2: For any template \( t \), if \( t \) was not treated by \textit{mergeTemplateInstances}, then 
\[ \forall u, v \in t \forall i \text{ LPW}(M^n, u(i), v(i)) = \text{LPW}(H, u(i), v(i)). \]

Step: Let \( M^{n+1} \) be the \((n+1)\)-\textit{Merged Graph}.

1. There are 2 cases:
   a. Let \( t \) be some template that was already treated by \textit{mergeTemplateInstances}, before iteration \( n+1 \) and \( u, v \in t \).

   We will show that \( \forall i \text{ LPW}(M^{n+1}, u, v) = \text{LPW}(H, u(i), v(i)). \)

   Since \( t \) was already treated before the iteration started, Proposition 5 implies 
   \[ \forall i \text{ LPW}(M^n, u, v) = \text{LPW}(M^n, u, v). \]

   By IH1, \( \forall i \text{ LPW}(M^n, u, v) = \text{LPW}(H, u(i), v(i)). \)

   Thus \( \forall i \text{ LPW}(M^{n+1}, u, v) = \text{LPW}(H, u(i), v(i)). \)

   b. Let \( t \) be some template that is treated by \textit{mergeTemplateInstances} in iteration \( n+1 \) and \( u, v \in t \).

   We shall show that \( \forall i \text{ LPW}(M^{n+1}, u, v) = \text{LPW}(H, u(i), v(i)). \)

   Since \( t \) was treated in iteration \( n+1 \), By Proposition 4 \( \forall i \text{ LPW}(M^{n+1}, u, v) = \text{LPW}(M^n, u(i), v(i)). \)

   By IH2, \( \forall i \text{ LPW}(M^n, u(i), v(i)) = \text{LPW}(H, u(i), v(i)). \)

   Thus \( \forall i \text{ LPW}(M^{n+1}, u, v) = \text{LPW}(H, u(i), v(i)). \)

2. Let \( t \) be some template that was not treated by \textit{mergeTemplateInstances}, before/in iteration \( n+1 \) and \( u, v \in t \).

   Will show \( \forall u, v \in t \forall i \text{ LPW}(M^n, u(i), v(i)) = \text{LPW}(H, u(i), v(i)). \)

   By Proposition 5 \( \forall u, v \) that are not in a treated template holds: \( \text{LPW}(M^{n+1}, u, v) = \text{LPW}(M^n, u, v), \)
Thus $\text{LPW}(M^{n+1}, u(i), v(i)) = \text{LPW}(M^n, u(i), v(i))$.

By IH2, $\forall i \, \text{LPW}(M^n, u(i), v(i)) = \text{LPW}(H, u(i), v(i))$.

Hence $\forall i \, \text{LPW}(M^{n+1}, u, v) = \text{LPW}(H, u(i), v(i))$.

Q.E.D.

**Corollary 1**: A layout $L$ has a feasible solution that preserves a hierarchy structure $\iff$ its merged graph $M$ does not contain positive cycles.

Proof:

(1) By Proposition 2 $L$ has a feasible solution that preserves a hierarchy structure $\iff$ its HCG $H$ has feasible solution.

(2) By Proposition 3 feasible solution exists for HCG $H$ of the layout $L$ $\iff$ no positive cycles exists in $H$.

(3) By Proposition 6 $\forall u, v \in t \forall i \, \text{LPW}(M, u, v) = \text{LPW}(H, u(i), v(i))$; thus positive cycles exist in $H$ $\iff$ positive cycles exist in $M$.

Thus using (1), (2), and (3), the layout $L$ has feasible solution that preserves hierarchy structure $\iff$ its merged graph $M$ does not contain positive cycles.

Q.E.D.

Corollary 1 implies that returning *false* on the presence of positive cycles in the merged graph is safe.

**4.2.2 Reduce**

As presented by the top level flow in Figure 5, the procedure $\text{reduceTemplateInstances}$ is iteratively applied to merged graphs in the template hierarchical order.

**Definition 11**: An *$i$-Reduced Layout Graph* is a graph returned by $\text{reduceTemplateInstances}$ after exactly $i$ calls to it.

Proposition 7 captures the invariant of the Reduce phase: the longest path weight between a pair of vertices in the graph.
**Proposition 7:** Given a layout graph $G(V^G, E^G)$ and a reduced layout graph $R(V^R, E^R)$ generated by a single invocation of procedure reduceTemplateInstances on $G$, the following holds: $\forall u, v \in V^R \ LPW(R, u, v) = LPW(G, u, v)$.

**Proof:** By induction on the number of iterations of loop 2-8. Denote by $R^i$ the reduced graph after the $i$-th iteration of the loop. Note, that at every loop iteration exactly one vertex (with all it incident arcs) is removed and some other arcs are added to the graph.

