A Fast Algorithm for Identifying Good Buffer Insertion Candidate Locations

Charles J. Alpert, Milos Hrkic, and Stephen T. Quay
IBM Corp. 11400 Burnet Road, Austin, Texas 78758
University of Illinois at Chicago, 851 S. Morgan St., Chicago, Illinois 60607
alpert@us.ibm.com, mhrkic@cs.uic.edu, quayst@us.ibm.com

ABSTRACT
Van Ginneken's algorithm [18] for performing buffer insertion is a classic in the field, since it optimally solves the problem subject to a set of fixed buffer insertion candidate locations for a given Steiner topology. The generation of these candidate locations is typically performed by dividing the routed wires into small uniformly sized pieces [1]. However, certain regions of the layout are generally more attractive to place buffers than others, e.g., sparse regions are preferred to dense ones. This work presents a fast, shortest path based algorithm to identify good candidate buffer insertion locations to be passed to van Ginneken's algorithm. Our experiments show that the buffers inserted significantly improve the overall design density with virtually no impact on either CPU time or buffered net delays.

Categories and Subject Descriptors
B.7.2 [**Integrated Circuits**]: Design Aids - Placement and routing; J.6 [**Computer-aided Engineering**]: Computer-aided design.

General Terms
Algorithms, Performance, Design

Keywords
Buffer insertion, physical synthesis, global routing, planning.

1. INTRODUCTION
Interconnect's domination of system performance has made buffering a critical component of modern VLSI design methodologies. As a result, one must manage design density when inserting buffers that are used to improve performance and/or fix electrical violations. For example, in hierarchical designs, generally only limited space is allocated for buffer usage. These spaces are either the "alleys" [5] in between blocks or special purpose "buffer blocks" [8] that contain only buffers inside. Whichever scheme is used, it is imperative that one manages the density during physical synthesis. If one fills up an alley or a buffer block too early, then critical space resources may not be available when trying to buffer a critical net.

Even on flat designs, design density management is increasingly important due to the sheer number of buffers required. Cong expects that close to 800,000 repeaters will be needed for designs in the 50 nanometer node. Saxena et al. [16] argues that the number of repeaters will rise even faster, e.g., reaching about 15% of the total cell count for intrablock communications for the 65 nanometer node.

The most famous buffer insertion algorithm is van Ginneken's $O(n^2)$ dynamic programming algorithm [18], which finds the optimal slack solution for a given net and a fixed topology. Due to its flexibility, several works have extended this approach to handle more complicated objectives and constraints (e.g., [2][3][12][13][14][15]). Recently, Shi and Zhou [17] showed how one can obtain an $O(n \log n)$ implementation of the algorithm.

Van Ginneken's algorithm assumes that a set of buffer insertion candidate locations are predetermined for the given topology. The most common method for selecting insertion points is to choose them at regular intervals. Alpert et al. [1] show how the quality of results is affected by the degree of wire segmenting that is performed on the topology.

For example, Figure 1(a) shows uniform segmenting for a Steiner tree with three sinks and a single blockage. For these regions for which buffer insertion is forbidden, one simply avoids inserting buffer candidate locations on top of the blockage. In (b), one sees the same uniform segmenting scheme, but with finer spacing. The additional buffer insertion locations could potentially improve the timing for the buffered net, for additional runtime cost. This work proposes using roughly the same number of buffer insertion candidates as in uniform segmenting, but spacing them asymmetrically (c). The purpose is not to improve timing performance, but rather to bias van Ginneken's algorithm to insert buffers in regions of the design that are more favorable, such as areas with lower density.

To accomplish this buffer candidate selection, we propose a linear time and linear memory shortest path algorithm, which we call SPA. The algorithm constructs a directed acyclic graph (DAG) over the set of potential candidate locations and chooses a subset by constructing a shortest path via a topological sort [6]. For virtually no additional CPU cost and timing penalty, SPA identifies candidate locations that causes van Ginneken's algorithm to significantly improve its design density management.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ISPD'04 April 18-21, 2004, Phoenix, Arizona, USA.
Copyright 2004 ACM 1-58113-817-2/04/0004 . . . $5.00.

