Local Search for Final Placement in VLSI Design*

Oluf Faroe, David Pisinger, Martin Zachariasen†

February 23, 2001

Abstract

The design of a VLSI circuit consists of two main parts: First, the logical functionality of the circuit is described, and then the physical layout of the modules (or cells/circuits/macros) and connections is settled. In the latter process one wishes to place the modules such that the necessary wiring becomes as small as possible in order to minimize area usage and delays on signal paths. The placement problem is the subproblem of the layout problem which considers the geometric locations of the modules.

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 placement problems have often failed as the neighbourhood 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-neighbourhoods, thus reducing the time complexity considerably.

Comprehensive computational experiments with the developed algorithm are reported on industrial circuits of small, medium and large size, 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.

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

*Tech. Rep. 2001/1, Dept. of Computer Science, University of Copenhagen

†Dept. of Computer Science, University of Copenhagen, Universitetsparken 1, DK-2100 Copenhagen Ø, Denmark. E-mail: {oluf,pisinger,martinz}@diku.dk

significant signal delay. In this paper we ignore timing issues for individual nets, since there are ways in which this problem can be addressed in a pre- or post-processing phase.

A *netmodel* estimates the length of a net after routing. A good netmodel is the length of rectilinear Steiner minimum tree (RSMT) for the pins, since in practice it is possible to route most nets with close-to minimum length. Another netmodel is the bounding box (BB) length which is the half-perimeter of the smallest axis-aligned rectangle that contains the pins. This model has the advantage of being a good estimate of the RSMT length, but much faster to compute. In this paper we use the BB netmodel — a choice that is accounted for in Section 2.

The traditional approach in placement is first to construct a *global* placement that focuses on minimizing total netlength. That is, the disjointness of the modules is to a large extent ignored. A classical approach is to use hypergraph partitioning in which the modules are recursively divided into subsets for which the need for inter-communication is minimized [1, 14]. A more recent approach is to optimize total netlength directly using the clique or star netmodel, both of which result in a quadratic optimization problem that can be solved quickly. Combined with hierarchical partitioning that takes the density of the regions into account, placements with a a limited amount of overlap can be constructed [30, 31].

The task of the *final* placement problem is to turn an infeasible placement (e.g., a global placement) into a feasible placement in which all modules are disjoint. One objective is for example to minimize the total movement of all modules with respect to the global placement. Alternatively, total netlength may be optimized directly using the BB netmodel while legalizing the placement.

In this paper we present a new iterative placement algorithm that is well-suited for solving the final placement problem. The algorithm both takes the packing problem, i.e. placing the modules disjointly, and the total BB netlength into account. The algorithm uses the *Guided Local Search (GLS)* metaheuristic [28, 29] for controlling the search. The neighbourhood structure is simple: Flipping and/or moving a single module along one of the coordinate axes. This neighbourhood has previously been used [12, 18, 21, 23, 32], 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 [28, 29], 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 [9].

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

2 The Final 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

$$\mathcal{A} = \{0, \ldots, W\} \times \{0, \ldots, H\}. \tag{1}$$

Without loss of generality we may assume that all modules and pins must be placed at integer coordinates within $\mathcal{A}$. For each module $m \in \mathcal{M}$ the corresponding width is w_m and height is h_m. Moreover let (x_m, y_m) denote the coordinate 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

$$x_m \in \{0, \ldots, W - w_m\} \qquad y_m \in \{0, \ldots, H - h_m\} \tag{2}$$

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. Let $x_{offset}(p) \in \{0, \ldots, w_m - 1\}$ denote the offset along the x-axis, and $y_{offset}(p) \in \{0, \ldots, h_m - 1\}$ denote the offset along the y-axis. Assuming that the module m has coordinates (x_m, y_m) the absolute coordinates of the pin becomes $(x_m + x_{offset}(p), y_m + y_{offset}(p))$. 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 $N \in \mathcal{N}$.