Basis: after 0 iterations, $R^0$ equals to $G$ since nothing was changed.

Induction hypothesis: For any 2 vertices $u, v \in R^{i-1}$, the following holds: $LPW(R^{i-1}, u, v) = LPW(G, u, v)$. We shall show the correctness for $R^i$.

Let $r$ be the vertex that is going to be reduced at the step $i$.

First we will show that $LPW(R^i, u, v) \geq LPW(G, u, v)$

Let $u = x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_k = v$ be the longest path between $u$ and $v$ in $R^{i-1}$.

If $\forall j \ x_j \neq r$ all the vertices and arcs in the path remain exactly as in $R^{i-1}$. Thus $LPW(R^i, u, v) \geq LPW(R^{i-1}, u, v) = LPW(G, u, v)$.

Otherwise $\exists j \ x_j = r$. So the path is $u = x_1 \rightarrow \ldots \rightarrow x_n \rightarrow r \rightarrow y_1 \rightarrow \ldots \rightarrow y_m = v$, $x_j, y_j \neq r$. (We assume that there are no loops in the path).

As in line 5 of reduceTemplateInstances for the pair of arcs $(x_n, r)$ and $(r, y_1)$ the new arc $(x_n, y_1)$ is added to the graph:

$W(R^i, x_n, y_1) = W(R^{i-1}, x_n, r) + W(R^{i-1}, r, y_1)$.

Thus there is a path in $R^i$ $u = x_1 \rightarrow \ldots \rightarrow x_n \rightarrow y_1 \rightarrow \ldots \rightarrow y_m = v$, and

$LPW(R^i, u, v) \geq LPW(R^i, u, x_n) + W(R^i, x_n, y_1) + LPW(R^i, y_1, y_m) = LPW(R^{i-1}, u, x_n) + W(R^{i-1}, x_n, r) + W(R^{i-1}, r, y_1) + LPW(R^{i-1}, y_1, y_m) = LPW(R^{i-1}, u, v) = LPW(G, u, v)$.

It is left to show that $LPW(R^i, u, v) \leq LPW(G, u, v)$

Let $u = x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_k = v$ be the longest path between $u$ and $v$ in $R^i$.

Since there are no loops in the path, there is at most one arc that does not exist in $R^{i-1}$.

If all the arcs in the path are unchanged (i.e. the same path exists in $G$), we are done.
Otherwise, the path is $x_1 \rightarrow \ldots \rightarrow x_n \rightarrow y_1 \rightarrow \ldots \rightarrow y_m$. (Here we assume that there are no loops in the path).

As in line 5 of \texttt{reduceTemplateInstances} the arc $(x_n, y_1)$ is added to the graph when a pair of arcs $(x_n, r)$ and $(r, y_1)$ is removed so that:
\[
W(R^i, x_n, y_1) = W(R^{i-1}, x_n, r) + W(R^{i-1}, r, y_1);
\]

there exists a path in $R^{i-1}$: $u=x_1 \rightarrow \ldots \rightarrow x_n \rightarrow r \rightarrow y_1 \rightarrow \ldots \rightarrow y_m=v$; and
\[
LPW(G, u, v) = LPW(R^{i-1}, u, v) \geq LPW(R^{i-1}, u, x_n) + W(R^{i-1}, x_n, r) + W(R^{i-1}, r, y_1) + LPW(R^{i-1}, y_1, y_m) = LPW(R^i, u, v).
\]

Q.E.D

Proposition 8 follows directly from Proposition 7 by induction on the number of iterations of \texttt{reduceTemplateInstances}.

\textbf{Proposition 8}: Given a layout $L$, a merged graph $M(V^M, E^M)$ of $L$, and an $i$-reduced graph $R_i(V^R, E^R)$, we have $\forall u, v \in V^R_i$ $LPW(R_i, u, v) = LPW(M, u, v)$.

\textbf{Proof}: By induction on $i$.

Basis: The 0-Reduced Layout Graph $R_0$ is exactly the merged graph $M$.

Induction hypothesis: For any two vertices $u, v \in R_{i-1}$ the following holds: $LPW(R_{i-1}, u, v) = LPW(M, u, v)$. We will show the correctness for $R_i$.

Step: By Proposition 7, $LPW(R_{i-1}, u, v) = LPW(R_i, u, v)$, and using the induction hypothesis $LPW(R_i, u, v) = LPW(M, u, v)$.

Q.E.D

\textbf{Corollary 2}: Given a layout $L$, a merged graph $M$ of the layout $L$, and a reduced graph $R$, $\forall u, v \in V^R$ $LPW(R, u, v) = LPW(M, u, v)$.

\textbf{Proof}: immediate from Proposition 8, since a reduced graph is an $N$-reduced graph by its definition.

