Local Search for Final Placement in VLSI Design

Oluf Faroe, David Pisinger, and Martin Zachariasen

Dept. of Computer Science, University of Copenhagen, DK-2100 Copenhagen Ø, Denmark.
{oluf,pisinger,martinz}@diku.dk

Abstract

A new heuristic is presented for the general cell placement problem where the objective is to minimize total bounding box netlength. The heuristic is based on the Guided Local Search (GLS) metaheuristic. GLS modifies the objective function in a constructive way to escape local minima. Previous attempts to use local search on final (or detailed) placement problems have often failed as the neighborhood quickly becomes too excessive for large circuits. Nevertheless, by combining GLS with Fast Local Search it is possible to focus the search on appropriate sub-neighborhoods, thus reducing the time complexity considerably.

Comprehensive computational experiments with the developed algorithm are reported on small, medium and large industrial circuits, and for standard cell and general cell variants of the problem. The experiments demonstrate that the developed algorithm is able to improve the estimated routing length of large-sized general cell layouts with as much as 20 percent.

The general nature of the proposed method makes it easy to incorporate other goals, such as routability and timing constraints, into the objective function. Current layout algorithms use a feedback approach in which a placement is evaluated by performing (partial) routing and timing analysis; the output of this analysis is then used to construct an improved placement. This iterative nature of the design process calls for placement algorithms that take an existing placement and construct an improved placement that resembles the original one, but in which the information from the routing/timing analysis is taken into account.

1 Introduction

The placement problem in VLSI design is the first phase in the process of designing the physical layout of a chip. This makes the placement problem of paramount importance, since the quality of the attainable routing is to a high degree determined by the placement. In the placement problem we are given a set of rectangular modules (or cells/circuits/macros) of different height and width that should be placed disjointly on the chip surface. Every module has a number of connection points, so-called *pins*, and the *netlist* is a partitioning of the pins into *nets* that should be interconnected.

The problem is to place the modules such that an objective function that reflects the quality of the placement is minimized. Most objective functions in placement add up the contribution from each net separately, with the overall objective of minimizing total wiring length after routing. Clearly, such an objective function has the weakness of not taking timing issues explicitly into account, since minimizing total length may leave critical nets having a significant signal delay. However, the general nature of the method proposed in this paper makes it easy to incorporate additional or alternative objectives.

We present a new iterative placement algorithm which is well-suited for solving the final (or detailed) placement problem. The algorithm both takes the packing problem, i.e. placing the modules disjointly, and the total bounding box (BB) netlength into account. The algorithm uses the *Guided Local Search (GLS)* metaheuristic [18, 19] for controlling the search. The neighborhood structure is simple: Flipping and/or moving a single module along one of the coordinate axes. This neighborhood has previously been used by [6, 10, 11, 12, 14, 21], in particular in conjunction with simulated annealing. The weakness of all these algorithms is the slow convergence towards good solutions — which is an inherent feature of simulated annealing. By combining GLS with the *Fast Local Search (FLS)* approach [18, 19], an algorithm that both finds good solutions quickly and in the long run converges towards high-quality solutions is obtained.

In addition to its applicability as a final placement algorithm, the new heuristic can be used in the following setting. Current layout algorithms use a feedback approach in which a placement is evaluated by performing (partial) routing and timing analysis; the output of this analysis is then used to construct an improved placement. This iterative nature of the design process calls for placement algorithms that take an existing placement and construct an improved placement that resembles the original one, but in which the information from the routing/timing analysis is taken into account.

Finally, the new algorithm can be used to construct high-quality placements of small general cell circuits. This is known to be a very difficult problem in practice, and our experimental results show that the new algorithm on average produces significantly better solutions than existing algorithms from the literature. For some instances the total netlength is reduced by more than 20 percent when compared to the recent results for the O-Tree algorithm [5].

The paper is organized as follows. In Section 2 we define the placement problem. In Sections 3 and 4 we present the details of applying GLS and FLS, respectively, to the placement problem. Extensive computational results are presented in Section 5, and concluding remarks are given in Section 6.

2 The Placement Problem

The placement problem asks to assign locations to the modules of a circuit such that these are within the available placement area and do not overlap. We assume that modules may have arbitrary rectangular dimensions, thus the considered layout style is *general cell layout*. The objective of the problem is to minimize the total length of the nets connecting the modules.

To be more formal, a circuit $\mathcal{C}$ is defined by the tuple $\mathcal{C} = (\mathcal{A}, \mathcal{M}, \mathcal{P}, \mathcal{N})$ where $\mathcal{A}$ is the placement area, $\mathcal{M}$ is the set of modules, $\mathcal{P}$ is the set of pins, and $\mathcal{N}$ is the netlist defining which pins should be connected. We will assume that the placement area is defined as the integer grid $\mathcal{A} = \{0, \ldots, W\} \times \{0, \ldots, H\}$. For each module $m \in \mathcal{M}$ the corresponding (integer) width is w_m and (integer) height is h_m. Moreover let (x_m, y_m) denote the (integer) coordinates of the lower left corner of the module in the placement area $\mathcal{A}$. In order to not exceed $\mathcal{A}$ the coordinates of module m must satisfy $x_m \in \{0, \ldots, W - w_m\}$ and $y_m \in \{0, \ldots, H - h_m\}$. For technical reasons, some of the modules may be fixed at a given position. In this case the module coordinates (x_m, y_m) may not be changed. If some part of the circuit area $\mathcal{A}$ is not available for the modules, this space may be represented by one or more fixed modules which do not have any pins.

Depending on the restrictions from the layout style and the fabrication technology, modules may be allowed to change orientation. In the present definition we will allow modules to be rotated in steps of 90 degrees and to be reflected around the x- and y-axis. This gives eight different orientations of each module. To make the following discussion simpler, we will not mention the orientations explicitly, although they should be taken into account in all definitions and algorithms.