It should be noted that other works have looked at applying the shortest path to buffer insertion related problems. Lai and Wong [11] seek a shortest path in a tiled graph that corresponds to a buffer insertion solution. Unlike our approach, their graph is not acyclic and results in runtime that is quadratic with the size of the graph. Alpert et al. [5] proposed a linear buffer insertion algorithm which sought the least cost solution for a given length constraint and tiling. Finally, the approach of Gao and Wong [10] is most similar in spirit to SPA, though they use shortest paths in a DAG to optimize delay accuracy instead of density and speed, which results in a higher runtime. Unlike all three of these approaches, SPA candidate selection is more flexible and is better suited to multi-fanout nets since its purpose is to complement, not replace, van Ginneken's algorithm.

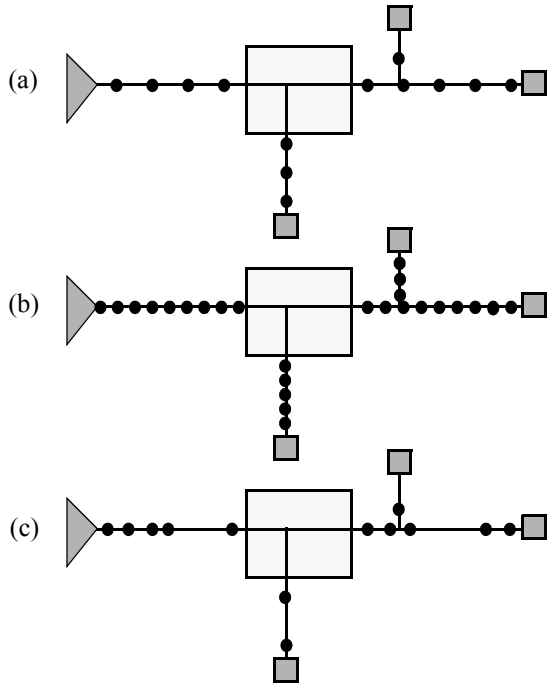

Figure 1 Given a fixed topology, one can segment wires uniformly via either (a) coarse or (b) finer spacing. We propose asymmetric segmenting (c) based on the design characteristics.

2. THE CANDIDATE IDENTIFICATION PROBLEM

Assume that the chip is divided into uniform tiles. The tiles should be fine enough so that the exact location of the buffer within the tile should not significantly affect the path's timing. Each tile t has a cost $cst(t)$ assigned to it. This cost reflects the preferences for putting buffers in certain parts of the chip. If the tile is sparsely populated, it should have low cost while a densely populated should have higher cost. Of course, the tile cost can be a function of the tile's via density, total power usage, etc.

Figure 2 shows a Steiner tree with source A and sinks B and C overlapping its tiles. The lighter areas indicate tiles of low cost, while the darker areas indicate higher cost. The problem is to pick a subset of these tiles to serve as candidate buffer insertion locations.

Let T be the set of tiles that overlap the given Steiner tree, and let $P \subseteq T$ be a set of tiles corresponding to chosen buffer insertion locations. Let $dist(t_i, t_j)$ be the number of tiles on the path from t_i to t_j in T. For example, $dist(D, C) = 9$ in Figure 2. Let L be the maximum allowable tiles between consecutive buffers. L could be determined either by a maximum allowable slew constraint or by determining the ideal buffer spacing for delay [9]. L should be such that if buffers are placed at a distance greater than L tiles away, then either an electrical violation results or performance is significantly sacrificed. The problem can be initially formulated as:

$$\text{Find } P \subseteq T \text{ that minimizes } \sum_{t \in P} cst(t) \text{ such that } dist(t_i, t_j) \le L \quad (1)$$

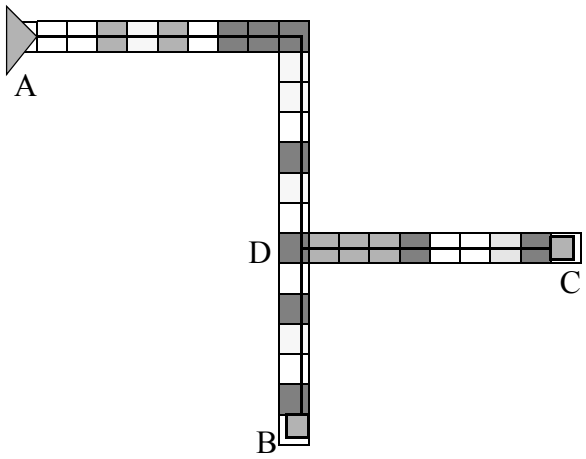

Figure 2 A Steiner tree overlapping the tiles in the layout.