\subsection*{4.2.3 Exact Solution Construction}

We continue with the derivation of the exact solution phase. This phase proceeds in iterations, a single template at a time.
Definition 12: An \textit{i–Partial Solution} for a flat visibility layout graph \( G \) is a function \( \sigma_i: V^i \rightarrow \mathbb{R} \) so that \( V^i \) is the set of all the vertices that have an exact solution exactly as after the \( i^{th} \) iterations of the top–down flow. These are the vertices that represent polygons of the templates \( I_1, \ldots, I_i \) in the order defined by the template’s hierarchical order.

Note that the phase starts with \( \sigma_0 \) for \( V_0 = \{ \text{all the borders of all the instances} \} \) and it finishes with an \( N \)–partial solution which is a \textit{layout solution}. Proposition 9 captures a single step of the exact solution derivation. It is shown later that the solution always exists and it is obtained from the solution derivation phase. We will start by proving a generic property of directed graphs in Proposition 9a, which is then used in Proposition 9.

\textbf{Proposition 9a:} Given a directed weighted graph \( G=(V, E) \) with weights \( w: E \rightarrow \mathbb{R}^+ \), and non-negative function \( f: V \rightarrow \mathbb{R}^+ \), let \( \pi=v_1 \rightarrow \ldots \rightarrow v_k \) be a path in \( G \), and \( w(\pi)=\sum w(v_i, v_{i+1}) \) be a weight of the path.

Then \( w(\pi) \leq f(v_k) - f(v_1) \) iff \( \forall (u,v) \in E \Rightarrow w(u, v) \leq f(v) - f(u) \)

\textbf{Proof:}

\( \Rightarrow \) Obvious

\( \Leftarrow \) By induction on the length of the path \( \pi \).

Basis for \( \pi=u \rightarrow v \): clearly \( (u,v) \in E \), thus \( w(\pi)=w(u, v) \leq f(v) - f(u) \).

Induction hypothesis: For any \( \pi=v_1 \rightarrow \ldots \rightarrow v_k \) of length \( k \leq n \), assume \( w(\pi) \leq f(v_k) - f(v_1) \).

Step: Let \( k=n+1 \). Define \( \pi'=v_1 \rightarrow \ldots \rightarrow v_{k-1} \) and \( \pi''=v_{k-1} \rightarrow \ldots \rightarrow v_k \).

By the induction hypothesis \( w(\pi') \leq f(v_{k-1}) - f(v_1) \) and \( w(\pi'') \leq f(v_k) - f(v_{k-1}) \).

\( w(\pi) = w(\pi')+w(\pi'' \leq f(v_{k-1}) - f(v_1) + f(v_k) - f(v_{k-1}) = f(v_k) - f(v_1) \).

Q.E.D.

\textbf{Proposition 9:} If an \( i \)–partial solution for a merged graph \( M \) of layout \( L \) is obtained from the exact solution derivation phase (as presented on Figure 15) at iteration \( i \) – it is a feasible solution of the merged graph \( M \) of \( L \).

\textbf{Proof:}

Let \( \sigma_i \) be an \( i \)-partial solution for \( G \).
We will show the correctness by induction on $i$.

Basis: $i=0$, $V^0=\emptyset$, thus of course partial solution is feasible solution for $M$.

Induction hypothesis: assumption of correctness for all $i < k$;

Step: Using Proposition 9a it is enough to show that for every two vertices $u, v \in V$, so that $(u,v) \in E$, $\sigma_k(v) - \sigma_k(u) \geq W(M, u, v)$.

By the induction hypothesis $\sigma_{k-1}$ is a feasible solution of $M$. Thus $\forall u, v \in V^{k-1}$, $\sigma_k(v) - \sigma_k(u) \geq W(M, u, v)$.

w.l.o.g $v \notin V^{k-1}$.

There are 2 cases: $u \in V^{k-1}$ and $u \notin V^{k-1}$.

1. In case $u \in V^{k-1}$, $u$ has already obtained an exact solution and, as in line 7, the constraint added to the LP solver for the arc $(u, v)$ is $\sigma_k(v) - \sigma_{k-1}(u) \geq W(M, u, v)$.

   Since $\sigma_{k}(u)=\sigma_{k-1}(u)$ thus $\sigma_k(v) - \sigma_k(u) \geq W(M, u, v)$, and we are done.

2. In case $u \notin V^{k-1}$, during the LP generation, as in line 3, the constraint is generated for the arc $(u, v)$, and thus the solution of the LP (if it exists – the existence is shown later) will satisfy:

   $\sigma_k(v) - \sigma_k(u) \geq W(M, u, v)$.

Q.E.D

**Corollary 3:** If a solution for the merged graph $M$ of layout $L$ is obtained from the solution derivation phase at iteration $i$ – it is a feasible solution of the merged graph $M$ of $L$.