The length of the wires which are required to connect the pins for net $N \in \mathcal{N}$ is called the *netlength* of N. Thus a simple formulation of the placement problem for a circuit $\mathcal{C}$ can be stated as

$$\begin{array}{ll} \text{minimize} & \sum_{N \in \mathcal{N}} \text{netlength}(N) \\ \text{subject to} & P \text{ is a feasible placement of } \mathcal{C} \end{array} \tag{3}$$

It is not possible to determine the netlength of a given placement P without routing the chip. As routing is very time consuming and thus only will be done after the placement problem has been solved, it is necessary to use netlength estimates in the objective function. For a given placement P we let $N_P \subset \mathbb{Z}^2$ denote the *set of pin coordinates* corresponding to net $N \in \mathcal{N}$. A *netmodel* is defined as a function M which to each finite net N_P assigns a real number which estimates the actual netlength after routing. In other words, given a placement of a circuit the netmodel determines an estimated netlength for each net $N \in \mathcal{N}$.

In order to make it possible to differentiate between how much the individual nets should contribute to the objective function, we introduce a *net weight function* as $w : \mathcal{N} \to \mathbb{R}_{\geq 0}$

which to each net $N \in \mathcal{N}$ assigns some weight $w(N)$. A weight function makes it possible to assign higher weights to critical nets such that these will become shorter. Given the definition of some netmodel M we can formulate the min-sum placement problem as:

Placement Problem: *For a given circuit $\mathcal{C}$ find a placement P which minimizes the objective*

$$\sum_{N \in \mathcal{N}} w(N) \cdot M(N_P)$$

for all feasible placements P of $\mathcal{C}$.

It is clear that the choice of netmodel is crucial, as it must provide the algorithm with a good estimate of the netlength after routing. Several netmodels have been proposed in the literature, where the most important ones are: Rectilinear Steiner Minimum Tree ($RSMT$), Rectilinear Minimum Spanning Tree ($RMST$), Clique (CL), Star (ST), and Bounding Box (BB). These netmodels are illustrated in Figure 1.

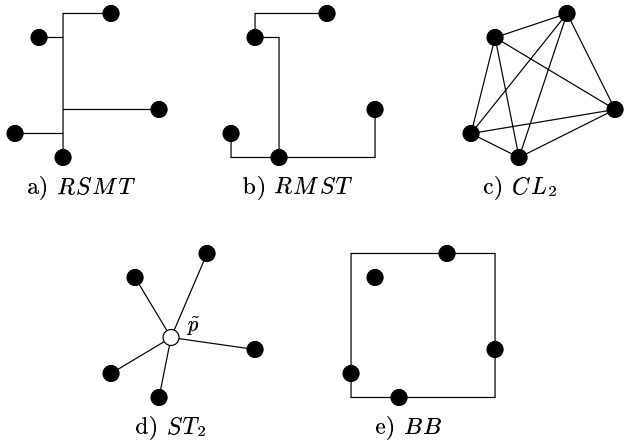

Figure 1: Five different netmodels for a finite net N_P with 5 points. The $RSMT$, $RMST$ and BB are illustrated with the L_1 metric, while CL and ST are illustrated with the L_2 metric.

Experimental results by Hetzel [10] on industrial VLSI circuits show that the $RSMT$ netmodel very closely estimates the actual netlength after routing. A drawback with the $RSMT$ netmodel is that the $RSMT$ problem is $\mathcal{NP}$-hard and thus inappropriate to use in a local search heuristic. Based on the theoretical and empirical study by Brenner and Vygen [2, 3] we chose to use the Bounding Box netmodel. This netmodel is defined as the length of the half perimeter of the smallest box which contains all the points in N_P. Due to its simplicity and good approximation to the $RSMT$ the BB netmodel is by far most popular model in the placement literature. For a set of points N_P in the plane the Bounding Box netmodel $BB(N_P)$ is formally defined as:

$$BB(N_P) = \max_{(x,y) \in N_P} x \; - \min_{(x,y) \in N_P} x \; + \max_{(x,y) \in N_P} y \; - \min_{(x,y) \in N_P} y \tag{4}$$