One problem with solving this formulation is that it serves to minimize the total number of buffer insertion candidates. Using a spacing of distance L between candidate locations does not give van Ginneken's algorithm much flexibility to decouple branches and utilize buffers of different sizes. Empirically, one needs to space candidate locations a distance of around $L/6$ to obtain high quality solutions.

Let S be the desired number of tiles between consecutive buffer insertion candidates. S is chosen by the user to obtain the desired timing performance/CPU trade-off. For example, Figure 1(a) has a value of S that is twice that of Figure 1(b). The result is that van Ginneken's algorithm takes longer to run on the latter example, yet it may achieve a better solution.

Given a subset of tiles $P \subseteq T$, let $C(P) \subseteq P \times P$ be the set of consecutive pairs of tiles in P. If P is constructed with uniform tile spacing S, then $dist(t_i, t_j) = S$ for all $(t_i, t_j) \in C(P)$ (except for potentially next to the sinks). For asymmetric spacing, we wish to associate a penalty for spacing tiles either closer to or further from the desired spacing S. Define a function $pen(x, S, L)$ that assigns a cost penalty when the distance x between tiles is not equal to S. The complete problem formulation is now:

Buffer Candidate Selection Problem: Given a set of tiles T that overlap a given Steiner tree for a net, a maximum buffer spacing L, and a desired spacing S, find $P \subseteq T$ that minimizes

$$\sum_{t \in P} cst(t) + \sum_{(t_i, t_j) \in C(P)} pen(dist(t_i, t_j), S, L) \qquad (2)$$

such that $dist(t_i, t_j) \leq L$.

3. IMPLEMENTATION STRATEGY

For a given Steiner tree, we break the wires into sets of disjoint 2-paths. A 2-path is a route in which the endpoints of the route are either sinks, Steiner points, or a source. Further, all internal nodes of the path have degree two. For example, the tree in Figure 2 has 2-paths $A - D$, $D - B$, and $D - C$.

If one solves the problem optimally for 2-paths, that does not guarantee an optimal solution to our problem formulation. The reason it may be suboptimal is due to buffer candidate spacing near Steiner points. However, in these cases, one actually should seek candidate locations that are on branches immediately following Steiner points in order to give van Ginneken's algorithm decoupling opportunities. Hence, breaking the tree into 2-paths and solving each problem separately can yield higher quality solutions.

Given a tile t, let $d(t)$ be the *density* of tile t, i.e., the ratio of cell area inside the cell to the total available area. Let D the density of a *full* tile. Typically one would use $D = 1.00$; however, it may be preferable to leave a little wiggle room to ensure there is enough space to insert a buffer. The experiments in Section 5 use $D = 0.96$. The choice of D also depends on the tile size.

We choose $cst(t) = d(t)^2$ for $t < D$ and $cst(t) = \infty$ otherwise. Assigning infinite cost for full tiles ensures no candidate will be inserted. Using squared cost ensures the cost increases more rapidly as a tile is closer to becoming full. For example, the cost of choosing two tiles with densities of 0.1 and 0.9 is 0.82, while the cost of choosing two tiles with density of 0.5 is 0.5.

For spacing penalty, we choose the quadratic function:

$$pen(x, S, L) = \frac{(x - S)^2}{(L - S)^2} \qquad (3)$$

If the spacing x between consecutive tiles is S, then there is no penalty. The maximum penalty is one since $x \leq L$ (as long as $S \leq L/2$, which is normally necessary to achieve good solutions). The penalty increases quadratically with the distance from the desired spacing since delay also increases quadratically with length. In practice, one can also add a scaling constant to trade off between the first and second terms in Equation (2) to obtain either looser or tighter adherence to the desired spacing.