**Proof:** immediate from Proposition 9, since a solution is an $N$-partial solution by its definition.

Proposition 10 states that it is possible to extend in iteration $i$ every partial solution generated in iteration $i-1$.

**Proposition 10:** $M$ is a merged graph of layout $L$ that does not contain positive cycles. For any feasible $(i-1)$–partial solution for a merged graph $M$ that is obtained from the exact solution derivation phase at iteration $i-1$, there exists an $i$–partial solution for the merged graph $M$, and it can be obtained at iteration $i$. 

**Proof:**

We will show the existence of a solution by construction. At iteration $i$, for any vertex $v$, define the solution $\sigma_i(v)$ as follows:

$$
\sigma_i(v) = \max \{ LPW(R_i, u, v) + \sigma_{i-1}(u) \mid u \in V^{i-1} \} \quad \text{if } v \notin V^{i-1} \\
\sigma_i(v) = \sigma_{i-1}(v) \quad \text{otherwise}
$$

Note that since $M$ does not contain positive cycles by Proposition 8, $R_i$ does not contain positive cycles and thus $LPW(R_i, u, v) < \infty$.

It is left to show that $\sigma_i$ is feasible, i.e. by Proposition 9a it satisfies $\forall u, v, \sigma_i(v) - \sigma_i(u) \geq W(M, u, v)$.

Assume in contradiction that there exist $u, v$ so that $\sigma_i(v) - \sigma_i(u) < W(M, u, v)$.

Clearly, since $\sigma_{i-1}$ is feasible, either $v \notin V^{i-1}$ or $u \notin V^{i-1}$ or $u, v \notin V^{i-1}$.

There are 3 cases:

1. $u \in V^{i-1}$, $v \notin V^{i-1}$

By Proposition 8, $LPW(R_i, u, v) = LPW(M, u, v)$, thus

$$
\sigma_i(v) = \max \{ LPW(M, u, v) + \sigma_{i-1}(u) \mid u \in V^{i-1} \} \quad \text{and thus } \sigma_i(v) \geq LPW(M, u, v) + \sigma_{i-1}(u),
$$

thus $\sigma_i(v) - \sigma_{i-1}(u) \geq W(M, u, v)$.

By definition $\sigma_i(u) = \sigma_{i-1}(u)$ and thus $\sigma_i(v) - \sigma_i(u) \geq W(M, u, v)$.

2. $u \notin V^{i-1}$, $v \notin V^{i-1}$

By Proposition 8, $LPW(R_i, u, v) = LPW(M, u, v)$, thus

$$
\sigma_i(v) = \max \{ LPW(M, s, u) + \sigma_{i-1}(s) \mid s \in V^{i-1} \}.
$$

Let $s \rightarrow \ldots \rightarrow u$ be the longest path chosen for $\sigma_i(u)$. Clearly $s \rightarrow \ldots \rightarrow u \rightarrow v$ is a path.

By Proposition 8, $LPW(R_i, u, v) = LPW(M, u, v)$, thus

$$
\sigma_i(v) = \max \{ LPW(M, s, v) + \sigma_{i-1}(s) \mid s \in V^{i-1} \} \geq \max \{ LPW(M, s, u) + \sigma_{i-1}(s) \mid s \in V^{i-1} \} + W(M, u, v) = \sigma_i(u) + W(M, u, v)
$$

and thus $\sigma_i(v) - \sigma_i(u) \geq W(M, u, v)$.

3. $u \notin V^{i-1}$, $v \in V^{i-1}$

By Proposition 8, $LPW(R_i, u, v) = LPW(M, u, v)$, thus

$$
\sigma_i(u) = \max \{ LPW(M, s, u) + \sigma_{i-1}(s) \mid s \in V^{i-1} \}.
$$

Let $s \rightarrow \ldots \rightarrow u$ be the longest path chosen for $\sigma_i(u)$. Note that $s \in V^{i-1}$. Clearly $s \rightarrow \ldots \rightarrow u \rightarrow v$ is a path in $R_i$. 
Note that $\sigma_{i-1}$ is feasible for $M$ and thus $\sigma_{i-1}(v) \geq \max \{LPW(M, s, v) + \sigma_{i-1}(s) \mid s \in V^{i-1} \} \geq \max \{LPW(M, s, v) + \sigma_{i-1}(s) \mid s \in V^{i-1} \} + W(M, u, v) = \sigma_i(u) + W(M, u, v)$.

By definition $\sigma_i(v) = \sigma_{i-1}(v)$ and thus $\sigma_i(v) - \sigma_i(u) \geq W(M, u, v)$.

Q.E.D.

**Corollary 4**: A solution for a merged graph $M$ that does not contain positive cycles always exists and is constructed by exact solution derivation phase.

