StarPlace: A new analytic method for FPGA placement

M. Xu, G. Grewal, S. Areibi *

Faculty of Engineering and Computer Science, University of Guelph, Guelph, Canada

1. Introduction

Placement is one of the most important and time-consuming problems in field-programmable gate array (FPGA) design [1]. The input to the problem is a circuit net-list specifying the various types of logic blocks to be implemented and the interconnections between them. The result of placement is the mapping of all blocks onto physical resources on the target FPGA in a way that minimizes one or more objective functions. The most basic objective is to minimize the total amount of interconnect (estimated by wire-length) required to route signals between blocks. Reducing wire-length is an important goal as the amount of interconnect in any given region of the FPGA is fixed. Thus, if more interconnect is required than is actually available, the circuit may not be routable. Moreover, reducing wire-length also reduces the values of a number of other important design parameters including circuit delay [2] and power [3].

Unfortunately, as modern FPGAs reach close to one million logic blocks, compilation times can easily take hours or even days to complete. It is estimated that the time required to perform placement accounts for roughly 50% of the total compilation time [1]. With even larger capacity FPGAs on the horizon, this problem in genetic algorithm (GA)-based methods. These methods start with an initial placement and then iteratively seek to improve upon it by searching for small perturbations to the placement that result in better solutions. An example of such a method is the well-known versatile place-and-route (VPR) tool [8] which employs the basic template for SA, but with several very effective placement-specific enhancements. Overall, VPR provides very good results, and is widely used in the FPGA CAD flow. However, VPR’s results are not scalable, since simulated annealing is inherently slow. The ultra-fast placement (UFP) method proposed by Sankar and Rose [17] seek to improve the runtime of VPR by combining VPR with a multi-level clustering strategy. Results show that a 50x improvement in runtime can be achieved, but at the cost of 33% more wire-length. Techniques for parallelizing simulated
annealing have also been used in [18] to accelerate VPR on multi-core hardware. However, as simulated annealing provides few opportunities for parallelization, only a modest speedup of 2.2x is achieved.

Recently, analytic placement methods have garnered a significant amount of attention within the application specific integrated circuit (ASIC) community because they have generated the best results at the most recent ISPD placement contest [19]. Unlike SA-based algorithms, analytic algorithms tackle the placement problem from the top down considering global connectivity rather than evaluating many small-scale provisional modifications. Often, a quadratic wire-length objective function is used. Clearly, using squared wire-length, by itself, does not give the best wire-length. However, the resulting cost function is continuously differentiable and can be minimized efficiently by solving a system of linear equations.

Despite their popularity in the ASIC community, relatively few analytic placement methods exist for FPGAs. The work in [20] presents a placement algorithm for FPGAs based on quadratic programming. The conjugate gradient method is used for solving the linear equations. However, since squared wire-length is optimized, the quality of results produced by the analytic placer are not competitive, requiring additional rounds of simulated annealing to be performed to further improve solution quality. CAPRI [21], on the other hand, employs a novel analytic metric of distance that seeks to better model delays within the FPGA routing grid. CAPRI shows improvement in placement quality compared with VPR, but only a marginal 2x improvement in speed. More recently, the study reported in [1] has sought to evaluate the suitability of using existing ASIC placement algorithms to solve the FPGA placement problem. In total, three state-of-the-art academic ASIC placers were considered: Capo [6], mPL [22], and FastPlace [23]. The results of the study showed that FastPlace, which is based on quadratic programming, is an order-of-magnitude faster than VPR. However, the resulting average wire-length is 20–50% worse compared with VPR. Interestingly, Capo and mPL, which are based on multi-level partitioning and non-linear optimization, respectively, were not found to be significantly faster than VPR. However, the quality of results in both cases was found to be competitive with VPR.

In this paper, we present an analytic placement method for FPGAs that seeks not only to run faster than VPR, but also to produce high-quality results that are competitive, if not superior, to those achieved by VPR. To achieve these goals, we employ a smooth linear wire-length objective [24] that can be minimized by solving a system of non-linear equations.

The main contributions of our work include the following.

- We introduce a near-linear wire-length function, called star+. The proposed model is continuously differentiable, and can be used with both analytic and iterative improvement placement methods. Moreover, the time to calculate incremental changes in cost from moving a block/swapping a pair of blocks can always be computed in O(1) time.
- We perform analysis and empirical studies of star+ using VPR [8]. When tested using the 20 MCNC benchmarks [25], our results show that the star+ model achieves a 2.4% reduction in wire-length and a 6–9% reduction in critical-path delay compared with HPWL [26]. Both HPWL and star+ require roughly the same amount of computational effort to compute. However, our results show that as net size (cardinality) increases, star+ is much faster to re-compute following an improving move/swap.
- Based on star+ we implement an analytic placement method, called StarPlace. In order to solve the resulting non-linear equation system we try two different iterative solvers: successive over-relaxation (SOR) and conjugate gradient (CG). In general, stationary methods, like SOR, are computationally cheaper per iteration compared with non-stationary methods, like CG. However, the additional computational expense of CG is often justified by much smaller numbers of iterations. We find SOR to be the best solver both with respect to runtime and quality of solutions.

The remainder of this paper is organized as follows. Section 2 presents a new model for estimating wire-length for FPGA placement. Section 3 demonstrates the basic structure of an analytic placement flow based on SOR and CG, respectively. Results obtained along with analysis of these results are presented in Section 4. Conclusions and future work are finally presented in Section 5.

2. Star+

In the placement phase, it is computationally expensive to determine the exact configurations of the routing resources required to realize physical connections for CLBs and I/O pads, since the problem is NP-hard. For this reason, the routing cost is approximated during placement using a net model. The speed and accuracy of the net model have a significant effect on performance of any placement method. In practice, the half-perimeter wire-length (HPWL) model [26] is the most widely used method for estimating the wire-length of a net. The wire-length is approximated by half the perimeter of the smallest bounding rectangle that encloses all terminals in the net. For a net with two or three terminals, the routing cost obtained by the HPWL model is accurate. When there are more than three terminals, a factor [27] is introduced to compensate for the fact that the HPWL model underestimates the wire-length necessary to connect all blocks. A fast method (with amortized complexity of O(1)) for incremental net bounding-box updating is presented in [28]. Unfortunately, HPWL cannot be efficiently minimized [29]. Therefore, many analytic methods employ a quadratic wire-length objective. However, squared wire-length tends to over-emphasize the optimization of longer nets at the expense of shorter nets. To compensate, we propose a more accurate, near-linear objective, called star+. An important concept in the star+ model is the center-of-gravity of a net. The center-of-gravity of a net is defined as follows. First, for Neti, we use k to represent the cardinality of the net (i.e., the number of blocks connected to Neti). Let ci represent the center-of-gravity of net i; we define the x-coordinate of ci as

\[
x_{cl} = \frac{1}{k} \sum_{i=1}^{k} x_i
\]

The expression \( x_{cl} \) simply means that blocki connects to Neti, and \( x_i \) represents the x-coordinate of blocki. The star+ model is defined as follows:

\[
\text{Net}/l_x = \gamma \sqrt{\sum_{v_i \in \text{Net}_i} (x_i - x_{cl})^2 + \phi}
\]

The factor \( \gamma \) in Eq. (2) is used to compensate for under-estimation of the wire-length. The parameter \( \phi \) is a positive number, which is used to make Eq. (2) always differentiable. (Note: The star+ model handles the x- and y-dimensions separately. For simplicity, only the x-dimension is discussed here, but the y-dimension is processed similarly.)

The graphical representation of the star+ model is based on the traditional star model. In a star model each block has a connection to the center-of-gravity of the net. The length of each edge is the
quadratic distance \((x_i - x_{cl})^2\) from block \(b\) to the center-of-gravity \(x_{cl}\). The star+ model differs from the traditional star model in that it does not seek to minimize the sum of all squared distances between any pair of blocks that connect a common net, but rather seeks to minimize the sum of the square roots of the sum of the quadratic distances between each block and the center-of-gravity of a net. Thus, by avoiding quadratic distance measures, the model seeks to avoid overemphasizing the cost of long nets at the cost of shorter nets when performing optimization.

Eq. (2) has three important features: First, it is differentiable when \(\phi\) is greater than zero, hence it is suitable for use with analytic placement methods that rely upon “second-order” information provided by the Hessian matrix. Second, the incremental change in cost caused by swapping two blocks can always be calculated in constant time. This feature is very important as it allows the star+ model to be used efficiently in iterative solvers, like SOR and CG, as well as iterative placement methods, like simulated annealing, where the time to calculate the incremental distance estimate, thus avoiding the problem of minimizing longer nets at the expense of shorter nets.