Each pin $p \in \mathcal{P}$ has a relative position within its module m. This is denoted the offset of the pin p in m. The netlist $\mathcal{N}$ is defined as a partitioning of the pins $\mathcal{P}$, such that every pin $p \in \mathcal{P}$ is part of exactly one net $N \in \mathcal{N}$.

For a given placement P we let $N_P \subset \mathbb{Z}^2$ denote the *set of pin coordinates* corresponding to net $N \in \mathcal{N}$, and $BB(N_P)$ be the bounding box netlength for the set N_P. In order to make it possible to differentiate between how much the individual nets should contribute to the objective function, a *net weight function* $w : \mathcal{N} \to \mathbb{R}_{\geq 0}$ assigns some weight $w(N)$ to each net $N \in \mathcal{N}$. We can now formulate the min-sum placement problem as:

Placement Problem: *For a given circuit $\mathcal{C}$ find a placement P which minimizes the objective $\sum_{N \in \mathcal{N}} w(N) \cdot BB(N_P)$ for all feasible placements of $\mathcal{C}$.*

To further correct the BB netmodel a multiplier function $w_{BB}(N)$ may be applied. Brenner [1] empirically shows that the multiplier function $w_{BB}(N) = \frac{7}{10}|N|^{\frac{1}{4}}$ makes the bounding box netmodel become very close to the wire length after routing. In our implementation, however, no multiplier function is used since this only makes a very small difference in practice [2]. Also, using no multiplier function makes it easier to compare our solution values with those from the literature.

3 Guided Local Search

In this section the application of the metaheuristic *Guided Local Search (GLS)* to the placement problem is described. GLS has proven to be effective on a wide range of problems [8, 17, 18, 19]; the metaheuristic can be applied to any combinatorial optimization problem given by a solution space $\mathcal{X}$ for which an objective function value $f(\mathbf{x})$ (to be minimized) and neighborhood $\mathcal{N}(\mathbf{x}) \subset \mathcal{X}$ is defined for every solution $\mathbf{x} \in \mathcal{X}$. The use of GLS for placement was motivated by recent results on packing problems in two and three dimensions [3].

$\mathbf{x}$ = initial solution
$\mathbf{x}_{best}$ = best solution
while stopping condition not satisfied **do**
  $\mathbf{x}^* = \text{LOCALOPT}_h(\mathbf{x})$
  if $f(\mathbf{x}^*) < f(\mathbf{x}_{best})$ **then**
    $\mathbf{x}_{best} = \mathbf{x}^*$
  end if
  penalize and modify h
  $\mathbf{x} = \mathbf{x}^*$
end while
return $\mathbf{x}_{best}$

Figure 1: Outline of Guided Local Search (GLS).

Given an initial solution $\mathbf{x}_0 \in \mathcal{X}$, local search visits a sequence of solutions $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_k$ such that $\mathbf{x}_i \in \mathcal{N}(\mathbf{x}_{i-1})$ for $i = 1, 2, \ldots, k$. When the series of solutions $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_k$ fulfills $f(\mathbf{x}_0) > f(\mathbf{x}_1) > \ldots > f(\mathbf{x}_k)$ the process is denoted *local optimization*. Local optimization stops when the current solution $\mathbf{x}_k$ is a local minimum, that is, when $\mathcal{N}(\mathbf{x}_k)$ contains no solution better than $\mathbf{x}_k$. Applying local optimization to a solution using the objective function f will be denoted by the operator LOCALOPT_f. In the above case we have $\mathbf{x}_k = \text{LOCALOPT}_f(\mathbf{x}_0)$.

GLS extends local search with the concept of *features*, i.e., a set of attributes which characterize a solution to the problem in a natural way. GLS assumes that any solution can be described using a set of M features, that is, a solution $\mathbf{x} \in \mathcal{X}$ either has or does not have a particular feature $i \in \{1, \ldots, M\}$. The indicator function $I_i(\mathbf{x})$ is 1 if $\mathbf{x}$ has feature i and 0 otherwise. Features should be defined such that the presence of a feature in a solution has a more or less direct contribution to the value of the objective function. This direct or indirect contribution is reflected in the *cost* c_i of the feature. A feature with a high cost is not attractive and may be *penalized*. The number of times a feature has been penalized is denoted by p_i, and is initially zero. Penalties are incorporated into the search by constructing an augmented objective function

$$h(\mathbf{x}) = f(\mathbf{x}) + \lambda \cdot \sum_{i=1}^{M} p_i \cdot I_i(\mathbf{x}) \qquad (1)$$

where λ is a regularization parameter which balances the objective function to the contribution from the penalty term. Instead of optimizing the function f, GLS optimizes the augmented objective function h.

The main GLS algorithm performs a number of optimization steps, each transforming a solution $\mathbf{x}$ into a local minimum $\mathbf{x}^* = \text{LOCALOPT}_h(\mathbf{x})$ (Figure 1). Note that since all penalties initially are zero, the first iteration of GLS actually finds a local optimum with respect to f. At each local minimum $\mathbf{x}^*$ GLS takes a *modification action* which modifies h by penalizing one or more features by incrementing their p_i value by one. The idea is to penalize, in a controlled matter, the features in $\mathbf{x}^*$ which have the largest contribution to the value of the objective function. The modification action therefore penalizes those features which have the maximum *utility*