**Proof**: immediate from Proposition 10, since a solution is an $N$-partial solution by its definition.

**Lemma 1**: If the layout $L$ has a feasible solution that preserves the hierarchy structure, the flow returns a solution $\sigma$ that is feasible for the HCG $H$ of $L$.

**Proof**: 

(1) The layout $L$ has a feasible solution that preserves the hierarchy structure. Thus, by Corollary 1 for the merged graph $M$ generated from the HCG $H$, its merged graph $M$ does not contain positive cycles.

(2) By Corollary 3, the solution $\sigma$ for the merged graph $M$ is a feasible solution of the merged graph $M$.

(3) By Corollary 4, solution $\sigma$ for the merged graph $M$ that does not contain positive cycles can always be obtained.

Thus, using (1), (2), and (3), the flow always returns the solution $\sigma$, which is feasible for the merged graph $M$, i.e.

$\forall t \in \text{templates}(L), \forall u, v \in t \ \sigma(v) - \sigma(u) = LPW(M, u, v)$.

By Proposition 6 for the merged graph $M$ generated from the HCG $H$,

$\forall t \in \text{templates}(L), \forall u, v \in t \ L P W ( M , u , v ) = L P W ( H , u ( i ) , v ( i ) )$, and we are done.

Q.E.D.

The lemma below states that any feasible solution can be derived using the algorithm (depending on the objective function used in the LP problem formulation).
Lemma 2: For any feasible solution $\sigma$ that preserves the hierarchy structure for a given layout $L$ there exists an objective function $F$ so that the flow will construct a solution $\sigma$ for $L$.

Proof:

Let $\sigma$ be a feasible solution for $L$. We shall show that $\sigma$ can be derived in an exact solution derivation phase for some objective function $F$ by induction on $i$ for an $i$-Solved Layout Graph.

Define the linear objective function $F$ for some solution $\varphi$:

$$F(\varphi) = \Sigma_{v \in V} \text{diff}(v)$$

where $\text{diff} = |\sigma - \varphi|$ and it can be written as the LP constraints:

$$\text{diff}(v) \geq \sigma(v) - \varphi(v)$$
$$\text{diff}(v) \geq \varphi(v) - \sigma(v)$$

Observation: $\forall v, \text{diff}(v) \geq 0 \Rightarrow F \geq 0$.

We will show that the flow will construct a solution $\varphi$ so that $F(\varphi) = 0$. Clearly in that case $\varphi = \sigma$. We will show this by induction on the solution construction process.

Basis: By Proposition 2 $\sigma$ is a feasible solution for the HCG $H$ of layout $L$ and thus is equal to the 0-partial solution $\sigma_0$ for all the vertices $V_0$ that are already defined in $\sigma_0$ (borders).

Induction hypothesis: Assume that the $(i-1)$-Partial Solution $\varphi_{i-1}$ can be obtained for the objective function $F$ so that $\varphi_{i-1}$ is equal to $\sigma$ for all vertices in $V^{i-1}$.

Step: will show that $i$-Partial Solution $\sigma_i$ can be obtained for objective function $F$ so that $\sigma_i$ is equal to $\sigma$ for all vertices in $V^i$.

Note that $V^i \setminus V^{i-1} = V_{Ti}$ which are vertices that represent exactly the polygons of instances of template $Ti$.

Define $\varphi^*: V_{Ti} \rightarrow \mathbb{R}$ the solution of all the vertices that represent polygons of instances of template $Ti$ so that $\forall v \in V_{Ti}, \varphi^*(v) = \sigma(v)$.

As part of chooseExactSolution procedure, an LP problem is built and solved by the LP solver.

We need to show 2 items:
1. The solution $\phi^*$ is feasible for the LP solver, \textit{i.e.} it satisfies all the constraints sent to the LP solver.

2. The solution $\phi^*$ is optimal: \textit{i.e.} any other solution $\alpha \neq \phi^*$ will satisfy $F(\alpha) > F(\phi^*)$.

From the claims above it will follow that the solution obtained from the LP solver is $\phi^*$ and due to the definition of $\phi^*$ this will conclude the proof.

We start with 1: Let constraint $c$ be any one of the constraints sent to the LP solver. \textit{w.l.o.g.} the constraint is “arc.vertex2 - arc.vertex1 >= arc.weight” for some arc.

$\sigma$ is a feasible solution that preserves the hierarchy structure for $L$, thus by Proposition 6 it is a feasible solution for $M$, and by Proposition 8 $\sigma$ is a feasible solution for $R_i$. Thus $\phi^*$ satisfies all the constraints generated from $R_i$ and is feasible.

To show 2 we will assume that the solution $\alpha$ for vertices $V_i$ obtained from the LP solver is not equal to $\phi^*$ for some vertex $v'$. Meaning that $\text{diff}(v') > 0$. Since $\forall v \text{ diff}(v) \geq 0$, $F(\sigma) \geq \text{diff}(v') > 0$. However $F(\sigma) = 0$ by the definition of $F$. Thus $\alpha$ is suboptimal.