### 2.1. \(O(1)\) time cost update

In its current form, Eq. (2) is not suitable for estimating wire-length, as it is too expensive to calculate each time a block moves to a new position. Therefore, it must be transformed into a new form that allows incremental changes in cost (resulting from block movement) to be calculated in constant time. This can be done as follows:

\[
||Net||_x = \sum_{i \in N} (x_i - x_{cl})^2 + \phi
\]

\[
= \sum_{i \in N} x_i^2 - 2x_{cl} \sum_{i \in N} x_i + \sum_{i \in N} x_{cl}^2 + \phi
\]

\[
= \sum_{i \in N} x_i^2 - 2x_{cl}^2 + k_{x_{cl}} + k_{x_{cl}}^2 + \phi
\]

\[
= \sum_{i \in N} x_i^2 - k_{x_{cl}}^2 + \phi
\]

Let \(U_i = \sum_{i \in N} x_i^2\) and \(V_i = \sum_{i \in N} x_i\), then we have

\[
||Net||_x \approx \sum_{i \in N} x_i^2 - k_{x_{cl}}^2 + \phi
\]

\[
= \sum_{i \in N} x_i^2 - \frac{k_{x_{cl}}^2}{k_i} + \phi
\]

Now suppose \(b\) is a block connected to Net \(i\) (i.e., \(b \in N_{i}\)) and that block \(b\) moves from position \(x_b\) to the new position \(x_{b_{new}}\). This movement causes \(U_i, V_i\) and \(||Net||_x\) to change values. The new value of \(U_{b_{new}}\) and \(V_{b_{new}}\) can be calculated, respectively, in constant time using the following formulas:

\[
U_{b_{new}} = U_b - x_b^2 + (x_{b_{new}}^2 - x_b^2)
\]

\[
V_{b_{new}} = V_b - x_b + x_{b_{new}}
\]

As a result, the new value of \(||Net||_x\) can also be calculated in a constant time using Eq. (4).

### 2.2. Evaluating star+

The objective of the following experiments is to show that the star+ model is both accurate and fast to compute. As VPR [8] is the current public domain state-of-the-art place-and-route tool, we use it as a basis of comparison. Our methodology is as follows. First, we run VPR on all 20 MCNC [30] benchmarks using VPR’s original placement algorithm, VPlace, which uses the HPWL wire-estimation model. Each placed circuit is then routed. Following routing, the minimum number of required channels, critical path, and total wire-length is obtained for each circuit. The HPWL model in VPlace is then replaced with the star+ model, and the process repeated. Finally, the routing results from both models are used to compare the speed and effectiveness of the two models head-to-head.

#### 2.2.1. Routability—channel widths

We begin by considering the effectiveness of the star+ model in producing routable placements. As a measure of the quality of a routable placement, we measure the minimum channel width (or number of tracks per channel) that VPR’s router needs to successfully route each circuit placement. Channel width is one of the most commonly used criteria for assessing the routability of a potential placement.

For the purpose of this experiment, we downloaded VPR 4.3 source code, architecture file, and the complete set of 20 MCNC benchmark circuits used by VPR from [25]. We used the default architecture file as is, which assumes that each CLB contains four LUTs, and each LUT has four inputs and is paired with one flip-flop. We first ran the benchmarks through the entire VPR flow having first configured VPlace to use the HPWL estimation model, and to report the minimum channel width when using its breadth-first strategy to route the components. We then repeated the experiment, but this time configured VPR to route the components using its timing-driven algorithm. Finally, we replaced the HPWL model in VPlace with our star+ model, and repeated the previous experiments using both breadth-first and timing-driven timing options.

For each benchmark, each model, and each routing algorithm, VPlace was executed with the option \(inner\_num\) equal to 1 and 10, respectively. VPlace uses \(inner\_num\) to trade-off quality for speed. In particular, the number of moves attempted at each temperature equals: \(inner\_num\times\)the number of blocks. The default value of \(inner\_num\) is 10. Specifying an \(inner\_num\) of 1 will speed up VPlace by a factor of 10, but will typically reduce the quality by about 10%. Setting \(inner\_num\) greater than 10 barely improves the quality of the placement, but does increase the running time of VPlace.

VPlace is based on simulated annealing which is stochastic. Therefore, running VPlace more than once with different seed values results in different placements being produced. To make the previous experiments more accurate, for each set of parameters, VPR is executed 10 times using 10 different (randomly generated) seed values.

Table 1 summarizes the previous experimental results. The entries in the table indicate the smallest “successful” routable channel width found for at least half of the 10 runs. The last row of the table shows the aggregate of the smallest “successful” routable channel widths for all benchmarks and each situation. Overall, it can be seen that both HPWL and star+ performed similarly across the entire set of benchmarks, with star+ slightly outperforming HPWL when \(inner\_num\) = 1 (215 versus 216 using breadth first and 223 versus 226 using timing-driven).

#### 2.2.2. Critical-path delay

We now turn our attention to critical-path delay. Table 2 reports the critical-path delay computed by VPR’s router when
performing the experiments described in Section 2.2.1. Column 1 identifies the benchmark. Column 2 gives the channel width used by the router (Note: When providing a minimum channel width to VPR’s router for each benchmark, we used the largest minimum successful channel width from Table 1 plus 1. By using a channel width one more than the largest minimum, the probability of the router finding a feasible route is effectively 1 in all cases.).

Column 3 gives the average critical-path delay for HPWL when VPlace is run with inner_num equal to 1. Column 4 provides similar information for the star+ model. Column 5 shows the P-value (described below). Columns 6–8 give similar information as columns 3–5, but for inner_num equal to 10. Note that the results presented in Table 2 were obtained running VPR’s router with the timing-driven option invoked. (Results were also obtained using the breadth-first timing option; however, these results (critical-path delays) in all cases were consistent with, but inferior to, those found using the timing-driven option. As such, they are not reported.)

Our results show that when VPlace is run with inner_num equal to 1, star+ results in a lower critical-path delay compared with HPWL for 15 of the 20 benchmarks. Moreover, the placements found using star+ are, on average, 6% faster than those found using HPWL. Similarly, when VPlace is run with inner_num equal to 10, star+ results in a lower critical-path delay compared with HPWL for 15 of the 20 benchmarks. Moreover, the placements found using star+ are, on average, 6% faster than those found using HPWL. The previous results were somewhat unexpected. To verify whether or not the results of the previous experiments were due to variance, we employed a standard t-test [13].

The t-test can provide insight into whether or not the difference in performance of star+ and HPWL, with respect to critical-path delay, is attributable to “chance”. Using the data in Table 2, the t-test was used to test the null hypothesis that the mean value of the results found using HPWL and star+, respectively, is equal, implying that the differences in results between the two models are due to “chance”. The alternative hypothesis states that HPWL and star+ are unique models that provide different results, not attributable to “chance”. The corresponding t-test P-value scores for the previous four experiments are reported in Table 2 (columns 5 and 8). The null hypothesis is rejected in favor of the alternative hypothesis in cases, where the P-value for a set of results is less than 0.05. This corresponds to a significance level of 5%. For cases where the P-value is greater than 0.05, the null hypothesis is not rejected.

The P-values in Table 2 reveal that for inner_num = 1 the null hypothesis is rejected for 13 of the 20 cases; indicating that in 65% of the test cases, the difference between the star+ and HPWL models with respect to critical-path delay is not due to chance. Also, Table 2 reveals that for inner_num = 10 the null hypothesis is rejected for 9 of the 20 cases. Thus, from this we can conclude that variance can only account for the difference in wire-length

Table 1
Summary of minimum routable channel widths.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>HPWL</th>
<th>star+</th>
<th>HPWL</th>
<th>star+</th>
</tr>
</thead>
<tbody>
<tr>
<td>inner_num = 1</td>
<td>inner_num = 10</td>
<td>inner_num = 1</td>
<td>inner_num = 10</td>
<td></td>
</tr>
<tr>
<td>alu4</td>
<td>11</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>apex2</td>
<td>12</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>apex4</td>
<td>13</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>Bigkey</td>
<td>7</td>
<td>6</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>Clma</td>
<td>13</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>Des</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Diffreq</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Dsp</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Elliptic</td>
<td>12</td>
<td>10</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>ex1010</td>
<td>11</td>
<td>11</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>ex5p</td>
<td>14</td>
<td>13</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>Frisc</td>
<td>13</td>
<td>12</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>MikeX3</td>
<td>11</td>
<td>12</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>Pdc</td>
<td>17</td>
<td>16</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>S298</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>S38417</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>S38584.1</td>
<td>9</td>
<td>9</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Seq</td>
<td>12</td>
<td>12</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>Spla</td>
<td>14</td>
<td>15</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>Tseng</td>
<td>8</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Total</td>
<td>216</td>
<td>215</td>
<td>200</td>
<td>206</td>
</tr>
</tbody>
</table>

Table 2
Critical-path delay (ns).