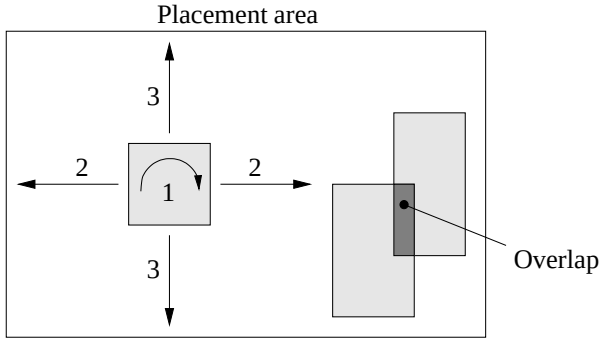

Figure 2: Illustration of possible moves of a single module: (1) change of orientation combined with either (2) translation along the x-axis or (3) translation along the y-axis.

defined as

$$\mu_i(\mathbf{x}^*) = \frac{c_i}{1+p_i} \cdot I_i(\mathbf{x}^*) \tag{2}$$

Informally, these are the features with maximum cost in $\mathbf{x}^*$ which have not been penalized too often in the past. After having penalized these features, the local optimization continues from $\mathbf{x}^*$ — now with respect to the modified h function.

3.1 Applying GLS to Placement

In order to apply local search to the placement problem the constraint saying that no modules may overlap is removed, and instead overlap is penalized in the objective function. The problem handed to the GLS heuristic is thus a *relaxed* placement problem in which all the modules must be placed within the placement area and the objective is to minimize total netlength as well as the overlap between modules.

For a solution $\mathbf{x} \in \mathcal{X}$ we define the neighborhood $\mathcal{N}(\mathbf{x})$ as the set of solutions which can be obtained by orienting and translating any single module along one of the coordinate axis (Figure 2). The neighborhood $\mathcal{N}(\mathbf{x})$ makes it possible to traverse between any pair of solutions by following a path of neighboring solutions. The neighborhood size $\mathcal{O}(|\mathcal{M}|(W+H))$ is *not* polynomial in the input size.

The objective value $f(\mathbf{x})$ is defined as a linear combination of pairwise overlap and BB netlength:

$$f(\mathbf{x}) = \sum_{m,n \in \mathcal{M}} \text{overlap}_{mn}(\mathbf{x}) + \beta \sum_{N \in \mathcal{N}} BB_N(\mathbf{x}) \tag{3}$$

Here $\text{overlap}_{mn}(\mathbf{x})$ of two distinct modules $m, n \in \mathcal{M}$ is defined as the area of the intersection between the modules, and $BB_N(\mathbf{x})$ is the BB netlength for net $N \in \mathcal{N}$ in solution $\mathbf{x}$. The first sum in (3) is only evaluated for each pairs of modules where $m < n$, assuming some linear ordering of the modules. We observe that $\mathbf{x} \in \mathcal{X}$ is a feasible solution to the placement problem if and only if $\sum_{m,n \in \mathcal{M}} \text{overlap}_{mn}(\mathbf{x}) = 0$. The parameter β is used to balance the two conflicting terms of the objective function.

Two sets of features are defined, each reflecting the contribution of the overlap and the contribution of the total netlength in the objective function. An *overlap feature* is defined for each pair of modules $m, n \in \mathcal{M}$, such a particular solution $\mathbf{x} \in \mathcal{X}$ exhibits an *overlap feature* if the modules overlap, as stated by the indicator function:

$$I^{of}_{mn}(\mathbf{x}) = \begin{cases} 1 & \text{if overlap}_{mn}(\mathbf{x}) > 0 \\ 0 & \text{otherwise} \end{cases}$$

All overlap features must be eliminated to produce a feasible placement. The overlap feature *cost* should reflect the contribution of the feature to the overlap term in the objective function. As in [3] we identify *bad* overlaps with the general principle that an overlap between large modules is worse than an overlap between small modules. This observation is also made in most packing heuristics, where better solutions usually are obtained if the boxes (modules) which cover a large area are placed prior to the small. Thus, the overlap feature cost c^{of}_{mn} should both depend on the overlap between the modules m and n, and on the area of the modules:

$$c^{of}_{mn}(\mathbf{x}) = \text{overlap}_{mn}(\mathbf{x}) + \text{area}(m) + \text{area}(n)$$

This definition places a high cost on features which correspond to pairs of large overlapping modules. The utility function corresponding to the overlap features is defined as in (2).

A *connection feature* is defined for each pair of modules $m, n \in \mathcal{M}$. Let $\text{rdist}_{mn}(\mathbf{x})$ be the minimum rectilinear distance between two points covered by modules m and n, respectively; if the modules overlap then $\text{rdist}_{mn}(\mathbf{x}) = 0$. For a given pair of modules $m < n$ and a particular solution $\mathbf{x} \in \mathcal{X}$ we let $\mathbf{x}$ exhibit a connection feature if there is a net connecting the modules and the rectilinear distance between the modules is positive, as given by the indicator function:

$$I^{cf}_{mn}(\mathbf{x}) = \begin{cases} 1 & \text{if } \text{rdist}_{mn}(\mathbf{x}) > 0, \text{ and } m, n \text{ connected} \\ 0 & \text{otherwise} \end{cases}$$

For modules m and n the connection feature cost c^{cf}_{mn} should reflect how much the modules contribute to the *BB* netlength of the nets which connect m and n. As $\text{rdist}_{mn}(\mathbf{x})$ is a lower bound on the BB netlength for the nets which connect m and n, pairs of connected modules which are placed at a large rectilinear distance from each other should be punished:

$$c^{cf}_{mn}(\mathbf{x}) = \text{rdist}_{mn}(\mathbf{x})$$

Again, the utility function is given by (2).