4. THE SPA ALGORITHM

Given a single 2-path, label the set of tiles it spans as $t_1, t_2, \ldots, t_n$ and construct the following directed graph $G^T(V, E)$, where $V = \{t_1, \ldots, t_n\}$. Let $E = \{(t_i, t_j)\}$ for all i and j such that $i < j$, $j - i \leq L$, $cst(t_i) \neq \infty$ and $cst(t_j) \neq \infty$. The edges in E connect tiles within the distance constraint as long as neither tile has infinite cost. Note that G^T is a DAG and that $|E| < nL$.

The set of buffer insertion candidate locations can be realized by performing a shortest path computation on G^T. The shortest path on a DAG can be efficiently computed via a topological sort [6]. Let c_i be the cost of the best solution from t_1 to t_i and let P_i be the path corresponding to that solution. The tiles visited on the path P_n correspond to the solution for the two-path T.

Figure 3 describes the SPA algorithm. In Step 1, the DAG G^T is constructed from the given path of tiles T. Step 2 initializes the cost of the solution at the first tile to zero, and its corresponding path to the empty set. Step 3 loops through the tiles via the index j and Step 4 initializes the cost of the j^{th} tile to infinity. Step 5 then loops through all edges that go from an earlier tile t_i to t_j. Step 6 computes the cost of constructing a path to t_j in which the preceding tile is t_i and stores this as tmp. Step 7 then checks whether this new cost is smaller than the best cost seen to date. If it is smaller, the cost c_j and the path P_j are updated accordingly. Finally, the path to the last tile is returned in Step 8. Note that if $c_n = \infty$, no solution exists that satisfies the constraints.

Input:	$T = \{t_1, \ldots, t_n\} \equiv$ 2-path of possible tiles
	$L \equiv$ Maximum tile spacing constraint
	$S \equiv$ Desired spacing between buffer candidates
Vars:	$c_i \equiv$ Cost of optimal solution from t_1 to t_i
	$P_i \equiv$ Set of tiles for solution from t_1 to t_i
	$tmp \equiv$ Cost of current sub-path
	$G^T(V, E) \equiv$ DAG constructed from T
Output:	$P \subseteq T \equiv$ Set of tiles for candidate locations
1. Construct $G^T(V, E)$ from T.	
2. Set $c_1 = 0$, $P_1 = \varnothing$.	
3. for $j = 2$ to n do	
4. Set $c_j = \infty$, $P_j = \varnothing$.	
5. for each edge $(t_i, t_j) \in E$ do	
6. Set $tmp = c_i + cst(t_j) + pen(j - i, S, L)$	
7. if $tmp < c_j$ then	
Set $c_j = tmp$, $P_j = P_i \cup \{t_j\}$	
8. Return P_n.	

Figure 3 SPA Algorithm description.

The algorithm runs in $O(nL)$ time which is linear time, given that L is a constant. The runtime is dominated by the $O(n)$ loop in Step 3 which calls the $O(L)$ loop in Step 5. Note that because the graph is a DAG, one does not need the more expensive Dijkstra's shortest path algorithm since a topological sort is sufficient. Further, the memory requirement is only $O(n)$ since only one cost value need be

stored for each tile. The paths do not have to be stored explicitly (as indicated in Figure 3), but may be stored implicitly by pointers back to the previous tile. Then the final path can be uncovered by tracing back along the pointers.

5. EXPERIMENTAL RESULTS

Prior works [1][13] typically select buffer insertion candidates for van Ginneken's algorithm by uniformly spacing candidates along a given wire. We call this scheme UNI (for uniform), where S is the spacing between consecutive buffers. Note that SPA can be transformed into UNI by using $pen(x, S, L) = 0$ if $x = S$ and $pen(x, S, L) = \infty$ otherwise. In addition, tiles with infinite cost need to be assigned large finite cost and included in the set of edges. These candidates with infinite cost can then be hidden for van Ginneken's algorithm. To show the impact of SPA, we used the following flow.

1. Timing-driven Steiner tree construction via [5],
2. Buffer candidate selection via either SPA or UNI
3. van Ginneken style buffer insertion.