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>inner_num = 1</th>
<th>inner_num = 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPWL</td>
<td>star+</td>
<td>HPWL</td>
</tr>
<tr>
<td>inner_num = 1</td>
<td>inner_num = 10</td>
<td></td>
</tr>
<tr>
<td>alu4</td>
<td>12</td>
<td>120.3</td>
</tr>
<tr>
<td>Apex2</td>
<td>14</td>
<td>128.7</td>
</tr>
<tr>
<td>Apex4</td>
<td>15</td>
<td>127.9</td>
</tr>
<tr>
<td>Bigkey</td>
<td>8</td>
<td>100.9</td>
</tr>
<tr>
<td>Clma</td>
<td>15</td>
<td>264.9</td>
</tr>
<tr>
<td>Des</td>
<td>9</td>
<td>123.0</td>
</tr>
<tr>
<td>Diffreq</td>
<td>10</td>
<td>106.1</td>
</tr>
<tr>
<td>Dsp</td>
<td>8</td>
<td>91.0</td>
</tr>
<tr>
<td>Elliptic</td>
<td>14</td>
<td>257.3</td>
</tr>
<tr>
<td>ex1010</td>
<td>13</td>
<td>205.5</td>
</tr>
<tr>
<td>ex5p</td>
<td>16</td>
<td>116.0</td>
</tr>
<tr>
<td>Frisc</td>
<td>15</td>
<td>227.3</td>
</tr>
<tr>
<td>MikeX3</td>
<td>13</td>
<td>108.4</td>
</tr>
<tr>
<td>Pdc</td>
<td>19</td>
<td>254.4</td>
</tr>
<tr>
<td>s298</td>
<td>9</td>
<td>249.9</td>
</tr>
<tr>
<td>s38417</td>
<td>9</td>
<td>196.9</td>
</tr>
<tr>
<td>s38584.1</td>
<td>10</td>
<td>123.8</td>
</tr>
<tr>
<td>Seq</td>
<td>14</td>
<td>123.0</td>
</tr>
<tr>
<td>Spla</td>
<td>16</td>
<td>205.0</td>
</tr>
<tr>
<td>Tseng</td>
<td>9</td>
<td>81.7</td>
</tr>
<tr>
<td>Total</td>
<td>3204.0</td>
<td>2919.7</td>
</tr>
</tbody>
</table>
estimates in approximately 40% of the test cases. For the remaining 60% of the test cases, the star+ and HPWL models were found to be different, with star+ outperforming HPWL with respect to critical-path delay.

2.2.3. CPU runtime

One of the reasons that the HPWL model is so popular is that calculation of a bounding box can be performed quickly. Calculating the bounding box of a net from scratch is linear with respect to the number of blocks in the net. However, Betz et al. [28] have developed a method, called incremental bounding box evaluation, which can recompute the bounding box in a constant amount of time on average. In practice, the only case for which the net bounding box must be recomputed from scratch is when the block moves inward and is the only block on a side of the bounding box. In this case, the recomputation takes $O(k)$ time, while in all other cases it is $O(1)$. In contrast, the recomputation of the star+ model always takes $O(k)$ time. To implement the constant-time recomputation of the star+ model, one has to store $U_l$ which is $\sum_{i=1}^{n} x_i^2$, and $V_l$ which is $\sum_{i=1}^{n} x_i$, for each net $l$, as shown below in Algorithm 1.

Algorithm 1. Constant-time recomputation of the star+ model

```c
if(xnew != xold) { // [block x has moved
    delta = xnew - xold;
    V += delta;
    U += delta * (xnew + xold);
}
```

One immediately notices that the recomputation of the star+ model is extremely straightforward and very easy to implement. Most importantly, it is $O(1)$ in all cases. It should be noted, however, that the star+ model uses Eq. (4) which has a square-root operation, while HPWL requires multiplication and addition operations to be performed.

Table 3 reports the average running time of VPlace when performing the experiments described in Section 2.2.1. Column 1 identifies the benchmark. Columns 2 and 3 give the average-runtime for HPWL and star+, respectively, when VPlace is run with inner_num equal to 1. (Running VPlace with an inner_num of 10 simply causes the runtime to increase by a factor of 10 and, hence, is not shown.) The last row of Table 3 shows the aggregate runtime for VPR for each model. Clearly, there is very little difference in the actual run-times.

It should be noted, however, the run-times reported in Table 3 ultimately depend on the number of actual swaps performed during the optimization process. Thus, the fact that the total time required by star+ is slightly more than the total time required when using HPWL does not necessarily mean that HPWL is faster to compute. On the contrary, changes in cost can be computed faster using the star+ model as the size of the nets becomes larger. To illustrate this, we randomly generate nets with cardinalities (number of blocks) of size 2, 3, 5, 10, 50, 100, 500, 1000, and 2000. Each net is then randomly placed on a 100 by 100 FPGA chip. We then perform one million improving swaps/moves on each net, and calculated the average time required by star+ and HPWL to recompute the wire-length estimate. The results are shown in Table 4, where the times recorded in the table are given in nanoseconds. Notice that the time required by HPWL to recompute the wire-length estimate varies from 19 ns for a small net with just two blocks to 8380 ns for a large net with 2000 blocks. However, the star+ model requires a small recomputation time of only 53 ns, for all size nets.

2.2.4. Wire segments

As a final basis of comparison, we compare the star+ and HPWL models with respect to the total number of wire segments required by

### Table 3

<table>
<thead>
<tr>
<th>Average CPU runtimes (seconds).</th>
<th>HPWL</th>
<th>star+</th>
</tr>
</thead>
<tbody>
<tr>
<td>alu4</td>
<td>2.66</td>
<td>2.91</td>
</tr>
<tr>
<td>apex4</td>
<td>3.86</td>
<td>4.06</td>
</tr>
<tr>
<td>Bigkey</td>
<td>2.2</td>
<td>2.3</td>
</tr>
<tr>
<td>Clima</td>
<td>3.83</td>
<td>3.7</td>
</tr>
<tr>
<td>Des</td>
<td>3.47</td>
<td>3.67</td>
</tr>
<tr>
<td>Diffq</td>
<td>2.95</td>
<td>2.95</td>
</tr>
<tr>
<td>Dsp</td>
<td>2.8</td>
<td>2.89</td>
</tr>
<tr>
<td>Elliptic</td>
<td>10.11</td>
<td>10.09</td>
</tr>
<tr>
<td>ex1010</td>
<td>13.55</td>
<td>14.28</td>
</tr>
<tr>
<td>ex5p</td>
<td>1.83</td>
<td>1.86</td>
</tr>
<tr>
<td>Frisc</td>
<td>9.59</td>
<td>9.94</td>
</tr>
<tr>
<td>misex3</td>
<td>2.53</td>
<td>2.69</td>
</tr>
<tr>
<td>Pdc</td>
<td>11.24</td>
<td>14.33</td>
</tr>
<tr>
<td>S298</td>
<td>3.45</td>
<td>4</td>
</tr>
<tr>
<td>S38417</td>
<td>22.69</td>
<td>21.66</td>
</tr>
<tr>
<td>S38584.1</td>
<td>22.3</td>
<td>21.27</td>
</tr>
<tr>
<td>Seq</td>
<td>3.45</td>
<td>3.81</td>
</tr>
<tr>
<td>Spla</td>
<td>9.84</td>
<td>10.64</td>
</tr>
<tr>
<td>Tseng</td>
<td>1.86</td>
<td>1.89</td>
</tr>
<tr>
<td>Total</td>
<td>169.5</td>
<td>172.15</td>
</tr>
</tbody>
</table>

### Table 4

<table>
<thead>
<tr>
<th>Cardinality</th>
<th>HPWL (ns)</th>
<th>star+ (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>19</td>
<td>53</td>
</tr>
<tr>
<td>3</td>
<td>26</td>
<td>53</td>
</tr>
<tr>
<td>5</td>
<td>38</td>
<td>53</td>
</tr>
<tr>
<td>10</td>
<td>50</td>
<td>53</td>
</tr>
<tr>
<td>50</td>
<td>227</td>
<td>53</td>
</tr>
<tr>
<td>100</td>
<td>431</td>
<td>53</td>
</tr>
<tr>
<td>500</td>
<td>2127</td>
<td>53</td>
</tr>
<tr>
<td>1000</td>
<td>4213</td>
<td>53</td>
</tr>
<tr>
<td>2000</td>
<td>8380</td>
<td>53</td>
</tr>
</tbody>
</table>