As the neighborhood $\mathcal{N}(\mathbf{x})$ allows translation of a module along the horizontal *or* vertical axis it is suitable to also split the indicator function I^{cf}_{mn} into a horizontal and a vertical part I^{cfx}_{mn} and I^{cfy}_{mn}. The *augmented objective function* for the relaxed placement problem becomes

$$\begin{aligned} h(\mathbf{x}) = f(\mathbf{x}) &+ \lambda^{of} \sum_{m,n \in \mathcal{M}} p^{of}_{mn} \cdot I^{of}_{mn}(\mathbf{x}) \\ &+ \lambda^{cf} \sum_{m,n \in \mathcal{M}} p^{cf}_{mn} \cdot (I^{cfx}_{mn}(\mathbf{x}) + I^{cfy}_{mn}(\mathbf{x})) \end{aligned} \tag{4}$$

where $I^{cfx}_{mn}(\mathbf{x}) = 1$ if and only if the horizontal distance between two connected modules m, n is strictly positive, and $I^{cfy}_{mn}(\mathbf{x})$ is defined in a similar way. The sums are only evaluated for pairs of modules where $m < n$. Parameters λ^{of} and λ^{cf} are the regularization parameters corresponding to the overlap and connection features, respectively. The concrete choice of these will be discussed in Section 5.2.

3.2 Feature Costs and Duration of Penalties

We distinguish between *soft* and *hard* features. A soft feature indicates whether a *feasible* solution has or does not have a certain attribute. In many applications of GLS only this type of features is used. Hard features are related to the *infeasibility* of solutions. The presence of a hard feature in a solution means that the solution is not feasible, corresponding to the violation of a constraint. This is typically reflected through a penalty term in the objective function — in a manner similar to Lagrangian relaxation.

The fundamental difference between soft and hard features affects the costs of features and the duration of their penalties. Soft features are often given fixed costs while hard features have variable costs which depend on the amount of violation of the respective constraint.

The augmented objective function (4) contains two terms related to the overlap features and connection features. The penalties corresponding to the *overlap features* are chosen such that they increase monotonously as this makes it easier to balance the conflicting terms in the augmented objective function (4). At the beginning of the search the overlap penalties will be small and the search will mostly be guided by the total bounding box length. As the overlap penalties increase, the control is gradually transferred to eliminate overlap between modules. When a feasible placement is found all penalties are reset; this is a variation of the reset strategy described in [16].

The *connection features* are soft and thus some techniques are needed to restrict the duration of the penalties. We used a refinement of the multiple feature sets (MFS) strategy proposed by Voudouris [16]. Two sets of counters p_i and f_i are used where f_i is increased each time the penalty p_i is increased. But whereas the penalty is decreased after t iterations, f_i continues to increase. As opposed to the MFS strategy, only the penalty term appears in the augmented objective function while the f_i values are used in the utility function for the overlap features as follows

$$\mu_i(\mathbf{x}^*) = \frac{c_i}{1 + f_i} \cdot I_i(\mathbf{x}^*)$$

Thus the short term memory list is used to retract the penalty from the objective function after t LOCALOPT$_h$ operations, but the frequency memory is maintained in f_i, thus providing diversity in the utility function.

3.3 Problem Subdivision for Large Problems

Although much effort has been done to reduce the computational complexity of each LOCALOPT$_h$ operation (see Section 4), practical experience indicates that the algorithm performs badly for instances with a huge number of modules. Thus a technique was developed where the placement problem is divided into a number of overlapping regions, each region being of appropriate size for the algorithm. The regions are then considered by the GLS heuristic in a round-robin fashion. As the regions overlap, modules are allowed to traverse between the regions and move over the whole placement area. The technique has similarities to the tiles of overlapping windows by Kennings [7].

The overlapping regions are constructed such that they contain at least M modules, where M is an experimentally determined constant (see Section 5.2). Initially the whole placement area $\mathcal{A}$ is split into a grid, where each grid cell is marked as a candidate cell for growing a region. Then the algorithm repeatedly selects an unmarked candidate cell, grows a region around the cell, and marks all the encircled cells.

The GLS algorithm is repeatedly applied to a region keeping all cells outside the region fixed. The regions are considered according to increasing density, such that the easiest regions are considered first. The GLS algorithm moves from one region to another when no improving placement has been found within some fixed number of FLS calls.

4 Fast Local Search

In most applications of GLS the LOCALOPT$_h$ operation is the computational bottleneck. Searching large neighborhoods for an improving solution can be very time consuming. *Fast Local Search (FLS)* is a modification of local search which speeds up the search by shadowing less promising parts of the neighborhood. Although the development of FLS was closely connected to its application in the GLS framework, it can be used with other local search metaheuristics as well.

In FLS the neighborhood is divided into a number of smaller sub-neighborhoods which can be either *active* or *inactive*. Initially all sub-neighborhoods are active. FLS now continuously visits the active sub-neighborhoods in some order. If a sub-neighborhood is examined and does not contain any improving move it becomes inactive. Otherwise it remains active and the improving move is performed; this may cause some sub-neighborhoods to be *reactivated*, if we expect these to contain improving moves as a result of the move just performed. As the solution value improves, more and more sub-neighborhoods become inactive, and when all sub-neighborhoods have become inactive the best solution found is returned by FLS as a (pseudo) local minimum.

The neighborhood is split into sub-neighborhoods by making an association between features and sub-neighborhoods. This association is used each time GLS settles in a local minimum. As penalties are assigned to one or more features, the sub-neighborhoods associated with the penalized features are *activated* and FLS is restarted. The *limited reactivation* and the association between the penalized features and the reactivated sub-neighborhoods focuses the search.

To apply FLS to the placement problem we let each module $m \in \mathcal{M}$ correspond to a sub-neighborhood. A sub-neighborhood thus holds moves for all orientations and all translations of a single module along the coordinate axes. A module can be either *active* or *inactive* corresponding to the state of the sub-neighborhood. The active modules are kept in a queue. The FLS optimization repeatedly removes a module from the queue and evaluates the moves which can be performed for the module. If a move exists which improves the augmented objective function the move is performed and the module is re-appended to the queue. Otherwise the module is made inactive. FLS stops when the queue is empty and returns the current solution as the local minimum.