We picked 1000 of the largest nets (in terms of wire capacitance) from an industrial ASIC design in 0.18 micron technology with about 340K instances. We ran both the UNI and the SPA flow on all 1000 nets. An example showing part of the design and the candidate selection algorithms for a particular net is shown in Figure 5 and Figure 6. For these experiments, the chip was broken into 330 tiles on a side and we used $S = 5$ tiles and $L = 35$ tiles.

Tile Density	UNI	SPA	Tile Density	UNI	SPA
0-4	0	0	50-54	405	302
5-9	160	347	55-59	462	301
10-14	221	567	60-64	333	231
15-19	245	553	65-69	318	186
20-24	220	479	70-74	445	265
25-29	193	389	75-79	513	263
30-34	223	372	80-84	440	213
35-39	253	335	85-89	404	181
40-44	274	325	90-94	435	181
45-49	275	280	95-99	81	34

Table 1 Distribution of buffers of UNI and SPA as a function of tile density for 1000 large nets.

For the 1000 nets, UNI inserted 5900 buffers while SPA inserted 5804 buffers. Every time a buffer was inserted, we recorded the density of the tile in which the buffer was placed. The distribution of all the buffer insertions as a function of the tile densities is shown in Table 1. For example, for tiles with density between 10% and 14.99%, UNI inserted 221 buffers while SPA inserted 567. For tiles with density between 90% and 94.99%, UNI inserted 435 buffers while SPA inserted 181. From the table, one can see that SPA was more successful at inserting buffers in tiles with lower density that UNI, with the crossover point being between 45% and 50%.

Figure 4 also shows the distribution of buffers, but in a different way. Instead of grouping the bin densities into 5% ranges as in Table 1, we grouped them in 1% bins, and measured the cumulative percent of buffers inserted into bins. For example, looking at a tile density of 50% on the x-axis shows that the cumulative percentages for UNI and SPA are 36% and 63%, respectively. This means that 36% of the 5900 buffers that UNI inserted were in bins with density of 50% or less, while 63% of the 5804 buffers that SPA inserted were in bins with density of 50% or less. The further to the upper left that the curve lies, the better the job the corresponding algorithm does at density management.

While SPA does a better job than UNI at density management, Table 2 shows virtually nothing is sacrificed in terms of runtime or quality of results. From the table, observe that UNI improves the slack on average of 5.040 ns per net, versus 5.019 ns for SPA. This 21 ps difference is less than one percent of the average slack reduction.[1]

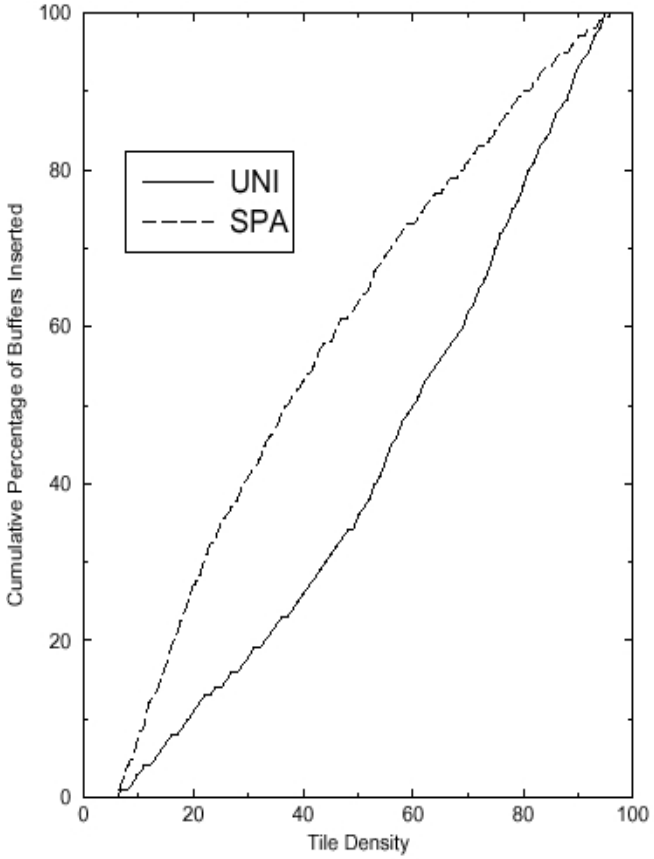

Figure 4 Cumulative percentage of buffers inserted as a function of tile density for 1000 large nets.

