Clustering based Simulated Annealing for Standard Cell Placement

Sivanarayana Mallela and Lov K. Grover[1]

AT & T Bell Laboratories
Murray Hill, NJ 07974

Abstract

Simulated annealing has been shown to be effective in producing good quality results for the standard cell placement problem. Its main drawback is the excessive computation time required, which increases significantly with the problem size. In this paper we present a novel technique for *reducing the effective problem size* for simulated annealing without compromising the solution quality. We form *clusters* of cells based on their interconnections, and place them first using conventional simulated annealing. We then break up the clusters, and place the individual cells using another simulated annealing process that does a refinement on the placement. The original problem is thus divided into two subproblems, each requiring much less time. The results with this 2-stage simulated annealing have been superior to those with our conventional simulated annealing implementation, with more significant improvements observed for larger chips. For chips with more than 2500 cells, we have observed a factor of 2 to 3 speed-up in CPU time, together with a 6 to 17% improvement in the estimated wire length.

1. Introduction

Simulated annealing is a general purpose combinatorial optimization technique to determine the global minimum of an objective function, and is based on the annealing phenomenon in crystallization [KIR83]. Its characteristic feature is the ability to explore the configuration space via configurations that actually increase the cost function being minimized. A parameter called *temperature* is defined, which is of the same dimension as the cost function. A set of *moves* is selected, with which one state of the configuration space can be generated from another. Moves that reduce the cost are called *downhill* moves, and those that increase the cost are called *uphill* moves. The Metropolis algorithm from statistical physics is used to simulate the system at a given temperature [MET53]. The system is simulated starting from a high temperature, and the temperature is gradually lowered. Moves are generated at random; all downhill moves are accepted, and uphill moves are accepted with a probability of $exp(-\frac{\Delta E}{T})$, where ΔE is the increase in cost and T is the temperature. At very high temperatures almost all the moves are accepted, and the system moves about freely between different areas of the configuration space. At very low temperatures it accepts only downhill moves, and behaves like a greedy algorithm. It has been proven that simulated annealing will asymptotically produce the global minimum, if certain conditions are satisfied on the moves generated at each temperature [MIT86].

In the case of simulated annealing for standard cell placement, the cost function chosen for minimization is the total estimated wiring length for the placement. Simulated annealing has been shown to be effective for the problem [SEC85] [GRO86], yielding 10-30% smaller layouts than from the use of the min-cut partitioning technique [DUN83]. However, the superior results with simulated annealing have been achieved at the expense of an enormous amount of computation time.

Several approaches have been proposed to reduce this computation time for simulated annealing by improving the implementations of move selection, cost computation and temperature scheduling. Green and Supowit [GRE84] presented a technique that improves the efficiency of the algorithm at lower temperatures by only generating moves that will be accepted. This requires the storing of the value for each possible move of a cell, and could have excessive storage requirement. Sechen and Sangiovanni-Vincentelli [SEC86] refined an earlier notion of range-limiting [SEC85] to limit the displacement of a cell to a rectangular window centered at the cell. This window is gradually decreased in size as the temperature is lowered. Huang et. al. [HUA86] devised a way to dynamically determine the cooling schedule and stopping condition, as opposed to predetermining them. Grover [GRO86] used the notion of approximate calculations for the cost computation, which allows the consideration of only cell moves that do not cause overlap of cells. This improves the search efficiency as only "legal" configurations are examined at all times, but without the computational overhead of having to recalculate the lengths of signals associated with the displaced cells after each move. More recently, Sechen and Lee [SEC87] reported a further improvement through early

1. Presently on leave at Cornell University, School of Electrical Engineering, Ithaca, NY 14853.

rejection of non-promising new states, and newer range-limiter window and temperature profile.

In contrast to the above techniques that deal with the simulated annealing process, we address the growth in the computation time due to the size of the problem. In this paper we present a novel technique to reduce the computation time by reducing the size of the problem that simulated annealing has to handle. We do this by splitting the simulated annealing process into two stages. We form *clusters* of cells based on the strength of their interconnections, and place them first using the conventional simulated annealing. Then we break up the clusters, and refine the placement by another stage of simulated annealing at the cell level, but by starting at a lower temperature and greatly limiting the range of cell moves. The original problem is thus divided into two smaller problems, each with much less computational requirement than the original problem.