Features are penalized when FLS terminates in a local minima. Each penalized feature corresponds to a pair of modules on the placement area. After the features have been penalized we therefore reactivate

1. the pair of modules corresponding to the penalized overlap feature and all modules which overlap with these two modules

2. the pair of modules corresponding to the penalized connection feature and all modules which are connected to these two modules.

This reactivation scheme permits FLS to pay attention not just to the penalized modules but also to the modules interacting with the concerned modules. It may be profitable to penalize more than one feature in a FLS minimum. This makes it possible to reactivate more modules after each FLS and shifts the computational effort from the penalty assignment to the evaluation of FLS instead. The number of penalties assignments after FLS is called the *penalty depth* of GLS (the setting of this parameter is discussed in Section 5.2).

The performance of FLS depends on the size of the sub-neighborhoods. For the placement problem, a naive evaluation of a sub-neighborhood would require pseudo-polynomial time, since there are $O(W + H)$ possible locations of module m.

As the terms in the augmented objective function (1) are all piecewise linear functions, we may restrict the sub-neighborhood to the break-points of the terms. In this way the minimum of $h(\mathbf{x})$ when considering the translation of module $m \in \mathcal{M}$ along the x-axis can be found in time $O(|\mathcal{P}_m| + (|\overline{\mathcal{M}}_m| + |\mathcal{M}_m| + |\mathcal{N}_m|)\log(|\overline{\mathcal{M}}_m| + |\mathcal{M}_m| + |\mathcal{N}_m|))$. Here $\mathcal{N}_m$ is the set of nets which contain a pin of module m, $\mathcal{P}_m$ is the set of pins for the nets in $\mathcal{N}_m$, and $\mathcal{M}_m$ is the set of modules that are connected to m by some net.. Finally, $\overline{\mathcal{M}}_m$ is the set of modules that m intersects when translated along the x-axis (see [2] for details).

During the course of FLS, the algorithm maintains for every module the list of modules that it overlaps with. When FLS finishes, this information is used to quickly identify the overlap feature with maximum penalty as defined by (2). The connection feature with maximum utility is similarly identified quickly be iterating through the nets and computing the distance between the modules that are interconnected by each net.

5 Computational Experiments

In this section we present the results from the computational experiments. The GLS heuristic was applied to three different layout styles which besides the general cell layout included pure standard cell layout and large real-life circuits with mixed cell layout.

In order to evaluate the placements produced by GLS three different placement heuristics were used as benchmarks: O-Tree [5], TimberWolf v.1.2 commercial edition (TW) [12, 15] and XQ [20]. The core in TW is a simulated annealing placement heuristic which is specialized for standard cell placement [13]. The XQ placement program uses quadratic optimization and sophisticated partitioning.

For the small circuits, GLS is used as a stand-alone heuristic starting with a random initial placement. For large-sized circuits the quality of the initial solution becomes crucial for the overall performance of the heuristic. One of the reasons is the use of overlapping regions (Section 3.3), which is unable to give the algorithm the necessary global view of the problem [2]. Therefore, we here report on *improving* existing placements on large-sized instances.

In Section 5.1 we describe the computational environment and the benchmarks used in our computational study. The setting of parameters is presented in Section 5.2. Finally, in Section 5.3 the computational results are presented and compared with the output from other heuristics.

5.1 Benchmark Circuits

The GLS placement heuristic was implemented in C++ and compiled using the GNU C++ compiler. All tests were performed on an Intel Pentium III 800Mhz with 1GB memory running the Linux RedHat 6.1 operating system.

The following benchmark circuits were used in the experiments (see Table 1). It should be noted that for some of the benchmarks there are restrictions on the possible orientations of the modules. These restrictions are obeyed by the algorithm.

MCNC general cell circuits

These circuits were considered in a recent paper on the O-Tree heuristic [5]; this heuristic reports some of the best results on these circuits. One problem with these results is that O-Tree has no restrictions on the size of the placement area. To resolve this problem we have chosen to create a square placement area with the same area as reported for placements produced by O-Tree. This solution does not allow an exact comparison between O-Tree and the GLS heuristic, but on the other hand does not favor GLS: It restricts the GLS heuristic to a square placement area and reduces the number of possible placements.

Circuit	Modules		Pins		Nets
	Free	Fixed	Non-I/O	I/O	
MCNC general cell circuits					
Apte	9	0	287	73	97
Xerox	10	0	698	2	183
Hp	11	0	309	45	71
Ami33	33	0	520	40	122
Ami49	49	0	953	22	396
MCNC standard cell circuits					
Primary1	752	0	2,941	130	903
Struct	1,888	0	5,471	64	1,920
Industry1	2,271	0	8,837	814	2,593
Primary2	2,907	0	11,226	188	3,029
Biomed	6,417	0	21,040	97	5,742
Industry2	12,142	0	48,404	495	13,419
Industry3	15,059	0	68,418	374	21,940
Avqlarge	25,114	0	82,752	64	25,385
Golem3	99,932	0	338,623	2768	144,950
Industrial IBM circuits					
CLK	29,056	42	111,902	414	30,293
DECODER	54,911	17	188,275	1,016	59,256
PU	163,960	550	617,190	744	184,231

Table 1: Circuit characteristics.

MCNC standard cell circuits

The MCNC standard cell circuits were obtained from the recent GSRC release [4]. The circuits are published with benchmark placements produced by TW, which has produced some of the best results published on these circuits [15].

Industrial IBM circuits

The GLS heuristic has also been applied to real-life industrial IBM circuits made available by courtesy of IBM in Böblingen and Research Institute for Discrete Mathematics, University of Bonn. Three IBM circuits and the corresponding placements produced by the XQ placement program were made available.