### Table 5

<table>
<thead>
<tr>
<th>CW</th>
<th>inner_num = 1</th>
<th>inner_num = 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPWL</td>
<td>star+</td>
<td>P-value</td>
</tr>
<tr>
<td>alu4</td>
<td>11</td>
<td>22038</td>
</tr>
<tr>
<td>Apex2</td>
<td>13</td>
<td>32545</td>
</tr>
<tr>
<td>Bigkey</td>
<td>8</td>
<td>22395</td>
</tr>
<tr>
<td>Clima</td>
<td>14</td>
<td>142509</td>
</tr>
<tr>
<td>Des</td>
<td>8</td>
<td>27916</td>
</tr>
<tr>
<td>Diffq</td>
<td>9</td>
<td>16263</td>
</tr>
<tr>
<td>Dsp</td>
<td>7</td>
<td>17171</td>
</tr>
<tr>
<td>Elliptic</td>
<td>13</td>
<td>53811</td>
</tr>
<tr>
<td>ex1010</td>
<td>10</td>
<td>72613</td>
</tr>
<tr>
<td>ex5p</td>
<td>15</td>
<td>19923</td>
</tr>
<tr>
<td>Frisc</td>
<td>14</td>
<td>59957</td>
</tr>
<tr>
<td>misex3</td>
<td>13</td>
<td>22699</td>
</tr>
<tr>
<td>Pdc</td>
<td>19</td>
<td>104298</td>
</tr>
<tr>
<td>S298</td>
<td>9</td>
<td>22703</td>
</tr>
<tr>
<td>S38417</td>
<td>9</td>
<td>66586</td>
</tr>
<tr>
<td>S38584.1</td>
<td>10</td>
<td>63519</td>
</tr>
<tr>
<td>Seq</td>
<td>13</td>
<td>29610</td>
</tr>
<tr>
<td>Spla</td>
<td>16</td>
<td>71103</td>
</tr>
<tr>
<td>Tseng</td>
<td>8</td>
<td>10419</td>
</tr>
</tbody>
</table>

Total 902282 | 880271 | 836281 | 838039 |
As discussed, respectively. We now discuss how these values were arrived at.

Given that a placement is of no value if it cannot be responsible for improving the quality of the placement. In all of the differences between the star+ wire-length estimate and the actual placements are. Using Effect of varying VPR’s router to route each placement. Table 5 reports the total number of wire segments required to route each placement when performing the experiments described in Section 2.2.1. Column 1 identifies the benchmark. Column 2 gives the channel width (CW) used by the router. Columns 3 and 4 indicate the average number of wire segments (WS) required to route each placement for HPWL and star+, when VPlace is run with inner_num equal to 1. Column 5 gives the P-value. Columns 6–8 provide similar information as Columns 3–5 when VPlace is run with inner_num equal to 10.

Results obtained show that when VPlace is run with inner_num equal to 1, star+ results in a lower total number of wire segments compared with HPWL for 17 of the 20 benchmarks. The number of wire segments required when using star+ are, on average, 2.4% less than based on HPWL. In contrast, when VPlace is run with inner_num equal to 10, star+ results in a lower total number of wire segments compared with HPWL for only 11 of the 20 benchmarks. Moreover, the number of wire segments required when using star+ are, on average, 0.2% more than those found when using HPWL. On closer inspection, star+ outperforms HPWL for 17 of the 20 benchmarks, with 12 of these cases having a P-value less than 0.05 (indicating that the difference between the two models cannot be attributable to chance).

3. Structure of an analytic placer

Having established that the star+ model is effective when employed in SA-based methods, like VPR [8], we now seek to determine its suitability for use with analytic placement methods. Algorithm 2 shows the basic structure of the proposed StarPlace analytic placement algorithm. Since the numerical techniques employed by analytic placement methods typically apply to graphs, a net model must be used to transform the circuit net-list (represented as a hyper-graph) into a graph prior to solving any optimization problem; StarPlace uses the star+ model for this purpose. To avoid trivial solutions when solving the subsequent (non-linear) equation system, some blocks must have their locations on the FPGA initially fixed. We do this by employing a novel algorithm, called shrubPlace, to pre-place all I/O pads around the perimeter of the FPGA prior to placing logic blocks as will be explained later. This pre-placement is not the final placement of I/O pads, but helps with the placement of logic blocks later on as some logic blocks share connections with various I/O pads. Based on the star+ model presented in Section 2, the one-dimensional (x direction only) placement problem for all nets can be expressed by the following objective:

\[
f(x) = \sum_{i} \left( \gamma \sum_{j} (X_i - X_{d,j})^2 + \phi \right)
\]

Algorithm 2. Analytic placement approach

1: Convert circuit hyper-graph to graph using star+
2: pre-place all I/O pads
3: repeat
4: solve non-linear equation system
5: partition solution to enforce legality constraints
6: until maximum number of iterations reached

The corresponding two-dimensional placement problem is addressed by means of independent horizontal and vertical placements. As the objective is differentiable, classic minimization methods can be used to solve the resulting non-linear equation system. In this work, we experiment with two well-known solvers: conjugate gradient (CG) and successive over-relaxation (SOR). In both cases, a typical "global placement" tends to concentrate the blocks at the center of the FPGA. Consequently, a second step is required to legalize the placement; that is, to ensure that all block coordinates are integers and that each location on the FPGA is occupied by at most one block. To deal with this problem, we employ a bisection method that seeks to recursively divide the circuit into partitions, until each partition contains a single block.

The legalization process begins by vertically splitting the FPGA into two partitions: left and right. The location of the vertical partition is chosen, so that an equal number of blocks appear in both partitions, with blocks with smaller x coordinates appearing in the left partition and blocks with larger x coordinates appearing in the right partition. Table 6. Effect of varying \(\phi\) on wire-length estimate and number of successful routings.

<table>
<thead>
<tr>
<th>(\phi)</th>
<th>SR</th>
<th>Wire</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>81</td>
<td>865799</td>
</tr>
<tr>
<td>0.6</td>
<td>83</td>
<td>865667</td>
</tr>
<tr>
<td>0.7</td>
<td>87</td>
<td>871859</td>
</tr>
<tr>
<td>0.8</td>
<td>90</td>
<td>874087</td>
</tr>
<tr>
<td>0.9</td>
<td>92</td>
<td>879111</td>
</tr>
<tr>
<td>1</td>
<td>95</td>
<td>880271</td>
</tr>
<tr>
<td>1.1</td>
<td>95</td>
<td>880465</td>
</tr>
<tr>
<td>1.2</td>
<td>93</td>
<td>881230</td>
</tr>
<tr>
<td>1.3</td>
<td>91</td>
<td>877857</td>
</tr>
<tr>
<td>1.4</td>
<td>90</td>
<td>878011</td>
</tr>
<tr>
<td>1.5</td>
<td>86</td>
<td>882427</td>
</tr>
</tbody>
</table>
appearing in the right partition. The two partitions (left and right) are then divided horizontally into two sub-partitions by placing half of the blocks with smaller y coordinates in the bottom sub-partition and the remaining blocks in the upper sub-partition. This type of bi-partitioning is alternately performed in the vertical and horizontal directions until the FPGA is divided into partitions, where each partition contains a single block. Once a partition contains a single block, the x and y coordinates of the block are assigned the integer values closest to the coordinates of the center of the partition. The legalization method used in our work is similar to that proposed in [31].

In practice, the CG solver does not consider the physical positions of the CLBs. When CG converges, the blocks will move towards the center of the FPGA. The blocks will not be assigned to CLBs and will often overlap. Therefore, it is crucial to perform the legalization step after each iteration of CG. Another objective of the legalization process applied (i.e., bi-partitioning) is to help the solvers (CG/SOR) converge gradually towards a feasible high-quality placement. The number of iterations performed by the CG solver tend to decrease as the bi-partitioner is called to enforce legality in the system.

3.1. Conjugate gradient solver

First, we use the well-known conjugate gradient method to optimize the objective function described in Eq. (2). The CG method is a fundamental algorithmic technique for minimizing unconstrained non-linear functions. The method has the basic form shown in Algorithm 3, where $\nabla f(x_i)$ denotes the gradient, $r_i$ is the residual, $x_i$ is the step, and $d_i$ is the search direction. The loop calculates the next value of $x$ (i.e., estimated solution), the residual $r$, and the search direction $d$. Each iteration yields a better x by “walking downhill” in the conjugate direction given by the vector $d$. The algorithm terminates either after a fixed number of iterations has been reached or once a convergence test is satisfied.