$BB(N_P)$ can be computed in linear time $\mathcal{O}(|N_P|)$. One of the attractive properties of the BB model is that only the points on the perimeter of the boundary box need to be known to derive the value. Another attractive property is that its deviation from $RSMT$ is bounded by

$BB(N_P) \leq RSMT(N_P)$ and $RSMT(N_P) \leq (\lceil\sqrt{|N_P|}\rceil + \frac{3}{2})/2 \cdot BB(N_P)$. For $|N_P| \leq 3$ we have $RSMT(N_P) = BB(N_P)$. These bounds were proven by Chung and Graham [4] with additional notes by Brenner and Vygen [3].

To further correct the *BB* netmodel a multiplier function $w_{\text{BB}}(N)$ may be applied. Brenner [2] empirically show that using the multiplier function $w_{\text{BB}}(N) = \frac{7}{10}|N|^{\frac{1}{4}}$ means that the Bounding Box netmodel gets very close to the *RSMT*. In our implementation, however, no multiplier function is used since this only makes a very small difference in practice [6]. Also, using no multiplier function makes it easier to compare our solution values with those from the literature.

3 Guided Local Search and Fast Local Search

In this section we introduce the metaheuristic *Guided Local Search (GLS)* and the concept of *Fast Local Search (FLS)*. The use of GLS for placement was motivated by recent results on packing problems in two and three dimensions [7]. Since a feasible placement is a packing of the modules with the additional objective of minimizing total netlength, it is natural to consider an extension of the algorithm for the packing problem.

The GLS metaheuristic has proven to be effective on a wide range of problems [15, 27, 28, 29]. GLS can be applied to any combinatorial optimization problem given by a solution space $\mathcal{X}$ for which an objective function value $f(\mathbf{x})$ and neighborhood $\mathcal{N}(\mathbf{x}) \subset \mathcal{X}$ is defined for every solution $\mathbf{x} \in \mathcal{X}$. In the following we consider minimization problems only.

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 every $i = 1, 2, \ldots, k$. 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 $\textsc{LocalOpt}_f$. In the above case we have $\mathbf{x}_k = \textsc{LocalOpt}_f(\mathbf{x}_0)$.

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 cost of a feature may be constant or variable (see Section 3.1). 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

$$h(\mathbf{x}) = f(\mathbf{x}) + \lambda \cdot \sum_{i=1}^{M} p_i \cdot I_i(\mathbf{x}) \tag{5}$$

where λ is a regularization parameter which balances the objective function to the contribution from the penalty term (see Section 3.2). 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 $\mathbf{x}^* = \textsc{LocalOpt}_h(\mathbf{x})$. Note that since all penalties initially are zero, the first local optimization 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 the features in $\mathbf{x}^*$ which have the largest contribution to the value of the objective function, but doing this in a controlled manner. The modification action therefore penalizes those features which have the maximum *utility* defined as

$$\mu_i(\mathbf{x}^*) = \frac{c_i}{1 + p_i} \cdot I_i(\mathbf{x}^*) \tag{6}$$

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 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.

When using variable feature costs, it is necessary to retract penalties dynamically; penalized features may become less unattractive as the search progresses. Several retract strategies were proposed and evaluated by Voudouris [26]. One possibility is to reset all penalties at certain intervals during the search, denoted the *Reset* strategy. Another option is to use a short term memory: record the last t features which are penalized. The memory is implemented as a circular list of length t of penalized features. As the penalty of a feature is increased it is recorded in the list at the current position, and the feature previously recorded at this position is decreased. In this way the effect of a feature penalty will have limited duration.