The circuits consist of a few very large fixed modules and a large number of (mainly) small standard cells.

5.2 Parameter Settings

In the following we briefly describe the setting of all the parameters in the GLS heuristic. For a more thorough discussion and motivation of these values, we refer to [2]. To ensure that the heuristic performs well on a large range of instances, the values of the parameters were made dependent on some general characteristics of the instances. Numerous computational results indicated that the *average module area*, denoted by a, was suitable for this purpose.

The *bounding box weight* β in the objective function (3) is used to balance the contribution of the total netlength to the amount of overlap in the placement solution. As the objective function is a sum of a linear and quadratic function a variable value of β was chosen. Initially $\beta_{\text{initial}} = \sqrt{a}$ and after each FLS call we let β decrease by 1%, until some lower limit is reached. Each time a feasible placement is found, the value of β is doubled to stimulate the process of finding alternative placements. The value of the λ *parameters* in (4) determines to what degree an increased feature penalty modifies the augmented objective value. Suitable values were found to be $\lambda^{of} = \lambda^{cf} = a/10$.

With respect to the length of the connection feature memory described in Section 3.2, the best results were obtained for a very short memory of length $t = 3 \cdot$ penalty depth (cf. Section 4). A penalty depth of 1 was used on circuits with less than 100 modules while the penalty depth was 3 for larger circuits.

The minimum subproblem size as defined in Section 3.3 was set to $M = 80$. Thus fairly small subproblems were considered for large circuits.

5.3 Results

MCNC general cell circuits

The results for the small MCNC general cell circuits are presented in Table 2. For these instances we used the so-called *flat* configuration of GLS which starts from a random initial placement. A certain fixed time limit is given to the heuristic. The heuristic runs until the number of non-improving FLS calls exceeds 20,000 or the time limit is exceeded. In case the GLS heuristic terminates before the time limit is exceeded a shuffle algorithm is applied to the placement and the algorithm restarted with the shuffled placement as the initial solution. The shuffle algorithm performs a random displacement of all modules, but restricts the displacement distance to 10% of the maximum displacement distance (which is the BB length of the chip surface). Thus the randomization makes small changes to the placements which may help the algorithm to find new improving solutions.

In the last column in Table 2 we compare the minimum and average solutions of the final GLS solutions to the corresponding solutions for the O-Tree heuristic as reported by Guo et al. [5]. No results are reported on maximum O-Tree solutions. The O-Tree experiments were run on a 200MHz Ultra-1 Sparc station with 512 MB memory. No run times are reported for these results, but for similar results the authors report approximate run times of less than 1 second for Apte and Xerox, 6 seconds for Hp, 25 seconds for Ami33 and 177 seconds for Ami49.

Circuit		GLS			O-Tree	Impr
		BB short	BB medium	BB long	BB	%
Apte	Min	355,705	355,705	355,705	317,000	-12.2
	Avr	357,227	356,785	355,720	347,000	-2.5
	Max	362,373	361,440	356,088	-	-
Xerox	Min	378,044	371,997	371,121	368,000	-0.8
	Avr	440,048	409,228	384,370	426,000	9.8
	Max	617,490	409,228	384,370	-	-
Hp	Min	129,477	129,477	129,347	153,000	15.5
	Avr	131,216	130,747	130,262	163,000	20.1
	Max	140,034	140,034	131,706	-	-
Ami33	Min	43,311	43,311	39,234	51,500	23.8
	Avr	55,846	53,710	44,786	57.200	21.7
	Max	93,504	93,504	52,827	-	-
Ami49	Min	658,597	558,963	546,100	636,000	14.1
	Avr	815,189	687,773	601,861	734,000	18.0
	Max	994,846	829,094	677,851	-	-

Table 2: Results for the MCNC general cell circuits. Rows indicate the minimum, average and maximum results for 100 runs with random initial solutions. The time limits are 5 seconds (short), 10 seconds (medium) and 60 seconds (long). The last column gives the relative improvement when compared to the O-Tree solutions.

On average, GLS finds significantly better solutions for the three largest circuits within a time frame comparable to the running time of O-Tree. For the Xerox circuit, GLS was only able to improve the average solution, while for Apte, GLS finds inferior solutions. It is notable that when GLS finds better solutions than O-Tree this happens within 10 seconds. An explanation why GLS performs worse for the Apte and Xerox circuits may be the restriction to a fixed placement area (cf. Section 5.1).

MCNC standard cell circuits

For the MCNC standard cell instances high-quality placements (with regard to total BB netlength) produced by the TW heuristic were used as initial solutions for GLS. For the TW results no exact run time information is supplied. Approximate run times were obtained from Madden [9] who reported around half an hour for Biomed, 1 hour for Avqlarge and around 30 hours for Golem3 on a Sparc 10 with 512MB memory. It should be noted that the TW placements are not very suitable as initial solutions. In these placements all the modules are compressed as much as possible to the left. This creates a very dense placement which is potentially difficult for GLS to rearrange into alternative feasible placements.

The time limit was 1 hour for the small circuits and 3 hours seconds for the large circuits (cf. Table 3). Even if GLS is not designed for standard cell problems, significant improvements can be obtained on several circuits. It is interesting that for most of the circuits there are notable improvements already within the first 10 minutes, even for the large Golem3 circuit.