Algorithm 3. Non-linear conjugate gradient

1. $d_0 = r_0 = -\nabla f(x_0)$
2. while termination criteria not met do
3. $x_i = x_{i-1} + a_i d_i$
4. $r_i = r_{i-1} - \beta_i a_i d_i$
5. $r_i = r_{i-1}$
6. $\beta_i = \frac{r_i^T r_i}{r_{i-1}^T r_{i-1}}$
7. $d_{i+1} = r_{i+1} + \beta_i d_i$
8. end while

Clearly, in order to implement CG as described in Algorithm 3 it is necessary to calculate the gradient which is used in steps 1 and 5. Starting with Eq. (2) for the star+ model (described in Section 2), we define

$$S_i = \sum_{v_l \in Net_i} (x_l - x_{c_l})^2 + \phi$$ (8)

We have

$$f(x) = \gamma \sum_i S_i$$ (9)

To calculate $\frac{\partial f(x)}{\partial x_j}$ we calculate $\frac{\partial S_i}{\partial x_j}$ first. From the definition of $S_i$, we have

$$S_i = \sum_{v_l \in Net_i} (x_l - x_{c_l})^2 + \phi$$
$$= \sum_{v_l \in Net_i} (x_l^2 - 2x_l x_{c_l} + x_{c_l}^2) + \phi$$
$$= \sum_{v_l \in Net_i} x_l^2 - 2x_l x_{c_l} + \sum_{v_l \in Net_i} x_{c_l}^2 + \phi$$
$$= \sum_{v_l \in Net_i} x_l^2 - 2x_l x_{c_l} + x_{c_l}^2 \sum_{v_l \in Net_i} 1 + \phi$$
$$= \sum_{v_l \in Net_i} x_l^2 - 2x_l x_{c_l} + x_{c_l}^2 + \phi$$
$$= \sum_{v_l \in Net_i} x_l^2 - 2x_l x_{c_l} + x_{c_l}^2 + \phi$$

(10)

If $j \notin Net_i$, it is obvious that $\frac{\partial S_i}{\partial x_j} = 0$. If $j \in Net_i$, the partial derivative of $S_i$ with respect to $x_j$ is

$$\frac{\partial S_i}{\partial x_j} = \frac{\partial}{\partial x_j} \left( \sum_{v_l \in Net_i} x_l^2 - 2x_l x_{c_l} + x_{c_l}^2 + \phi \right)$$
$$= \frac{1}{2} \sum_{v_l \in Net_i} \frac{\partial}{\partial x_j} \left( x_l^2 - 2x_l x_{c_l} + x_{c_l}^2 + \phi \right)$$
$$= \frac{1}{2} \sum_{v_l \in Net_i} \left( \frac{\partial}{\partial x_j} x_l^2 - \frac{\partial}{\partial x_j} 2x_l x_{c_l} + \frac{\partial}{\partial x_j} x_{c_l}^2 + \frac{\partial}{\partial x_j} \phi \right)$$
$$= \frac{1}{2} \sum_{v_l \in Net_i} \left( \frac{\partial}{\partial x_j} x_l^2 - 2x_l \frac{\partial}{\partial x_j} x_{c_l} + x_{c_l} \frac{\partial}{\partial x_j} x_l^2 + \frac{\partial}{\partial x_j} \phi \right)$$
$$= \frac{1}{2} \sum_{v_l \in Net_i} \left( x_l \frac{\partial}{\partial x_j} x_l^2 + 2x_l \frac{\partial}{\partial x_j} x_{c_l} - 2x_l x_{c_l} \frac{\partial}{\partial x_j} x_l^2 + \frac{\partial}{\partial x_j} \phi \right)$$
$$= \frac{1}{2} \sum_{v_l \in Net_i} \left( x_l \frac{\partial}{\partial x_j} x_l^2 + 2x_l \frac{\partial}{\partial x_j} x_{c_l} - 2x_l x_{c_l} \frac{\partial}{\partial x_j} x_l^2 + \frac{\partial}{\partial x_j} \phi \right)$$
$$= \frac{1}{2} \sum_{v_l \in Net_i} \left( x_l \frac{\partial}{\partial x_j} x_l^2 + 2x_l \frac{\partial}{\partial x_j} x_{c_l} - 2x_l x_{c_l} \frac{\partial}{\partial x_j} x_l^2 + \frac{\partial}{\partial x_j} \phi \right)$$
$$= \frac{1}{2} \sum_{v_l \in Net_i} \left( x_l \frac{\partial}{\partial x_j} x_l^2 + 2x_l \frac{\partial}{\partial x_j} x_{c_l} - 2x_l x_{c_l} \frac{\partial}{\partial x_j} x_l^2 + \frac{\partial}{\partial x_j} \phi \right)$$

(11)

Because $\frac{\partial \phi}{\partial x_j} = 0$, the above equation can be simplified as follows:

$$\frac{\partial S_i}{\partial x_j} = \frac{1}{2} \sum_{v_l \in Net_i} \left( x_l (0) + 2x_l (1) - 2k x_l \frac{1}{x_j} \right)$$
$$= \frac{1}{2} \sum_{v_l \in Net_i} \left( x_l (0) + 2x_l (1) - 2k x_l \frac{1}{x_j} \right)$$
$$= \frac{1}{2} \sum_{v_l \in Net_i} \left( x_l (0) + 2x_l (1) - 2k x_l \frac{1}{x_j} \right)$$
$$= \frac{1}{2} \sum_{v_l \in Net_i} \left( x_l (0) + 2x_l (1) - 2k x_l \frac{1}{x_j} \right)$$

(12)

In general,

$$\frac{\partial S_i}{\partial x_j} = \begin{cases} \frac{x_l - x_{c_l}}{S_i} & \text{if } j \in Net_i \\ 0 & \text{if } j \notin Net_i \end{cases}$$

(13)

Based on Eqs. (9) and (13), the partial derivative of $f(x)$ with respect to $x_j$ is

$$\frac{\partial}{\partial x_j} f(x) = \gamma \left( \sum_i \frac{\partial S_i}{\partial x_j} \right)$$

(14)

$$= \gamma \left( \sum_{v_l \in Net_i} \frac{\partial S_i}{\partial x_j} + \sum_{v_l \in Net_i} \frac{\partial S_i}{\partial x_j} \right)$$

(15)

$$= \gamma \left( \sum_{v_l \in Net_i} \frac{x_l - x_{c_l}}{S_i} + 0 \right)$$

(16)

(From Eq. 13)
\[
\frac{\partial f(x)}{\partial x_j} = \frac{\gamma}{\sum_{l \in \text{Net}_I} x_l - x_j} \frac{\partial}{\partial x_j} \left( \sum_{l \in \text{Net}_I} x_l - x_j \right)
\]

(14)

Note that these values are the \(j\)th components of vector \(f'(x)\), the gradient of \(f(x)\).

To summarize, the first-order partial derivative can be computed as follows:

\[
\nabla f(x) = \frac{\partial f(x)}{\partial x_j} = \frac{\gamma}{\sum_{l \in \text{Net}_I} x_l - x_j} \frac{\partial}{\partial x_j} \left( \sum_{l \in \text{Net}_I} x_l - x_j \right)
\]

(15)

Similarly, it is necessary to calculate the Hessian which is used in step 3. Starting with previous equation for the gradient, the second-order partial derivative for the element located on the \(i\)th row and \(j\)th column can be calculated as follows:

\[
\frac{\partial^2 f(x)}{\partial x_i \partial x_j} = \frac{\gamma}{\sum_{l \in \text{Net}_I} x_l - x_i} \frac{\partial}{\partial x_j} \left( \sum_{l \in \text{Net}_I} x_l - x_i \right)
\]

(16)

In general, the computation of the Hessian is \(O(n^2)\). Therefore, in our conjugate-gradient-based solver we seek to avoid having to compute the Hessian matrix by denormalizing in step 3 (i.e., \(d^2 \nabla^2 f(x)\)) in \(O(n)\) time. This is done as described next.

The direction vector \(d = (d_1, d_2, \ldots, d_n)\). The production \(d^T \nabla^2 f(x)d\) is a single real number. By the definition of the production of a vector and a matrix, we have

\[
d^T \nabla^2 f(x)d = \sum_{j=1}^{n} \sum_{k=1}^{n} d_j \frac{\partial^2 f(x)}{\partial x_i \partial x_j} d_k
\]

(17)

\[
d^T \nabla^2 f(x)d = \gamma \sum_{j=1}^{n} \sum_{k=1}^{n} \sum_{l \in \text{Net}_I} x_l - x_j \frac{\partial^2 f(x)}{\partial x_i \partial x_j} d_k
\]

(18)

Now we know that \(\sum_{l \in \text{Net}_I} x_l - x_j\) and \(\sum_{l \in \text{Net}_I} \frac{\partial^2 f(x)}{\partial x_i \partial x_j} d_k\) are independent on \(j\). For different \(d_j\) and \(\frac{\partial^2 f(x)}{\partial x_i \partial x_j} d_k\), their values are the same. We also have \(\sum_{l \in \text{Net}_I} x_l - x_j\) and \(\sum_{l \in \text{Net}_I} \frac{\partial^2 f(x)}{\partial x_i \partial x_j} d_k\). Therefore, Eq. (18) can be rewritten as

\[
d^T \nabla^2 f(x)d = \gamma \sum_{j=1}^{n} \sum_{k=1}^{n} d_j \frac{\partial^2 f(x)}{\partial x_i \partial x_j} d_k
\]

(19)

Although using Eqs. 17 and 19 will give the same result, the time complexities to calculate these two equations are significantly different. The complexity of Eq. (17) is \(O(n^3)\), as the Hessian matrix has \(n^2\) elements. In contrast, the complexity of Eq. (19) is \(O(n)\) as described below.