2. Simulated Annealing With Clusters

A common approach to handle placement problems is to combine top-down and bottom-up techniques. The bottom-up technique is referred to as *clustering*, and involves the grouping of highly connected cells into clusters, and clusters into larger clusters [SCH72]. The goal of the top-down technique is to determine the location for all the clusters. Thus, for row-based standard cell placement, the problem is divided into 2 subproblems; the placement of the clusters into rows, and the placement of the cells within the clusters. All of the cells within a cluster are considered to be together and in the same row during the placement of the clusters, and, usually, also in the final placement [KOZ83]. One-dimensional cluster placement algorithms [SCH72] are used for determining the exact location of the cells in a cluster. In one exception [KAM82], clustering is used to generate an initial placement and each row contains only one cluster. So, to improve the placement, neighboring clusters are repeatedly combined and redivided.

The motivation for using clustering with simulated annealing is to reduce the number of entities that need to be placed. This reduces the search space by reducing the degrees of freedom for simulated annealing, because we do not examine cell configurations where the cells within a cluster are apart. The resulting computational gains need not be at the expense of the quality of the solution, because the search efficiency is also improved. This follows by observing that without clustering, for two highly connected cells to move together, first one would have to be selected to move. Then, the second would have to somehow find its way close to the location of the first one. This would take much longer because several moves would have to be tried before they can come together.

While reducing the computation time is our principal goal, we want to ensure that the cost of the final placement is *no greater* than what can be achieved with the conventional simulated annealing. In the final placement, it is, therefore, important not to place any constraints on the relative locations of cells that were in the same cluster during the top-down placement phase. An inexpensive version of the simulated annealing technique is used to achieve this goal and determine the locations of the cells. Thus, for standard cell placement, we use a 2-stage simulated annealing process.

In the first stage, the conventional simulated annealing algorithm is used to place the various clusters into rows. Each cluster is a *composite* of the individual cells it contains, and has a width equal to the sum of the individual cell widths. We do not attempt to determine the precise locations of the cells within a cluster. So all the terminals of the cluster are assumed to be at the center of the cluster. Thus the simulated annealing program tries to solve a problem with fewer, but larger, entities, and fewer nets -- only those between clusters. The temperature scheduling and the move-selection function are left unchanged from the conventional version. The aim is to still start with a high value for the temperature and gradually decrease it to the freezing point. At the end of this stage, the cells are placed within a small neighborhood of their "best" locations.

In the second stage, we determine the precise location for each cell. First, we break up all the clusters and convert the problem back to the one involving the placement of individual cells. Then we use an inexpensive simulated annealing technique, which limits the range of cell moves to a small neighborhood of their locations, and anneals quickly from a low temperature [GRO87]. The number of moves per cell is also less than in the conventional process. The effect is to permit some amount of hill-climbing, while ensuring that the final configuration does not stray too far from the starting state. After this stage we obtain the final placement, which is a refinement of the placement obtained from the first stage.

The computational requirements for simulated annealing depend on the number of temperatures, and the number of moves considered at each temperature. If t_S is the time taken by a conventional simulated annealing program, the 2-stage strategy takes $(\alpha t_S + \beta t_S)$ amount of time, where α is the factor due to stage 1, and β is the factor due to stage 2. Clearly, the aim is to have $\alpha + \beta < 1$. For a given problem size, β depends mostly on the number of temperatures and the number of moves used in the refinement stage. However, α depends on parameters such as the number of rows for placement and the interconnections among the cells, because they influence the clustering.

3. Clustering Algorithm

Our purpose in using clustering is to reduce the problem size for simulated annealing, while ensuring that the results are at least as good as those without clustering. If larger clusters are formed, there would be fewer of them in number. This leads to a greater reduction in the CPU time for simulated

annealing. But, the reduced degrees of freedom, in terms of allowed standard cell locations, tend to degrade the placement quality. This requires the selection of an *appropriate* limit on the size of the clusters. After some experimentation, we fixed the cluster size limit to be 4% of the average standard cell row length. Clearly, if the the number of rows into which the cells are placed is changed, the resulting clusters and also the impact on the simulated annealing algorithm will vary.

The broad criterion for clustering is that cells within a cluster must have strong connections among themselves and weak connections to other cells. Clusters are usually formed by selecting a set of cells as *seeds* and growing the clusters up from these seeds. A *candidate* cluster that can be combined with a seed must have some connections in common with the seed. In addition, the combined cluster must *not* exceed the size limit.

In the refinement stage of the 2-stage simulated annealing, the extent of the improvement is somewhat bounded, and the quality of the final placement is limited by the quality of the placement from Stage 1. Therefore, we need an effective clustering scheme, which basically comprises a procedure for combining cells into clusters, and suitable criteria for evaluating various candidates for a seed.