Industrial IBM circuits

The results on the standard cell benchmarks showed promising results for the region-based GLS heuristic with regard to improving existing placements produced by TW. Table 3 presents the results of GLS using the XQ placements as initial solutions. For each circuit we let GLS run for 12 hours and output the solution after 3, 6 and 12 hours. GLS is able to produce significant improvements on all placements. Even for the large PU circuit GLS improves the placement 6.1% within 3 hours,

Circuit	Initial	GLS			Impr
	solution	BB short	BB medium	BB long	%
Primary1	987,141	955,159	949,587	949,353	3.8
Struct	778,321	756,623	747,627	744,521	4.3
Industry1	1,866,177	1,742,048	1,695,917	1,634,801	12.4
Primary2	3,637,653	3,614,241	3,613,442	3,612,800	0.7
Biomed	3,467,190	3,457,618	3,447,534	3,442,250	0.7
Industry2	14,455,858	14,318,272	14,299,904	14,288,855	1.2
Industry3	42,652,420	42,622,856	42,612,226	42,582,937	0.2
Avqlarge	6,877,290	6,876,002	6,846,046	6,786,482	1.3
Golem3	118,572,063	118,341,850	116,347,638	113,614,220	4.2
CLK	5,286,746	4,959,800	4,933,650	4,903,923	7.2
DECODER	7,781,409	7,186,454	7,082,260	7,022,407	9.8
PU	62,368,385	58,557,990	55,763,712	52,414,317	16.0

Table 3: Results for the MCNC standard cell circuits (TW placement as initial solution) and IBM circuits (XQ placement as initial solution). For instances Primary1–Primary2 the time limits are 10 minutes (short), 30 minutes (medium) and 1 hour (long). For instances Biomed–Golem3 the time limits are 10 minutes (short), 1 hour (medium) and 3 hours (long). For instances CLK–PU the time limits are 3 hours (short), 6 hours (medium) and 12 hours (long).

and after 12 hours the placement is improved by 16.0%. It is interesting to see that larger circuits give rise to larger improvements.

In Figure 3 we show the displacements vectors for the PU circuit. A displacement is indicated by an arrow for each module which has been moved by GLS to a new location. Several interesting observations can be made from these illustrations. Note first that GLS has made many small modifications over the whole placement area. With respect to the displacements which are longer than 6% of the maximum displacement, the majority of these represent modules which have been moved across fixed macros. Most of the displacements are not parallel to the x- or y-axis, indicating that GLS has performed more than one move on the modules.

Finally, a few notes on the routability of the placements produced by GLS. The packing ability of the GLS heuristic results in placements for which dense clusters of modules appear more frequently than in the corresponding XQ placements. This could potentially cause problems for the router. In order to investigate this, we performed global routing on the PU circuit using an industrial quality router. Preliminary results indicated that the reduction in total netlength after routing was similar to the reduction given by the BB netlength. Also, the circuit was no harder to route; in fact, the number of nets that were routed by the global router decreased by more than 10%. The reason is that more nets belonged entirely to a tile in the division made by the global router.

6 Conclusion

A new local search heuristic based on Guided Local Search (GLS) has been presented for the final placement problem. The GLS heuristic differs from previous local search heuristics based on simulated annealing by having a greedy nature which makes it possible to quickly improve on existing solutions. By using Fast Local Search (FLS) it is possible to shadow parts of the solution space, thus quickly performing local improvements.

The computational comparison with the industrial codes TW and XQ showed promising results. The GLS algorithm was able to handle circuits with as many as 160,000 modules and 600,000 pins, obtaining total netlength reductions of

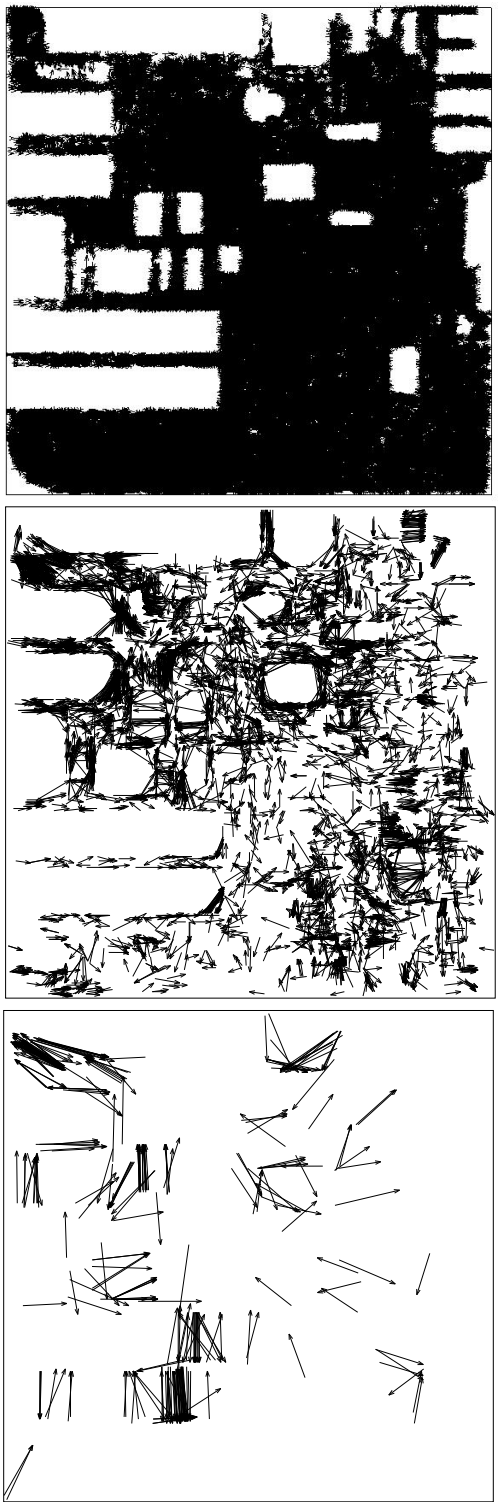

Figure 3: Displacements made by GLS on the PU circuit after 12 hours. At the top are shown the displacements from 0% to 2% of the maximum displacement. In the middle the displacements from 2% to 6% and at the bottom the displacements which are longer than 6%.

up to 20 percent. TW and XQ are both advanced placement programs — specialized for standard cell problems — which have been developed over many years. The success of the GLS heuristic should be measured from its ability to produce good results for general cell circuits, and for the capability of improving on placements produced by TW and XQ.