We know that computing \(x_{cl}^i\) and \(S_i\) is \(O(k)\); that is, linear to the cardinality of \(\text{Net}_I\). If \(x_{cl}^i\) and \(S_i\) are known, calculating \(\frac{\partial f}{\partial x_j}\) is \(O(1)\). In Eq. (19), all of the calculations contained within the curly brackets are \(O(k)\). Therefore, the time complexity of Eq. (19) is linear to the sum of the cardinalities of all nets. As the sum of the cardinalities of all the nets equals the sum of the fan in/outs of all the blocks, the complexity is also linear to the sum of the fan in/outs of all the blocks. Due to the physical limit of any FPGA architecture, each block can only connect to a certain number of nets. That means that the sum of the fan in/outs of all the blocks is \(O(n)\) (where \(n\) is the number of blocks). As a result, the time complexity of Eq. (19) is \(O(n)\).

3.2. Successive over-relaxation solver

Conjugate gradient methods are among the simplest and most elegant iterative techniques for performing non-linear optimization. However, regardless of the implementation, the performance of the search can be affected by a loss of conjugacy that arises from the use of non-quadratic terms. In this section, we describe a second solver based on successive over-relaxation (SOR). The SOR technique is an iterative method devised to speed up the well-known Gauss–Seidel method. Given a system of linear equations of the form \(Ax = b\), we can rewrite it as \(L + D + U\), where \(L, D, \) and \(U\) denote the strictly lower, diagonal, and strictly upper triangular sub-matrices of \(A\). The successive over-relaxation iteration is then defined by the following recurrence relation:

\[
x^{k+1} = (D + (\omega - 1)L)^{-1} (\omega b - (D - \omega L) x^k)
\]

(20)

where \(\omega\) is a relaxation factor. If matrix \(A\) is symmetric and positive-definite, the SOR iteration always converges when \(0 < \omega < 2\). However, it is not possible to estimate a priori the value of \(\omega\) that maximizes the rate of convergence. When \(\omega = 1\), the SOR iteration reduces to the Gauss–Seidel method and, therefore, like the Gauss–Seidel method, the computation of SOR can be performed in place. The actual implementation of SOR uses the following element-based iteration formula:

\[
x_i^{k+1} = (1 - \omega) x_i^{k} + \frac{\omega}{d_i} \left( b_i - \sum_{j \neq i} a_{ij} x_j^{k} - \sum_{f=1}^{n} d_f y_f^k \right)
\]

(21)

where \(i = 1, 2, \ldots, n\).
Clearly, to implement SOR based on Eq. (2) it is necessary to find the recurrence relation between \(x_i^{(k+1)}\) and \(x_i^{(k)}\). For the star+ model, recall that the partial derivative of \(f(x)\) with respect to \(x_j\) is \((\partial f/\partial x_j)(x) = \gamma \sum_{\forall j \in \text{Net}_i} (x_j - x_{cl_j})/S_j\). By making \((\partial f/\partial x_j)(x) = 0\), we obtain the equation:
\[
\gamma \sum_{\forall j \in \text{Net}_i} x_j - x_{cl_j} - \sum_{\forall j \in \text{Net}_i} x_{cl_j} = 0
\]
\[
\sum_{\forall j \in \text{Net}_i} x_j - \sum_{\forall j \in \text{Net}_i} x_{cl_j} = 0
\]
\[
x_j \sum_{\forall j \in \text{Net}_i} \sum_{\forall j \in \text{Net}_i} x_{cl_j} = \sum_{\forall j \in \text{Net}_i} x_{cl_j} \sum_{\forall j \in \text{Net}_i} x_{cl_j}
\]
\[
x_j = \frac{1}{\sum_{\forall j \in \text{Net}_i} x_{cl_j}} \sum_{\forall j \in \text{Net}_i} x_{cl_j}
\]
In the previous equation, \(x_{cl_j} = (1/k) \sum_{\forall j \in \text{Net}_i} x_{cl_j} = \sum_{\forall j \in \text{Net}_i} x_{cl_j} - kx_{cl_j} + 1\). Now, by putting the iteration number into these equations we obtain the Jacobi iteration for the placement problem:
\[
x_i^{(k+1)} = \frac{1}{k} \sum_{\forall j \in \text{Net}_i} x_j^{(k)}
\]
\[
S_j^{(k+1)} = \sqrt{\sum_{\forall j \in \text{Net}_i} (x_j^{(k)})^2 - k(x_j^{(k)})^2 + 1}
\]
\[
x_i^{(k+1)} = \frac{1}{k} \sum_{\forall j \in \text{Net}_i} x_j^{(k)}
\]
To implement the Gauss–Seidel iteration, we need to update \(x_{cl_j}\) and \(S_j\) immediately after \(x_j\) moves from \(x_i^{(k)}\) to \(x_i^{(k+1)}\). Fortunately, both \(x_{cl_j}\) and \(S_j\) can be updated in a constant time (see Section 2). This feature makes it possible to build a time-efficient SOR method for FPGA placement based on the star+ model. For the sake of simplicity, we introduce two new variables \(U_i\) and \(V_i\) and let \(U_i = \sum_{\forall j \in \text{Net}_i} x_j^2\) and \(V_i = \sum_{\forall j \in \text{Net}_i} x_j\). The SOR iteration for the placement problem can now be defined as
\[
U_i^{(0)} = \sum_{\forall j \in \text{Net}_i} (x_j^{(0)})^2
\]
\[
V_i^{(0)} = \sum_{\forall j \in \text{Net}_i} x_j^{(0)}
\]
\[
x_i^{(0)} = \frac{1}{k} V_i^{(0)}
\]
3.3. I/O pad pre-placement

To avoid trivial solutions, analytic placement methods require that some blocks must be assigned locations on the FPGA fabric prior to optimization. A simple way to achieve this is to assign blocks randomly. However, this does not always result in the best solution. In [32] we presented an interesting algorithm, with a runtime complexity of \(O(|E|\log|V|)\), for pre-assigning I/O blocks to I/O pads around the perimeter of the FPGA. Results obtained show that an average reduction in wire-length of 1.5% can be achieved over a random pre-placement strategy, and that the ability of the algorithm to help reduce wire-length typically improves as the number of I/O pads relative to logic blocks increases.

In this section, we briefly describe shrubPlace, that temporarily assigns I/O blocks to I/O pads around the perimeter of the FPGA prior to placing logic blocks. This pre-placement is not the final placement of I/O pads, but helps with the placement of logic blocks later on as some of the logic blocks share connections with the various I/O blocks.

The goal of shrubPlace is to place the I/O pads in such a way that those with higher connectivity are placed closer together than the I/O pads with lower connectivity. Also, this pre-placement provisionally locates components on the periphery of the FPGA, which causes other components to be distributed throughout the chip when solving the linear equation system.

Fig. 1a shows a simple example with three I/O pads (gray shading) L, M and N, three logic blocks (no shading) a, b, and c, and five nets connecting various blocks. Net 1 connects I/O pad L and logic block a; Net 2 connects blocks a and b; Net 3 connects logic block b to I/O pad M; Net 4 connects logic blocks b and c; and, Net 5 connects I/O pad N to logic block c. Observe that the connection between I/O pads L and M involves three nets: Net 1 (L–a), Net 2 (a–b), and Net 3 (b–M), while the connection between I/O pads L and N involves four nets: Net 1 (L–a), Net 2 (a–b), Net 4 (b–c), and Net 5 (N–c).

![Fig. 1. Effect of pre-placement of I/O blocks: (a) poor pre-placement and (b) better placement.](image-url)
In this case, the quality of placement in Fig. 1a is poor. This is partially caused by I/O pad M being placed too far away from I/O pads L and N. Moving logic block b towards logic blocks a and c improves the placement, but this movement increases the amount of wire needed to connect b and M. However, if we move I/O pad M beside I/O pads L and N, as shown in Fig. 1b, we will use the least amount of wire to connect the blocks. This is what the I/O pad pre-placement technique employed here intends to do.