Q.E.D.

**Corollary 5**: The flow satisfies the requirements as stated in Section 3.2

**Proof**:

1. Lemma 1 implies that the solution $\sigma$ returned by the flow is feasible for the HCG $H$ of $L$; thus $\sigma$ is a feasible solution for $L$ that preserves the hierarchy structure by Proposition 2. A feasible solution $\sigma$ satisfies: $\text{LPW}(G, u, v) \leq \sigma(v) - \sigma(u)$ and in particular it preserves the visibility relation (wires order).

2. $\sigma$ preserves the hierarchical structure, as stated in 1.

3. $\sigma$ is a feasible solution and thus $\sigma$ satisfies: $\forall u, v \in V \text{ LPW}(G, u, v) \leq \sigma(v) - \sigma(u)$. In particular for $u$ and $v$ that are left and right borders of an instance $i$, respectively. Since by construction of the FVLG there are 2 arcs $(u, v)$ and $(v, u)$ so that $W(u, v) = -W(v, u) = \text{width}(i)$ we have:

$\text{width}(i) \leq \sigma(v) - \sigma(u)$

$-\text{width}(i) \leq \sigma(u) - \sigma(v)$
This implies that $\sigma(v) - \sigma(u) = \text{width}(i)$ and thus $\sigma$ satisfies the new sizing and placement at the target manufacturing technology.

4. By the construction of the FVLG, every design rule is represented by an arc $(u, v)$ so that $W(u, v) = \text{design\_rule}$. $\sigma$ is a feasible solution and thus $\sigma$ satisfies: $\forall u, v \in V \text{LPW}(G, u, v) \leq \sigma(v) - \sigma(u)$. In particular $\sigma(v) - \sigma(u) \geq \text{design\_rule}$ and thus $\sigma$ satisfies the new manufacturing technology design rules.

Q.E.D.
5. EXPERIMENTAL RESULTS

We implemented the algorithm and ran tests, based on real designs of industrial microprocessors for a 65nm technology migrating to a 45 nm technology. Several data path blocks data path comprising a few thousand nets each were migrated. The target delay of every signal was set to 0.7 of its delay in the original design. Wire widths and line-to-line spaces specifications have then been derived from the Elmore delay model [23] based on the electrical parameters of the new process technology. The migration algorithm was applied, where positive cycles have been resolved by relaxing wire width and space specifications. As a result, the delay of some nets was degraded. Figure 13 plots the new delays vs. the old ones for one typical block (test id 3). Notice that most of the delays were scaled around 0.7. Of special importance are the encircled data points: they represent nets of significant delays which have been badly scaled. This is where most of the further manual design effort is invested in order to reach design convergence.

![Destination Layout vs Source Layout Net Delays](image)

**Figure 23: Destination Layout vs. Source Layout Net Delays. Encircled data points represent nets of significant delays which have been badly scaled**

The computational complexity of migrating a few of the blocks is summarized in Table 1 and Figure 14. The plot presents the size of the merged graphs and normalized run-times as a function of the flat graph's size. The number of flat graph vertices is given on the
horizontal axis; the size of the merged graph (number of vertices/edges) is presented on the left vertical axis; normalized run-times are given on the right vertical axis. As can be seen, the size of the merged graph is an order of magnitude smaller than that of the flat graph.

Table 2: Tests statistics – size of graphs and runtimes

<table>
<thead>
<tr>
<th>Test id</th>
<th>Flat #vertices</th>
<th>Flat #edges</th>
<th>Merged #vertices</th>
<th>Merged #edges</th>
<th>LP problem max #variables</th>
<th>Runtime [s]</th>
<th>%nets scaled &lt;0.7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1511</td>
<td>5755</td>
<td>499</td>
<td>2249</td>
<td>302</td>
<td>7</td>
<td>98.3</td>
</tr>
<tr>
<td>2</td>
<td>2064</td>
<td>8456</td>
<td>1687</td>
<td>7130</td>
<td>598</td>
<td>11</td>
<td>97.1</td>
</tr>
<tr>
<td>3</td>
<td>4457</td>
<td>16890</td>
<td>1253</td>
<td>6101</td>
<td>434</td>
<td>313</td>
<td>98.4</td>
</tr>
<tr>
<td>4</td>
<td>12928</td>
<td>55892</td>
<td>2535</td>
<td>19524</td>
<td>386</td>
<td>244</td>
<td>95.8</td>
</tr>
<tr>
<td>5</td>
<td>23020</td>
<td>106554</td>
<td>4683</td>
<td>40334</td>
<td>1561</td>
<td>1167</td>
<td>97.9</td>
</tr>
<tr>
<td>6</td>
<td>29714</td>
<td>109212</td>
<td>4205</td>
<td>16255</td>
<td>1173</td>
<td>676</td>
<td>97.6</td>
</tr>
<tr>
<td>7</td>
<td>45964</td>
<td>177946</td>
<td>5963</td>
<td>24928</td>
<td>1308</td>
<td>1532</td>
<td>95.9</td>
</tr>
<tr>
<td>8</td>
<td>62944</td>
<td>248915</td>
<td>7850</td>
<td>44963</td>
<td>1277</td>
<td>2099</td>
<td>98.4</td>
</tr>
<tr>
<td>9</td>
<td>73005</td>
<td>294479</td>
<td>9455</td>
<td>63737</td>
<td>1420</td>
<td>2560</td>
<td>98.2</td>
</tr>
</tbody>
</table>