From a theoretical point of view the presented results are interesting as they indicate that solutions constructed by current techniques still are quite far from optimum (Figure 4). As we do not have any tight lower bounds for the placement problem it is difficult to assess the quality of the solutions proposed. High-quality solutions, obtained through local search (even at the expense of unreasonable solution times), can be used for this purpose.

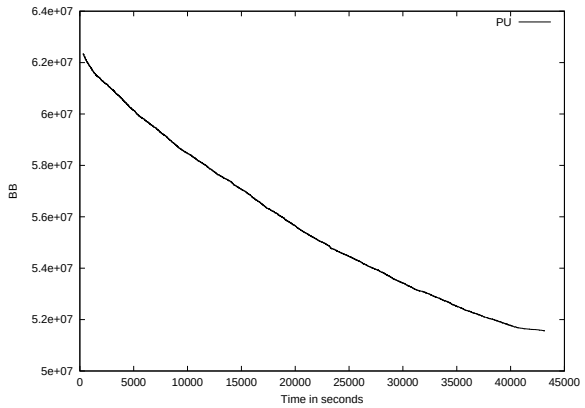

Figure 4: BB netlength as a function of time for the PU circuit.

The framework for evaluating the neighborhood in polynomial time as described in Section 4 is to our knowledge new, when applied to the placement problem. The fast evaluation of the neighborhood means that FLS terminates faster, and thus GLS can perform more iterations within the same time limit.

The GLS algorithm is still in an initial phase of development. Trimming of the parameters involved, and experimenting with other strategies for the duration of penalties may lead to additional improvements in solution quality. Also, experiments with increased solution time should be performed as the results for, e.g., the PU circuit indicate that the algorithm could improve the solution further if given additional time.

The general nature of the proposed method makes it easy to incorporate other goals, such as routability and timing constraints, into the objective function. The greatest potential of the algorithm might turn out to be its easy handling of additional objectives.

Acknowledgments

The authors would like to thank Ulrich Brenner, Karsten Muuss, Jürgen Schietke and Jens Vygen at the Research Institute for Discrete Mathematics, University of Bonn, for valuable help and fruitful discussions. Also, we would like to thank Jürgen Koehl at the IBM Research Center in Böblingen for providing industrial benchmarks circuits. The work was partially supported by SNF grant number 9701414 entitled “Experimental Algorithmics”.

References

[1] U. Brenner. Plazierung im VLSI-design. Master’s thesis, Research Institute for Discrete Mathematics, University of Bonn, 2000.

[2] O. Faroe. Placement of modules in VLSI layout. Master’s thesis, Dept. of Computer Science, University of Copenhagen, 2000.

[3] O. Faroe, D. Pisinger, and M. Zachariasen. Guided local search for the three-dimensional bin packing problem. Technical Report 99-13, Dept. of Computer Science, University of Copenhagen, 1999.

[4] Gigascale Silicon Research Center. `http://www.gigascale.org`.

[5] P.-N. Guo, C.-K. Cheng, and T. Yoshimura. An O-Tree representation of non-slicing floorplan and its applications. In *Proceedings of the 36th Design Automation Conference*, pages 268–273, 1999.

[6] D. Jepsen and C. Gelatt, Jr. Macro placement by monte carlo annealing. *Proc. IEEE Intl. Conference on Computer Design*, pages 495–498, 1983.

[7] A. Kennings. *Cell Placement Using Constructive and Iterative Improvement Methods*. PhD thesis, University of Waterloo, 1997.

[8] P. Kilby, P. Prosser, and P. Shaw. Guided local search for the vehicle routing problem. In *2nd International Conference on Metaheuristics - MIC97*, 1997.

[9] P. Madden, 2000. Personal communication.

[10] S. Mallela and L. K. Grover. Clustering based simulated annealing for standard cell placement. In *Proceedings of the 25th ACM/IEEE Design Automation Conference*, pages 312–317, 1988.

[11] C. Sechen. *VLSI Placement and Global Routing Using Simulated Annealing*. Kluwer Academic Publishers, Boston, 1988.

[12] C. Sechen and A. Sangiovanni-Vincentelli. The TimberWolf placement and routing package. *IEEE Journal of Solid-State Circuits*, SC-20(2):510–522, 1985.

[13] W.-J. Sun and C. Sechen. Efficient and effective placement for very large circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(3):349–359, 1995.

[14] W. Swartz and C. Sechen. New algorithms for the placement and routing of macro cells. *Proc. 27th Design Automation Conference*, pages 336–339, 1990.

[15] TimberWolf v.1.2. `http://www.internetcad.com`.

[16] C. Voudouris. *Guided Local Search for Combinatorial Optimization Problems*. PhD thesis, Dept. of Computer Science, University of Essex, Colchester, UK, 1997.

[17] C. Voudouris and E. Tsang. Partial constraint satisfaction problems and guided local search. In *Proceedings of Practical Application of Constraint Technology (PACT96)*, pages 337–356, 1996.

[18] C. Voudouris and E. Tsang. Fast local search and guided local search and their application to British Telecom’s workforce scheduling problem. *Operations Research Letters*, 20(3):119–127, 1997.

[19] C. Voudouris and E. Tsang. Guided local search and its application to the traveling salesman problem. *European Journal of Operational Research*, 113:469–499, 1999.

[20] J. Vygen. Algorithms for large-scale flat placement. *Proceedings of the 34th Design Automation Conference*, pages 746–751, 1997.

[21] D. Wong, H. Leong, and C. Liu. *Simulated Annealing for VLSI Design*. Kluwer Academic Publishers, 1988.