4. Results

Using the MCNC benchmarks [30] we compared StarPlace (using both the CG and SOR solvers) to VPR. Note that all of the algorithms were implemented in C using the GNU GCC compiler, and all experiments were conducted on the same machine—a single core HP 2100 workstation with 1 Gigabyte of RAM running Windows XP. To better illustrate the trade-off between the meta-heuristic methods and analytic methods proposed in this paper, VPR was run in fast mode (i.e., with inner_num = 1) first using HPWL then using star+. Following placement, each circuit was routed and information collected regarding the final wire-length and critical-path delay.

4.1. VPR versus analytic placers

Table 7 summarizes our results and shows a comparison of VPR with StarPlace. The final critical-path delay values are in nanoseconds, the final wire-length measured is in terms of the number of wire segments required to route each placement, and the runtime\(^1\) required to perform placement is in seconds. The final row of the table shows the average percentage increase/decrease in critical-path delay, wire-length, and runtime compared with VPR (using HPWL). These averages are taken with respect to all circuits. However, since VPR is stochastic, when reporting critical-path delay, wire-length and runtime for VPR, each circuit is placed-and-routed 10 times and the average reported in the table columns (see Section 2 for more details).

The previous results can be summarized as follows. When the star+ model is used as the placement objective and optimized using either simulated annealing (i.e., VPR) or successive over-relaxation very good results can be achieved with respect to solution quality. Using star+, both optimization methods are able to reduce critical-path delay by approximately 8–9% on average compared with VPR. With regards to wire-length, simulated annealing-based VPR holds a slight edge and is able to reduce wire-length by 2.4% on average. Successive over-relaxation, on the other hand, results in a slight 1.1% increase in wire-length compared with VPR. It should be noted, however, that minimizing critical-path delay is typically considered the more important objective as critical-path delay directly affects the speed of the final circuit. Most importantly, the analytic placer based on successive over-relaxation runs approximately 78% faster than VPR. This represents a 4.8 × improvement in speed. Overall, the previous observations reveal that the successive over-relaxation method described here is able to run faster than simulated annealing-based methods, like VPR, while producing competitive, if not superior, quality solutions.

Although the conjugate-gradient-based placement managed to result in a small –1.8% decrease in critical-path delay, the wire-length and runtime values were not found to be competitive with those of VPR or those found using successive over-relaxation. This discrepancy between the performance of successive over-relaxation and conjugate-gradient is likely attributable to the propensity of conjugate gradient to loose conjugacy when a non-quadratic objective is employed.

4.2. Convergence of successive over-relaxation

We now take a closer look at the convergence properties of successive over-relaxation-based placement. In practice, the relaxation factor (\(\omega\)) affects the convergence properties of SOR. For example, if \(\omega = 1\), the SOR method simplifies to the Gauss–Seidel method. Moreover, SOR does not guarantee

---

\(^1\) Run-times include time to perform pre-placement, as described in Section 3.3.
convergence if $\omega$ is not between 0 and 2. Typically, values of $\omega > 1$ are used to speedup convergence, while values of $\omega < 1$ are often used to establish convergence of a diverging iterative process. Generally, it is not possible to compute a priori the value of $\omega$ that is optimal with respect to the rate of convergence of SOR. Even when it is possible to compute the optimal value of $\omega$, the expense of such computation is usually prohibitive. Therefore, in this subsection, we perform a series of experiments to explore the effect of $\omega$ on the convergence of our SOR-based placement method.

In the experiments that follow, SOR is run on all 20 MCNC benchmarks, while the value of $\omega$ is varied from 0.2 to 2 with an incremental step of 0.2. The experimental results are summarized in Fig. 2. The x-axis shows the value of $\omega$, while the y-axis gives the aggregate wire-length for all 20 benchmarks. Clearly, as $\omega$ increases the total number of estimated wire segments decreases. In particular, when $\omega = 2$ the total estimated number of wire segments is smallest (942 435). Based upon these results, we chose to set $\omega = 2$ in our implementation. However, it is worthwhile noting that using the smallest $\omega$ value ($\omega = 0.2$) results in a wire-length estimate approximately 1% worse than when using $\omega = 2$ (952 746 versus 942 435). Thus, the proposed SOR method is quite stable and robust, regardless of what value of $\omega$ is used.

Fig. 3 shows the convergence properties of SOR compared with VPR for the CLMA benchmark (the largest benchmark among MCNC 20). In Fig. 3, the x-axis represents the time (in seconds), while the y-axis represents the number of wire segments. The blue solid line is the convergence curve of VPR with inner_num = 1, while the red dashed line is the convergence curve of SOR. Clearly, SOR converges much faster than VPR to a high-quality solution. (Note: Although Fig. 3 is only for a specific benchmark, the behavior it illustrates is typical of that found when using the other benchmarks.)

A close look at Fig. 3 shows that convergence curve of VPR is divided into three phases. The first phase (roughly from the 0th to the 11th second in this case) is the period that VPR (based on simulated annealing) is running at high temperature. During this period, the quality of the placement is improved very slowly since a large portion of non-improvement moves are accepted. During the second phase (from the 11th to the 25th second in this case), most of the improvement in solution quality is made as the temperature goes lower and VPR accepts fewer non-improving moves. During the third phase (after the 25th second in this case), solution quality only improves slightly, since only a few improving moves can be found due to the low temperature of simulated annealing.

In contrast, the convergence curve for SOR has only two phases. SOR improves the quality drastically during the first phase (from the 0th to the 6th second in this case). During the second phase (after the 6th second in this case), the solution quality is still improved but very slowly, and finally the curve flattens out where little further improvement is made. The ability to improve quality at an early stage in the search is essential when a fast solution is required. In this respect, SOR may be more attractive to users, since VPR first spends two thirds of its runtime at high and low temperatures where solution quality improves.
slowly. SOR, on the other hand, rapidly improves the quality finding a high-quality solution in small amounts of actual runtime.

4.3. ShrubPlace results

To determine the overall effectiveness of shrubPlace, we integrated the pre-placement algorithm into StarPlace (based on SOR), then ran StarPlace into two different ways. The first involved randomly pre-placing the I/O blocks around the perimeter of the FPGA then running StarPlace. The second involved running shrubPlace to determine the initial positions of the I/O pads followed by StarPlace. Both configurations were evaluated using the standard MCNC benchmarks.

The results are shown in Table 8. Column 1 identifies the benchmark by name. Column 2 shows the improvement in total wire-length (after routing) achieved by StarPlace when starting with a pre-placement generated by shrubPlace versus a randomly generated pre-placement. (Note: for each benchmark, 10 random pre-placements were performed. The average total wire-length was then used when comparing to the total wire-length found when using shrubPlace. Recall that shrubPlace is deterministic and, therefore, need only be executed once.)

The results in Table 8 show that shrubPlace outperformed traditional random pre-placement in 14 out of 20 cases (70%). In the best case, shrubPlace outperformed the random placement strategy by almost 8%, while the performance of shrubPlace over all 20 benchmarks is approximately 1.5% better than the random pre-placement strategy. It must be stressed that FPGA placers, like StarPlace, must work extremely hard to achieve even a 1% reduction in total wire-length. Moreover, a 1% reduction in wire-length has a significant affect on performance, power consumption, heat distribution and, to a great extent, routability. Therefore, the ability of shrubPlace to further reduce wire-length by 1.5% is significant, especially given its runtime performance. Column 3 in Table 8 shows that shrubPlace only requires (approximately) 5% of the total runtime, on average, when used in conjunction with StarPlace. In contrast, the pre-placements produced using a random pre-placement strategy may be poor, requiring a “pool” of pre-placements (e.g., 10) to be created and evaluated. Even if it is assumed that the time to create each random pre-placement is instantaneous, the time required to run StarPlace multiple times, versus once with shrubPlace, is significant.

Table 9 shows that the ratio of I/O blocks to logic blocks has a direct impact on the ability of shrubPlace to improve wire-length solutions. When the ratio of I/O blocks to logic blocks is small (less than 2%) the average wire-length improvement is negative (i.e., −0.34%). This suggests that when there are relatively few I/O pads to pre-place, a random strategy will typically fare better. However, as the ratio of I/O blocks increases (greater than 2%) the wire-length improvement is always positive, with average wire-length improvements of nearly 3% being possible.