Figure 24: Tests statistics - size of graphs and runtime
6. DISCUSSION AND FUTURE RESEARCH

We have shown that compaction of interconnect for purposes of layout migration can be performed efficiently while preserving the layout hierarchy. We have devised a data structure, the reduced layout graph that captures the constraints of the original visibility layout graph, and an algorithm that handles them properly. The significant savings were demonstrated on real data.

The next step would be to incorporate circuit design considerations into a process that employs our methodology. Since we use linear programming (LP) as the primary underlying solution engine, some further investigation into the type of optimization criteria that can be accommodated might be necessary.

Finally, LP does not handle discretization requirements gracefully. At extreme dimensions this is necessary and might require integer programming solutions, which in turn could present further algorithmic and complexity challenges.
7. REFERENCES


Destination Layout vs Source Layout Net Delays

לפהו נטייה של השישות הרצויות להפרת מסבלה של פונקציות של ת“These are non-linear functions of the form

לעפרות על האסונות해야 הפוקדולות. העצירה בابتיבו מתאימה לקולות בעבר של המחשים שהופצו נותר

לעפרותrena או לאחרים. כלisión או המספרים של התוכני

ודירקטיות – של בעיות התוכניrena הופכים לעברית התוכני בלתי מעניין וולן ו_signals של

v
בחלק ראשון santa ומכığı, ליניארב תכנות בעיית הבונים nous, אנחנו בתכנון תא כל ייעבור, הוא שתוצאת פתרון התאב אתו עבור. באיור הנתונה זה לשלב דוגמה 6 משפ יאנייצג מהגרף מוצג בו הפתרונות עבור תא B הכאשר פתרון בהסופי תא עבור A נגזר כבר. בעיית הגדולהחסומה תכנות ליניארי: פרמס בתא הפוליגונים כמספר הוא הנעלמים; עחסום אילוצים מספר "הנעלמים גודל ריבועי. הבעיה המתקבלת תעשייתיים משולבים מעגלים איעבור הנה. תוצאות בחלק הנתונים אלו בגדלים עבור תעשייתיים משולבים מעגלים. לעמוככיםmarkers אנחנו להשלמתו ניתן תמיד הזה שהתהליך חמים. האלגוריתם יمتاز, והרצינו אתו יצירה בテクノロジー תעשייתיים משולבים מעגלים מספר על 65nm . משולבים מעגלים קיבלו ומרים מיצירת ייצור בטכנולוגית 45nm . בטבלה נתונות התוצאות 1.

**טבלה 1: ת.isUser תעשיות משולבים معגלים בטכנולוגית 4וע**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1511</td>
<td>5755</td>
<td>499</td>
<td>2249</td>
<td>302</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>2064</td>
<td>8456</td>
<td>1687</td>
<td>7130</td>
<td>598</td>
<td>11</td>
<td>4</td>
<td>7</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>4457</td>
<td>16890</td>
<td>1253</td>
<td>6101</td>
<td>434</td>
<td>313</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>12928</td>
<td>55892</td>
<td>2535</td>
<td>19524</td>
<td>386</td>
<td>244</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>23020</td>
<td>106554</td>
<td>4683</td>
<td>40334</td>
<td>1561</td>
<td>1167</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>29714</td>
<td>109212</td>
<td>4205</td>
<td>16255</td>
<td>1173</td>
<td>676</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>45964</td>
<td>177946</td>
<td>5963</td>
<td>24928</td>
<td>1308</td>
<td>1532</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>62944</td>
<td>248915</td>
<td>7850</td>
<td>44963</td>
<td>1277</td>
<td>2099</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>73005</td>
<td>294479</td>
<td>9455</td>
<td>63737</td>
<td>1420</td>
<td>2560</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

 possui מדגמן אורב, חכם מאוד האפשרות את הброועים שבמקטגנה ובמקטגהנוגיות ישנה ובמקטגהנוגיות החמישה. hipוס בינ הבויות והיתר נתן בחר לקייר hil קייק על הגיה מגרגנט פלינון, ציון החשקה של פלי hipוסה היא כמב不克מש שאינד נוחות אנר חותר - ומידה. את השיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשיפור והשuridad.