3.1 Cluster Comparison Criteria

The usual approach to evaluate clusters has been to formulate a single cost function for comparison [SCH72] [AKE82] [PAL83]. In our approach, we first evaluate some of the attributes of combining a seed cluster with a candidate. Then, instead of formulating a cost function that is a weighted sum of the different attribute values, we prioritize these attributes and compare their values in the prioritized order. The attributes we consider are:

Δ terminal count:	Number of additional terminals the seed would require if the candidate is combined with it
Common net count:	Number of nets that are common between the seed and the candidate. We ignore nets that are connected to more than 40 modules. These would be widely distributed clock and control signals.
Δ local nets:	Number of additional nets that would get localized if the seed is combined with the candidate
Common net fanout:	Total number of clusters other than the combined cluster that the common nets would be connected to.
Total terminal count:	Total number of terminals the combined cluster would have.
Cluster size:	Total width of the combined cluster, which is the sum of the individual widths

For example, consider the 4 clusters in Fig. 1. For the seed cluster A, there are 3 candidates, B, C and D. All of them have a common net count of 1 and common net fanout of 2. No nets are localized, and hence Δ local nets is 0 for all the candidates. The Δ terminal counts are 1, 2 and 3 for B, C and D respectively.

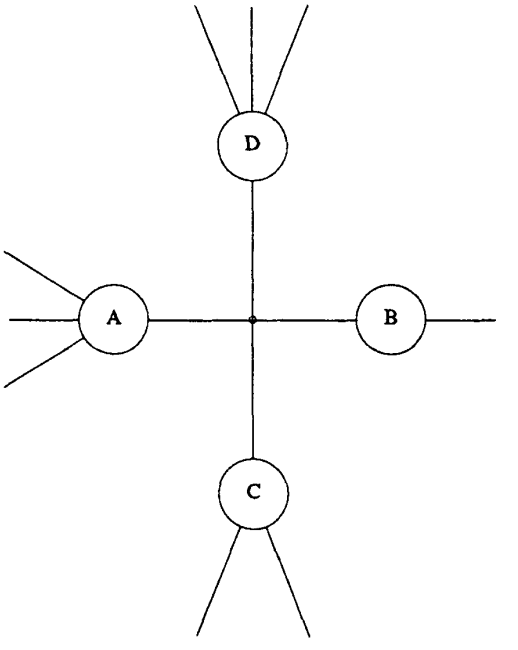

Fig. 1: A 4-Cluster Example

These attributes enable us to characterize seed-candidate pairs. An attribute can also serve to indicate the potential value for another attribute in a future cluster combination. For instance, a common net fanout value of 1 for a seed-candidate pair implies that if the seed and candidate are combined, there is a candidate for the combined cluster for which Δ local net is at least 1.

The priority of the attributes may have to be different for different applications. For the simulated annealing application, our experience indicates that to get good quality results, it is important to localize nets to within clusters. This is understandable since the cost function being minimized is the

total wire length. Therefore highest priority is given to candidates where nets can be localized or potential for localizing nets exists. Terminal count is the next important attribute, since it not only reflects strong connectivity within a cluster, but also impacts the computation time. We have implemented a priority scheme for the above attributes that has been effective in producing "good" clusters, as evidenced by the estimated wire length of the final placement.

3.2 Cluster Combination Algorithm

When clustering is used with simulated annealing, the CPU time for clustering represents only a small fraction of the total CPU time for placement. So our scheme is guided by the view that it is worthwhile to expend a little bit more CPU time to form "better" clusters. All clustering schemes operate by repeatedly combining clusters to form larger clusters, starting with clusters consisting of single standard cells. The distinguishing aspect of the various schemes is the order in which the clusters are selected for enlargement. We take a global approach and look at all of the clusters at any time, and select the best one to enlarge. That is, as new clusters are formed, they are immediately considered in evaluating whether other clusters should be combined. Our clustering algorithm is thus a multi-pass algorithm; in each pass some of the clusters are combined to form larger clusters. The algorithm terminates when no more clusters can be combined without violating the constraint on the size. The number of passes required by the algorithm is thus limited.

During each pass, for each possible seed cluster, we first determine the best candidate cluster to combine with the seed. The comparison routine determines the relative merits of the various candidates for a seed cluster. Note that we have an ordered seed-candidate pair, where if a cluster A is the best candidate for a seed B, B may not be the best candidate for A. We sort the various seed-candidate pairs, using the comparison routine, to determine the best order in which to combine clusters during that pass. Then we step down this ordered list of seed-candidate pairs and combine a cluster with its best candidate if and only if the following conditions are satisfied:

1. Neither the seed nor its candidate have already been combined during the pass;
2. Neither the seed nor its candidate forms a better cluster with any newly formed cluster or any seed cluster that precedes the current seed in the ordered list.