Next, observe that the total runtime to select buffer candidate locations is just a fraction of the total time it takes to run van Ginneken's algorithm. Since SPA is more

[1] Of course, for the absolutely most critical nets for which 21 ps is important, one may run with both SPA and UNI and pick the UNI result if it is significantly better.

selective of the potential buffer insertion candidates, this results in a total runtime of 278.7 seconds, versus 1439.8 for UNI. Thus, SPA enables van Ginneken's algorithm to generate a result in less than one-fifth the runtime. Note that one could get a faster runtime for UNI by increasing S. The key points are that SPA uses a fraction of the total runtime required by van Ginneken and that using SPA will not require any extra runtime in the overall flow (it may even reduce it).

	UNI	SPA
Total Buffers inserted for 1000 nets	5900	5804
Average slack improvement (ns)	5.040	5.019
Total Candidate Selection Runtime (s)	19.3	29.3
Total van Ginneken Runtime (s)	1420.5	249.4

Table 2 Statistics comparing UNI and SPA.

6. CONCLUSION

This work shows that for virtually no CPU or timing performance cost, one can bias van Ginneken's algorithm to select from a set of low density candidate locations. Selecting a set of good candidates can be accomplished simply and efficiently via a topological sort based shortest path algorithm on a DAG. Experiments verify that similar results are obtained between the shortest path algorithm and uniform sizing, except that the former does a much better job at managing design density.

Future work in this area could extend this approach to obtain a super fast buffer insertion algorithm that is also aware of its environment.

7. REFERENCES

[1] C. J. Alpert and A. Devgan, "Wire Segmenting for Improved Buffer Insertion", *IEEE/ACM Design Automation Conf.*, 1997, pp. 588-593.

[2] C. J. Alpert, A. Devgan, and S. T. Quay, "Buffer Insertion for Noise and Delay Optimization", *IEEE/ACM Design Automaton Conf.*, 1998, pp. 362-367.

[3] C. J. Alpert, A. Devgan, and S. T. Quay, "Buffer Insertion with Accurate Gate and Interconnect Delay Computation", *IEEE/ACM Design Automation Conf.*, 1999, pp. 479-484.

[4] C. J. Alpert, G. Gandham, M. Hrkic, J. Hu, A. B. Kahng, J. Lillis, B. Liu, S. T. Quay, S. S. Sapatnekar, and A. J. Sullivan, "Buffered Steiner Trees for Difficult Instances", *IEEE Trans. on Computer-Aided Design*, 21 (1), 2002, pp. 3-14.

[5] C. J. Alpert, J. Hu, S. S. Sapatnekar, and P. G. Villarrubia, "A Practical Methodology for Early Buffer and Wire Resource Allocation", *IEEE/ACM Design Automation Conference*, 2001, pp. 189-195.

[6] T. H. Cormen, C. E. Liserson, R. L. Rivest, and C. Stein. Introduction to Algorithms, Second Edition. MIT Press, 2001.

[7] J. Cong, "Challenges and Opportunities for Design Innovations in Nanometer Technologies", in *SRC Working Papers*, Dec. 1997.

[8] J. Cong, T. Kong and D.Z. Pan, "Buffer Block Planning for Interconnect-Driven Floorplanning", *IEEE/ACM Intl. Conf. on Computer-Aided Design*, 1999, pp. 358-363.

[9] S. Dhar and M. A. Franklin, "Optimum Buffer Circuits for Driving Long Uniform Lines", *IEEE Journal of Solid-State Circuits*, 26(1), 1991, pp. 32-40.

[10] Y. Gao and D. F. Wong, "A Graph Based Algorithm for Optimal Buffer Insertion Under Accurate Delay Models", *Design Automation and Test in Europe*, 2001, pp. 535-539.

[11] M. Lai and D. F. Wong, "Maze Routing with Buffer Insertion and Wiresizing", *IEEE/ACM Design Automation Conf.*, 2000, pp. 374-378.