5. Conclusion and future work

Analytic placement methods have been used with great success for ASIC designs. However, despite their popularity in the ASIC community, relatively few analytic placement methods exist for FPGAs. This paper begins by presenting a new model, called star+, for estimating the wire-length of a net. The star+ model is based on the traditional star model and is continuously differentiable, thus making star+ suitable for use with analytic placement methods. A constant-time method for star+ is developed for calculating the incremental changes in cost that incur by moving/swapping blocks, regardless of the size of the net. The effectiveness of the star+ model is first demonstrated by implementing star+ into the VPR framework and comparing with the well-known HPWL wire-length model. Results show that compared with HPWL, star+ is able to obtain an 8.3% reduction in critical-path delay and a 2.4% reduction in wire-length, while requiring only 1.5% more runtime. Two analytic placers, both based on star+, are also presented. The first is based on conjugate gradient, and employs a novel linear-time calculation on the inner step. The second is based on successive over-relaxation and achieves an 8.7% reduction in critical-path delay, while running 78% faster than VPR.

Although this study sheds some light on the suitability of star+ in the context of analytic placement, further research can be done on a number of issues. First, confirmation on the scalability of star+ ought to be studied with larger, academic FPGA circuits. Second, the role of the adjustable parameters γ and ϕ should be studied in the context of other, larger circuits. Finally, more complex, but effective legalization procedures should be experimented with and evaluated.

Table 9

<table>
<thead>
<tr>
<th>Ratio of I/O blocks to logic blocks (%)</th>
<th>Wirelength improvement (%) using shrubPlace</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 2</td>
<td>−0.34</td>
</tr>
<tr>
<td>2 and &lt; 5</td>
<td>+1.38</td>
</tr>
<tr>
<td>5 and &lt; 10</td>
<td>+2.95</td>
</tr>
<tr>
<td>&gt; 10</td>
<td>+2.86</td>
</tr>
</tbody>
</table>

References


Near linear wirelength


M. Xu et al. / INTEGRATION, the VLSI journal 44 (2011) 192–204

S. Areibi

Grewal has received the B.Sc. degree (Hons) in Computer Science from Brock University in 1990, the M.Sc. and Ph.D. degrees in Computer Science from the University of Guelph in 1992, and the Ph.D. degree in Engineering from the University of Guelph in 1998. He is currently an Adjunct Professor at the School of Computer Science, University of Guelph. Since 2007 he has worked as a member of the BlackBerry-Infrastructure Group at Research in Motion, Waterloo, Canada. His current research interests include discrete optimization, CAD tools/algorithms for FPGAs, and wireless computing.

G. Grewal received the B.Eng. degree in Computer Science from Brock University in 1990, the M.Sc. degree in Computer Science from the University of Guelph in 1992, and the Ph.D. degree in Engineering from the University of Guelph in 1998. He is currently an Associate Professor at the School of Computer Science at the University of Guelph. Dr. Grewal has published more than 60 technical papers in international journals and conferences in the areas of electronic design automation, reconﬁgurable computing, combinatorial optimization, and parallel computing. He has also won several best paper awards, and continues to serve on the program committees for various international conferences.

S. Areibi received the B.Sc. degree in Computer Engineering from El-Fateh University, Libya in 1984, and the M.A.Sc. and Ph.D. degrees in Electrical/Computer Engineering from the University of Waterloo, Ontario, Canada in 1991 and 1995, respectively. After completion of his Ph.D. degree he spent two years from 1995 until 1997 as a Research Mathematician with Shell International Oil Products in The Netherlands. From 1997 until 1999 he was a faculty member in the Electrical and Computer Engineering Department at Ryerson Polytechnic University, Toronto, Canada. Currently, he is an Associate Professor at the University of Guelph in the School of Engineering “Engineering Systems and Computing”. His research interests include VLSI physical design automation, combinatorial optimization, reconﬁgurable computing systems, embedded systems and parallel Processing. Dr. Areibi has authored/coauthored over 80 papers in international journals and conferences. He served on the technical program committees for several international conferences on computer engineering and embedded systems. Dr. Areibi was the Associate Editor of the International Journal of Computers and Their Applications. He also served as a member of the program committee for GECCO, HPC and several other IEEE conferences.

M. Xu received the B.Eng. degree in Computer Science and Engineering from Xi’an Jiaotong University, China in 1990, and the M.Sc. and Ph.D. degrees in Computer Science from the University of Guelph, Canada in 2004 and 2009, respectively. He is currently an Adjunct Professor at the School of Computer Science, University of Guelph. Since 2007 he has worked as a member of the BlackBerry-Infrastructure Group at Research in Motion, Waterloo, Canada. His current research interests include discrete optimization, CAD tools/algorithms for FPGAs, and wireless computing.

S. Areibi

Grewal has received the B.Sc. degree (Hons) in Computer Science from Brock University in 1990, the M.Sc. degree in Computer Science from the University of Guelph in 1992, and the Ph.D. degree in Engineering from the University of Guelph in 1998. He is currently an Adjunct Professor at the School of Computer Science, University of Guelph. Since 2007 he has worked as a member of the BlackBerry-Infrastructure Group at Research in Motion, Waterloo, Canada. His current research interests include discrete optimization, CAD tools/algorithms for FPGAs, and wireless computing.

G. Grewal received the B.Sc. degree (Hons) in Computer Science from Brock University in 1990, the M.Sc. degree in Computer Science from the University of Guelph in 1992, and the Ph.D. degree in Engineering from the University of Guelph in 1998. He is currently an Associate Professor at the School of Computer Science at the University of Guelph. Dr. Grewal has published more than 60 technical papers in international journals and conferences in the areas of electronic design automation, reconﬁgurable computing, combinatorial optimization, and parallel computing. He has also won several best paper awards, and continues to serve on the program committees for various international conferences.

S. Areibi received the B.Sc. degree in Computer Engineering from El-Fateh University, Libya in 1984, and the M.A.Sc. and Ph.D. degrees in Electrical/Computer Engineering from the University of Waterloo, Ontario, Canada in 1991 and 1995, respectively. After completion of his Ph.D. degree he spent two years from 1995 until 1997 as a Research Mathematician with Shell International Oil Products in The Netherlands. From 1997 until 1999 he was a faculty member in the Electrical and Computer Engineering Department at Ryerson Polytechnic University, Toronto, Canada. Currently, he is an Associate Professor at the University of Guelph in the School of Engineering “Engineering Systems and Computing”. His research interests include VLSI physical design automation, combinatorial optimization, reconﬁgurable computing systems, embedded systems and parallel Processing. Dr. Areibi has authored/coauthored over 80 papers in international journals and conferences. He served on the technical program committees for several international conferences on computer engineering and embedded systems. Dr. Areibi was the Associate Editor of the International Journal of Computers and Their Applications. He also served as a member of the program committee for GECCO, HPC and several other IEEE conferences.

M. Xu received the B.Eng. degree in Computer Science and Engineering from Xi’an Jiaotong University, China in 1990, and the M.Sc. and Ph.D. degrees in Computer Science from the University of Guelph, Canada in 2004 and 2009, respectively. He is currently an Adjunct Professor at the School of Computer Science, University of Guelph. Since 2007 he has worked as a member of the BlackBerry-Infrastructure Group at Research in Motion, Waterloo, Canada. His current research interests include discrete optimization, CAD tools/algorithms for FPGAs, and wireless computing.

G. Grewal received the B.Sc. degree (Hons) in Computer Science from Brock University in 1990, the M.Sc. degree in Computer Science from the University of Guelph in 1992, and the Ph.D. degree in Engineering from the University of Guelph in 1998. He is currently an Adjunct Professor at the School of Computer Science at the University of Guelph. Dr. Grewal has published more than 60 technical papers in international journals and conferences in the areas of electronic design automation, reconﬁgurable computing, combinatorial optimization, and parallel computing. He has also won several best paper awards, and continues to serve on the program committees for various international conferences.

S. Areibi received the B.Sc. degree in Computer Engineering from El-Fateh University, Libya in 1984, and the M.A.Sc. and Ph.D. degrees in Electrical/Computer Engineering from the University of Waterloo, Ontario, Canada in 1991 and 1995, respectively. After completion of his Ph.D. degree he spent two years from 1995 until 1997 as a Research Mathematician with Shell International Oil Products in The Netherlands. From 1997 until 1999 he was a faculty member in the Electrical and Computer Engineering Department at Ryerson Polytechnic University, Toronto, Canada. Currently, he is an Associate Professor at the University of Guelph in the School of Engineering “Engineering Systems and Computing”. His research interests include VLSI physical design automation, combinatorial optimization, reconﬁgurable computing systems, embedded systems and parallel Processing. Dr. Areibi has authored/coauthored over 80 papers in international journals and conferences. He served on the technical program committees for several international conferences on computer engineering and embedded systems. Dr. Areibi was the Associate Editor of the International Journal of Computers and Their Applications. He also served as a member of the program committee for GECCO, HPC and several other IEEE conferences.