The characteristic feature of our scheme is this *look-ahead*, which ensures that when we form a new cluster, it is the best possible with the seed and the candidate at that time.

For example, consider the seed-candidate pairs in Fig. 2 that are sorted in the non-increasing order of based on the comparison criteria. We first form a new cluster "AC". Since A is already combined, we skip over the pair (B,A). Then, before combining D and B we check if combining D or B with "AC" is better. If so, we do not combine D and B in the current pass; otherwise, we form a new cluster "DB". We skip over (C,D) since both C and D are already combined, and this terminates the current pass.

Seed	Candidate
A	C
B	A
D	B
C	D

Fig. 2: Example Of Sorted Seed-Candidate Pairs

4. Results and Observations

A 2-stage simulated annealing algorithm with a clustering preprocessor has been implemented in the LTX2 VLSI Layout system [COL82]. Our goal was to incorporate clustering into the existing simulated annealing program without making any additional changes. The simulated annealing program is based on the use of approximate calculations [GRO86]. This program uses a default value of 100 for the number of moves per cell per temperature, independent of the problem size. This is done to ensure that the CPU time grows nearly linearly with the number of cells to be placed. The number of temperatures used is 115. For Stage 2 placement refinement, we used GRIM [GRO87] with 12 temperatures and 40 moves per cell per temperature. For clustering, we limited the clusters to be no more than 4% of the average row length.

The clustering data for some actual production chips is presented in Table I. The number of clusters for a given chip depends on the desired row structure for the chip and the interconnection characteristics. Note that even though CHIP2 and CHIP3 have almost the same number of cells, the number of clusters is quite different. Also, the reduction in the total number of terminals is usually quite a bit less than the reduction in the number of modules to be handled by simulated annealing. Thus the clusters have, on average, more terminals than the original cells.

In Table II we compare our clustering based simulated annealing results with those from the conventional simulated annealing, for the chips in Table I, The experiments were all performed on an *Amdahl 5870* which is a 20 MIPS machine. We use the estimated wire length to compare the quality of the results, since this is the cost function being minimized during simulated annealing. As expected, we find that the speed of the program has improved significantly. However, the improvements in speed are closer to the reduction in the total number of terminals, rather than the reduction in the total number of modules. This is because evaluation of the cost associated with moves comprises most of the computation

Table I. Clustering Data for Some Chips

Name	Original Chip		After Clustering		Reduction	
	No. of cells	Tot. Terms	No. of Clust	Tot. Terms	No. of Mods	Tot. Terms
CHIP1	879	2980	624	2519	1.41X	1.18X
CHIP2	2785	9560	1009	5910	2.76X	1.62X
CHIP3	2853	9853	728	5128	3.92X	1.92X
CHIP4	6678	29496	1150	11045	5.81X	2.67X
CHIP5	6906	30632	1167	10852	5.92X	2.82X

Table II. Performance of clustering based simulated annealing
(Number of moves per Cell per Temp. = 100)

Name	Pure Sim. Ann.		Clust. Sim. Ann.		Improvement	
	Wir-Len	CPU Time	Wir-Len	CPU Time	Wir-Len	CPU Time
CHIP1	1.91	31 min	1.89	29 min	1%	1.1X
CHIP2	1.07	90 min	1.02	35 min	5%	2.5X
CHIP3	1.87	70 min	1.80	35 min	4%	2.0X
CHIP4	2.67	207 min	2.22	75 min	16%	2.8X
CHIP5	2.50	220 min	2.13	70 min	15%	3.1X

time. This evaluation time is greater for a module with more terminals. Since we fixed the number of moves per cell per temperature at 100, the cost computation has somewhat of a linear dependency on the total number of terminals.

In addition to the improvement in CPU time, we also find that the quality of the results has improved. Given infinite time, simulated annealing is guaranteed to find the globally optimum solution. However, in practical cases, the quality of the solution depends on the amount of CPU time expended on the problem. After some point, the improvement in the solution becomes smaller and smaller, and may not be worth the additional CPU time. This saturation time increases as the solution space gets larger. In our conventional simulated annealing program, the total number of moves grows only linearly with the problem size, while the search space grows exponentially. So, we operate closer to saturation for small chips but not for larger ones. Thus, there is scope for improvement in the quality of results for larger chips. By clustering the cells prior to simulated annealing, we cut down on the number of modules to be placed. This brings the permitted CPU time closer to the saturation time, and hence improves the quality of results.