[12] J. Lillis, C.-K. Cheng, T.-T. Y. Lin, and C.-Y. Ho, "New Performance Driven Routing Techniques With Explicit Area/ Delay Tradeoff and Simultaneous Wire Sizing", *33th IEEE/ ACM Design Automation Conference,* 1996, pp. 395-400.

[13] J. Lillis, C.-K. Cheng and T.-T. Y. Lin, "Optimal Wire Sizing and Buffer Insertion for Low Power and a Generalized Delay Model", *IEEE Journal of. Solid-State Circuits*, 31(3), 1996, 437-447.

[14] J. Lillis, C.-K. Cheng and T.-T. Y. Lin, "Simultaneous Routing and Buffer Insertion for High Performance Interconnect", *Sixth Great Lakes Symposium on VLSI*, 1996, pp. 148-153.

[15] T. Okamoto and J. Cong, "Buffered Steiner Tree Construction with Wire Sizing for Interconnect Layout Optimization", *IEEE/ACM International Conference on Computer-Aided Design*, 1996, pp. 44-49.

[16] P. Saxena, N. Menezes, P. Cocchini, and D. A. Kirkpatrick, "The Scaling Challenge: Can Correct-by-Construction Design Help?", *Proc. Intl. Symposium on Physical Design*, 2003, pp. 51-58.

[17] W. Shi and Z. Li, "An O(n log n) Time Algorithm for Optimal Buffer Insertion", *IEEE/ACM Design Automation Conf.*, 2003, pp. 580-585.

[18] L. P. P. P. van Ginneken, "Buffer Placement in Distributed RC-tree Networks for Minimal Elmore Delay", *Intl. Symposium on Circuits and Systems*, 1990, pp. 865-868.