Elmore. קיון לארט שמסטר גן של מראות, בעלת השתייה שלמקתעה,ANTI-1970200. ביוכיסט מוצללי של מקטגנה . רשתת אל יevenodd ליפש皮革 על עקר מגע.
ויתכן שהโหลניא בטעות התפרסמה במקום הנכון במסמך. המילים "הפוליגון" должна להיות "הגרף".

 notícia מתכונת הפוליגון התפרסמה במקום הנכון במסמך. המילים "הפוליגון" должна być "הגרף".

 לטיפול פוליגון saluteكيفון התפרסמה במקום הנכון במסמך. המילים "הפוליגון" должны быть "الجدول".

 המילה "הפוליגון" должна быть "הגרף".

 המילה "הפוליגון" должна być "הגרף".

 המילה "הפוליגון" должна być "הגרף".

 המילה "הפוליגון" должна być "הגרף".

 המילה "הפוליגון"应该是 "הגרף".

 המילה "הפוליגון" 应该是 "hagr".

 המילה "הפוליגון" 应该是 "hagr".

 המילה "הפוליגון" 应该是 "hagr".

 המילה "הפוליגון" 应该是 "hagr".

 המילה "הפוליגון" 应该是 "hagr".

 המילה "הפוליגון" 应该是 "hagr".

 המילה "הפוליגון" 应该是 "hagr".

 המילה "הפוליגון" 应该是 "hagr".

 המילה "הפוליגון" 应该是 "hagr".

 המילה "הפוליגון" 应该是 "hagr".

 המילה "הפוליגון" 应该是 "hagr".

 המילה "הפוליגון" 应该是 "hagr".

 המילה "הפוליגון" 应该是 "hagr".

 המילה "הפוליגון" 应该是 "hagr".

 המילה "הפוליגון" 应该是 "hagr".

 המילה "הפוליגון" 应该是 "hagr".

 המילה "הפוליגון" 应该是 "hagr".

 המילה "הפוליגון" 应该是 "hagr".

 המילה "הפוליגון" 应该是 "hagr".

 המילה "הפוליגון" 应该是 "hagr".

 המילה "הפוליגון" 应该是 "hagr".

 המילה "הפוליגון" 应该是 "hagr".

 המילה "הפוליגון" 应该是 "hagr".

 המילה "הפוליגון" 应该是 "hagr".

 המילה "הפוליגון" 应该是 "hagr".

 המילה "הפוליגון" 应该是 "hagr".

 המילה "הפוליגון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "حافظון" 应该是 "hagr".

 המילה "محافظון" 应该是 "hagr".

 המילה "محافظון" 应该是 "hagr".

 המילה "محافظון" 应该是 "hagr".

 המילה "محافظון" 应该是 "hagr".

 המילה "محافظון" 应该是 "hagr".
鼓舞我们全心全意地追求基于视觉的表达方式，使得视觉可以更好地服务于我们的人类认知。在视觉表达中，我们可以通过视觉语言和视觉符号来传递信息，使得人类可以更好地理解和表达自己的想法和情感。
**Diagram Description:**

The diagram illustrates a flowchart or algorithm representation. It shows a decision-making process with conditions and loops, likely related to graph theory or algorithm analysis. The notation includes variables and operations typical of technical documentation.

---

**Figure Notes:**

- The diagram includes a representation of a graph (G) and variables such as i, N, T_i, M_i, R_i, and M_N.
- The flowchart involves conditions like 'false' and 'true' branches.

מסקור והבנה בשתי מערכות של פרו"ח ישראלי רובר. חכ全自动 לתורה טכנית, תש"כ, בוית הספרית של הנדסת חשמל, חכ全自动 ח"ש מכונן, חכ全自动 ח"ש נגרש, אוניברסיטת בר-אילן, ורıyor/ית רון פנשר, חכ全自动 לתורה טכנית, חכ全自动 ח"ש מכונן, חכ全自动 ח"ש נגרש, אוניברסיטת בר-אילן, והערכה בפקולטה למדעי המחשב, הטכניון, ירושלים, תש"כ, וישראלי.

борגני מתודות שלב שלב למשתתפים והקרבה, במגזר לאריאט לפי, לביג לינאוד לאקיר לביג, על שוגי שוגי.

בצבייל.
חתנמיה מגה-息息וריות מבוססת-תאימי של
משלחת של מגוון משולבים

תいろ על מחקר

לשם מחלי חלך של דרישות לmeyeceği הווה
ricing שמות זרדים במדעי המחשב

יבגני.Shapes

הנושאים לפרסומם – מרכז מדעי טכנולוגי
רגל 5769 - חיפה – אפריל 2009