In simulated annealing programs where the number of moves per cell is increased as the number of cells gets larger, the clustering technique would yield a greater improvement in the CPU time. However, there may be no improvement in the quality of the results, since a more thorough search is done for the original problem. For example, for CHIP5 above, with our implementation, the number of moves per cell is decreased by a factor of 6 in Stage 1. If we use the optimal number of moves per cell as in the original TimberWolf program [HUA86], Stage 1 would require only require 400 moves per cell for CHIP5 above, as opposed to 1200 for the original problem. This represents a reduction of almost a factor of 18 in the number of moves.

5. Conclusion

We have implemented a combined top-down and bottom-up approach to simulated annealing for standard cell placement. The scheme involves a two-stage annealing process. We first form clusters of cells that are highly connected, and place these clusters using conventional simulated annealing. Then a second stage of simulated annealing is used to refine the placement at the cell level. This latter process confines the cells to local moves, since they are already in the proper neighborhood.

The degree of improvement in computation time depends on the particulars of the problem, such as the limiting size of clusters, and the interconnections

among the cells. So if the row structure for placement consists mainly of small rows, the improvement may be marginal. For larger problems, we can either achieve an additional improvement in the solution quality or a much more significant improvement in the CPU time, depending on whether or not the number of moves per cell is increased to account for the larger search space. We have obtained a factor of 2 - 3 improvement in CPU time, and 6 - 17% improvement in wire-length for large chips (> 2500 cells).

In conclusion, we have presented a general technique that can potentially improve the performance of any simulated annealing program for row-based standard cell placement.

6. References

[AKE82] S. B. Akers, "Clustering techniques for VLSI", *Proc. IEEE Int. Symp. on Circuits and Systems*, pp 472-476, 1982.

[COL82] B. W. Colbry and J. Soukup, "Layout aspects of the VLSI microprocessor Design", *Proc. Int. Symp. on Circuits and Systems*, May 1982, pp. 1214-1228.

[DUN83] A. E. Dunlop and B. W. Kernighan, "A placement procedure for polycell VLSI circuits," *Proc. Int. Conf. on CAD*, pp.1045-1048, 1983.

[GRE84] J. Greene and K. Supowit, "Simulated annealing without rejected moves", *roc. Int. Conf. on Computer Design*, Oct. 1984, pp. 658-663.

[GRO86] L. K. Grover, "A new simulated annealing algorithm for standard cell placement," *Proc. Int. Conf. on CAD*, 1986, pp. 378-380.

[GRO87] L. K. Grover, "Standard cell placement using simulated sintering,"*Proc. 24th Design Automation Conf.*, 1987, pp. 56-59.

[HUA86] M. D. Huang, F. Romeo and A. Sangiovanni-Vincentelli, "An efficient general cooling schedule for simulated annealing", *Proc. Int. Conf. on CAD*, Nov. 1986, pp. 381-384.

[KAM82] T. Kambe, et. al., "A placement algorithm for polycell LSI and its evaluation", *Proc. 19th Design Automation Conf.*, 1982, pp. 655-662.

[KIR83] S. Kirkpatrick, C. Gelatt Jr., and M. Vecchi, "Optimization by Simulated Annealing", *Science*, vol 220, No 4598, May 1983, pp 671-680.

[KOZ83] T. Kozawa, et. al., "Automatic placement algorithms for high packing density VLSI", *Proc. 20th Design Automation Conf.*, 1983, pp. 175-181.

[MET53] N. Metropolis, A. Rosenbluth, A. Teller and E. Teller, "Equation of state calculations by fast computing machines", *Jour. Chem. Phys.*, Vol. 21, pp. 1087, 1953.

[MIT86] D. Mitra, F. Romeo and A. Sangiovanni-Vincentelli, "Convergence and finite time behavior of simulated annealing," *Advances in Applied Probability, Vol. 18 1986*, pp. 747-771.

[PAL83] C. A. Palesko and L. A. Akers, "Logic partitioning for minimizing gate arrays", *IEEE Trans. on CAD*, pp 117-121, April 1983.

[SCH72] D. M. Schuler and E. Ulrich, "Clustering and linear placement", *Proc. 9th Design Automation Workshop*, pp 50-56, 1972.

[SEC85] C. Sechen and A. Sangiovanni-Vincentelli, "The TimberWolf placement and routing package", *IEEE Journal of Solid-State Circuits*, April 1985.

[SEC86] C. Sechen and A. Sangiovanni-Vincentelli, "TimberWolf3.2: A new standard cell placement and global routing package", *Proc. 23rd Design Automation Conf.*, 1986, pp. 432-439.

[SEC87] C. Sechen and K. Lee, "An improved simulated annealing algorithm for row-based placement", *Proc. Int. Conf. on CAD*, 1987, pp. 478-481.