A problem with this strategy is that GLS only gets a limited memory. This might create cycling where GLS keeps returning to the same local minimum. To alleviate this problem Voudouris creates an identical set of features where the penalties only increase (the cost of all these features is constant). This second set is the long term memory which is applied to prevent cycling and make a diverse search of the solution space. This last strategy Voudouris coins the *Multiple Feature Sets (MFS)* strategy.

3.2 The λ Parameter

The value of λ determines to what degree an increased penalty will modify the augmented objective value and push the local search out of a local minimum. The choice of λ is problem specific. A large value of λ will make the search more aggressive to avoid solutions with penalized features and force the search to make large jumps in the solution space without

paying much attention to the original objective function f. In contrast, a small λ may require more penalties to escape a local minimum, which results in a cautious — but slower and more restricted — exploration of the solution space.

Another issue is the effect of the penalty term as more and more penalties are assigned to all the features. By looking at function (5) we see that in the long term this can cause the ratio between the original objective and the penalty term to change such that the penalty dominates the original objective function. In this respect the value of λ determines to what extent the search should be controlled by the penalties rather than by f.

3.3 Fast Local Search

A bottleneck in many applications of GLS is the LocalOpt$_h$ operation. 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. The association should enable us to know exactly which sub-neighborhoods have a direct effect upon the state of a certain feature. 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. Each local optimization using FLS will be aimed at removing the penalized features from the solution instead of exploring all possible moves.

4 Application of GLS to the Placement Problem

One of the key obstacles in applying local search heuristics to placement problems is the lack of a natural representation of a solution space and a corresponding neighbourhood function which permits a natural traversal between all feasible placements. As even the construction of a feasible solution is $\mathcal{NP}$-complete some of the constraints need to be relaxed in order to allow a simple representation of a neighbourhood. A natural choice of relaxation is to remove the constraint that no modules may overlap, and instead penalize overlap in the objective function. A similar solution space was introduced by Jepsen et al. [12] for the placement problem and by Dowsland [5] and Faroe et al. [7] in the context of packing.

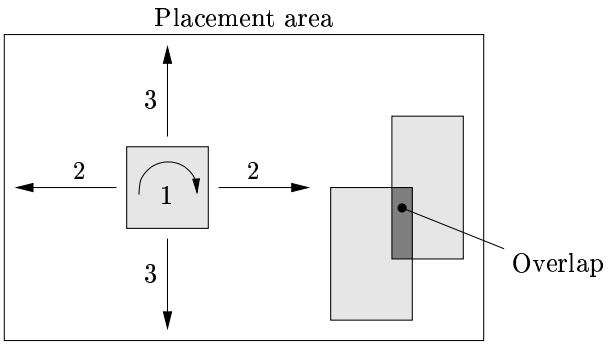

Figure 2: Illustration of possible moves of a single module. The moves are (1) change of orientation combined with either (2) translation along the x-axis or (3) translation along the y-axis. Modules are allowed to overlap.

The problem handed to the GLS heuristic is thus a *Relaxed Placement Problem* where all the modules must be placed within the placement area and the objective is to minimize total netlength as well as the overlap between individual modules.

To be more formal, we define the solution space $\mathcal{X}$ as all possible orientations and positions of the (free) modules $m \in \mathcal{M}$ within the placement area. For a placement solution $\mathbf{x} \in \mathcal{X}$ let $x_m(\mathbf{x})$ and $y_m(\mathbf{x})$ denote the coordinates for modules $m \in \mathcal{M}$. Thus, the domain of feasible coordinates is given as:

$$x_m(\mathbf{x}) \in \{0, \ldots, W - w_m\} \qquad y_m(\mathbf{x}) \in \{0, \ldots, H - h_m\} \tag{7}$$

Although not explicitly formulated, the modules may have eight different orientations as described in Section 2. In this case the width and height of the module and the pin offsets are updated accordingly.