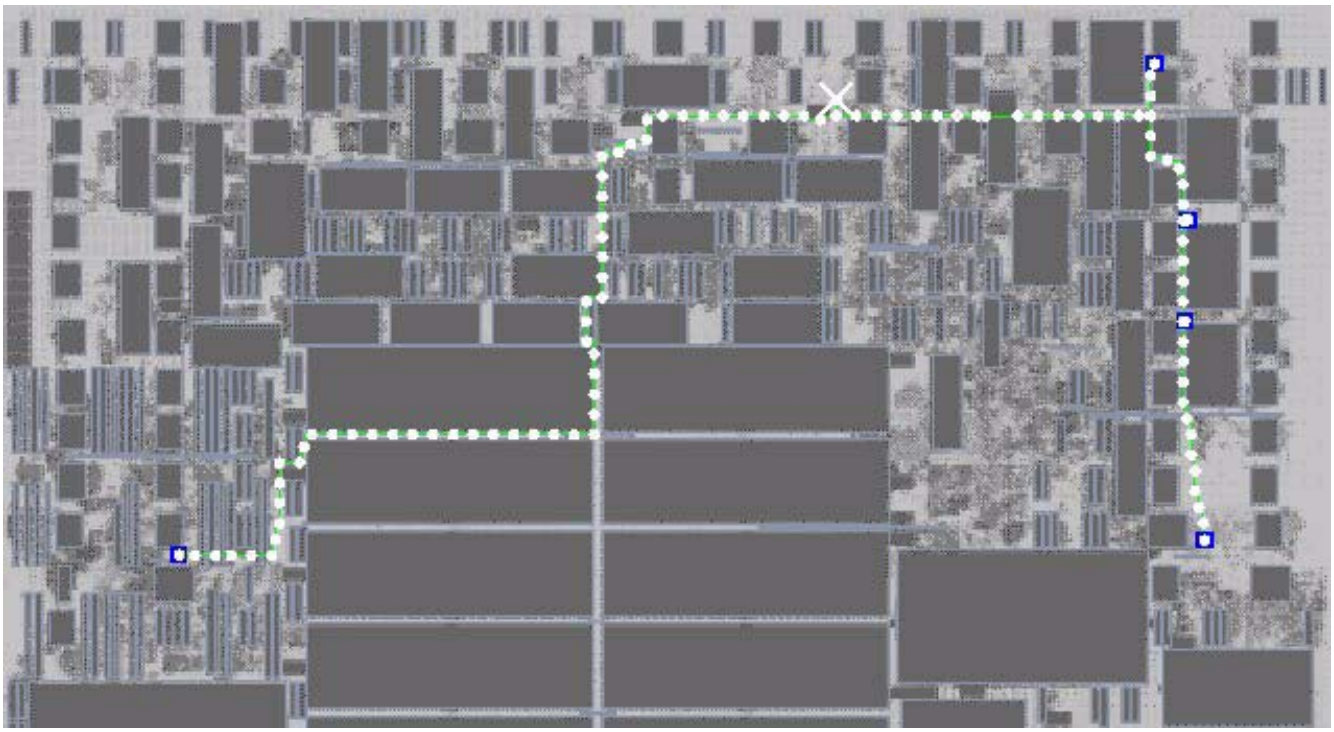

Figure 5 Net with UNI candidate selection. The source is indicated by an x.

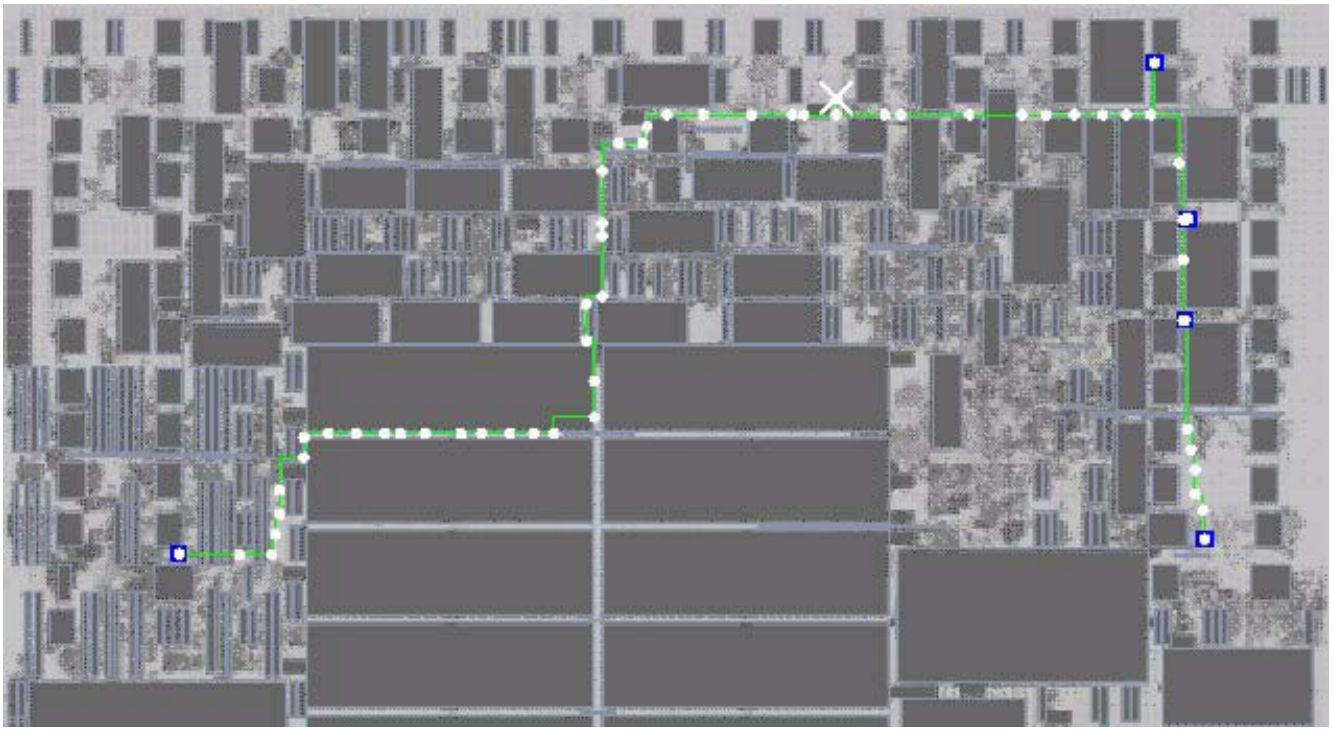

Figure 6 Net with SPA candidate selection. The source is indicated by an x. Note the more erratic spacing than for UNI segmenting.