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. The possible moves of a single module are illustrated on Figure 2. The neighbourhood $\mathcal{N}(\mathbf{x})$ thus satisfies that it is possible to traverse between any pair of solutions by following a path of neighboring solutions. The neighbourhood size $\mathcal{O}(|\mathcal{M}|(W+H))$ is *not* polynomial in the input size, but with the application of FLS described in Section 4.5 and a neighborhood reduction scheme described in Section 4.6, a best neighboring solution can be found in polynomial time. A similar neighborhood function was used by Faroe et al. [7] for the three-dimensional bin packing problem.

4.1 Objective Function

As discussed earlier the objective in the placement problem is to minimize total netlength while minimizing overlap between modules. By using the *BB* netmodel as an estimate for the netlength (see Section 2), the objective value $f(\mathbf{x})$ of a given solution $\mathbf{x} \in \mathcal{X}$ may be defined as a linear combination of the two terms:

$$f(\mathbf{x}) = \sum_{m,n \in \mathcal{M}} \text{overlap}_{mn}(\mathbf{x}) + \beta \sum_{N \in \mathcal{N}} BB_N(\mathbf{x}) \tag{8}$$

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. The first sum in (8) 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 final 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. A reduction in the overlap term will nearly always increase the bounding box value and vice versa. As the bounding box term is linear while the overlap term is quadratic, Upton et al. [25] note that this makes it difficult to assign a suitable value to β. A linear overlap function might therefore have been more suitable (e.g., the perimeter of the rectangular intersection), but experiments with the two alternative formulations indicated that the quadratic overlap function resulted in a faster convergence towards feasible placement solutions. The actual setting of β is discussed in Section 5.2.

4.2 Features

Two sets of features are defined, each reflecting the contribution of the overlap and the contribution of the total netlength in the objective function. The first contribution is characterized through an *overlap feature* and the second through a *connection feature*.

An *overlap feature* is defined for each pair of modules $m, n \in \mathcal{M}$. For a given pair of modules $m, n \in \mathcal{M}$ a particular solution $\mathbf{x} \in \mathcal{X}$ exhibits an *overlap feature* if the modules overlap. The presence of the feature is given by the indicator function:

$$I^{of}_{mn}(\mathbf{x}) = \begin{cases} 1 & \text{if } \text{overlap}_{mn}(\mathbf{x}) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

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 [7] 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. We define the overlap feature cost as:

$$c^{of}_{mn}(\mathbf{x}) = \begin{cases} \text{overlap}_{mn}(\mathbf{x}) + \text{area}(m) + \text{area}(n) & \text{if } \text{overlap}_{mn}(\mathbf{x}) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

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 (6).

A *connection feature* is defined for each pair of modules $m, n \in \mathcal{M}$. 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. The presence of the connection feature is given by the indicator function:

$$I^{cf}_{mn}(\mathbf{x}) = \begin{cases} 1 & \text{if } \text{rdist}_{mn}(\mathbf{x}) > 0 \text{ and } m, n \text{ connected} \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

where the rectilinear distance between two modules m and n is defined as

$$\text{rdist}_{mn}(\mathbf{x}) = \text{dx}_{mn}(\mathbf{x}) + \text{dy}_{mn}(\mathbf{x}) \tag{12}$$

Here $\text{dx}_{mn}(\mathbf{x})$ is the x-distance between modules m and n measured as the minimum distance between a gridpoint in m and a gridpoint in n. $\text{dy}_{mn}(\mathbf{x})$ is defined in a similar way with respect to the y-distances.

For modules m and n the connection feature *cost* c^{cf}_{mn} should somehow 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 of 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:

$$c^{cf}_{mn}(\mathbf{x}) = \text{rdist}_{mn}(\mathbf{x}) \tag{13}$$

Again, the utility function is given by (6).

4.3 Augmented Objective Function

We can now state the *augmented objective function* for the relaxed placement problem as:

$$h(\mathbf{x}) = f(\mathbf{x}) + \lambda^{of} \sum_{m,n \in \mathcal{M}} p^{of}_{mn} \cdot I^{of}_{mn}(\mathbf{x}) + \lambda^{cf} \sum_{m,n \in \mathcal{M}} p^{cf}_{mn} \cdot I^{cf}_{mn}(\mathbf{x}) \tag{14}$$

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.

As the neighbourhood $\mathcal{N}(\mathbf{x})$ 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. This means that the penalty term for the connection features in the augmented objective function becomes

$$\lambda^{cf} \sum_{m,n \in \mathcal{M}} p^{cf}_{mn} \cdot (I^{cfx}_{mn}(\mathbf{x}) + I^{cfy}_{mn}(\mathbf{x})) \tag{15}$$

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.

4.4 Feature Costs and Duration of Penalties

The augmented objective function (14) 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 (14). 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 Section 3.1. This allows the search to escape from the current minimum and make a diverse exploration of the solution space.

The *connection features* are soft and thus some techniques are needed to restrict the duration of the penalties as described in Section 3. Both the *Reset* and the *Multiple Feature Sets (MFS)* strategies were implemented but without any major success. Instead a new strategy was developed based on a refinement of the MFS strategy. As described in Section

3.1, 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

$$\mu_i(\mathbf{x}^*) = \frac{c_i}{1 + f_i} \cdot I_i(\mathbf{x}^*) \tag{16}$$

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

4.5 Fast Local Search

To apply *Fast Local Search (FLS)* 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 module, and 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.

In the worst case the evaluation of the feature utilities (6) may demand $\mathcal{O}(|\mathcal{M}|^2)$ time, which for large circuits can be very time consuming. Thus 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.

4.6 Fast Evaluation of the Neighbourhood

A bottleneck in the FLS algorithm is the size of the sub-neighborhoods. Each sub-neighborhood corresponds to the solutions that can be obtained by changing the orientation of a single module $m \in \mathcal{M}$ and moving it along one of the coordinate axes. A sub-neighbourhood can be evaluated in pseudo-polynomial time $\mathcal{O}((|\mathcal{M}| + |\mathcal{N}| + |\mathcal{P}|)(W + H))$.

The following description will only cover translation of a module $m \in \mathcal{M}$ along the x-axis as a symmetrical result can be obtained by considering the other axis. We will reduce the complexity of the evaluation to a polynomial expression by observing that the objective function is a piecewise linear function in x_m. This piecewise linear function will take its extreme values in one of the breakpoints or discontinuities. A necessary property is that

the functions considered take the smallest value of the two possible function values in the discontinuities. This property is satisfied for the augmented objective function (14). For module $m \in \mathcal{M}$ the problem to be solved in FLS is

$$\min_{x_m \in \{0,\ldots,W-w_m\}} A(\mathbf{x}) + B(\mathbf{x}) + C(\mathbf{x}) + D(\mathbf{x}) \tag{17}$$

where

$$A(\mathbf{x}) = \sum_{n \in \mathcal{M}, n \neq m} \text{overlap}_{mn}(\mathbf{x}) \qquad B(\mathbf{x}) = \beta \sum_{N \in \mathcal{N}_m} BB_N(\mathbf{x})$$

$$C(\mathbf{x}) = \lambda^{of} \sum_{n \in \mathcal{M}, n \neq m} p^{of}_{mn} \cdot I^{of}_{mn}(\mathbf{x}) \qquad D(\mathbf{x}) = \lambda^{cf} \sum_{n \in \mathcal{M}, n \neq m} p^{cf}_{mn} \cdot I^{cfx}_{mn}(\mathbf{x})$$

Here $\mathcal{N}_m \subset \mathcal{N}$ is the subset of nets which contain a pin of module m. $A(\mathbf{x})$ is the amount of overlap between m and the other modules, $B(\mathbf{x})$ is the bounding box of the nets which contain a pin of module m, $C(\mathbf{x})$ is the contribution of overlap penalties, and finally $D(\mathbf{x})$ is the contribution of connection penalties along the x-axis.

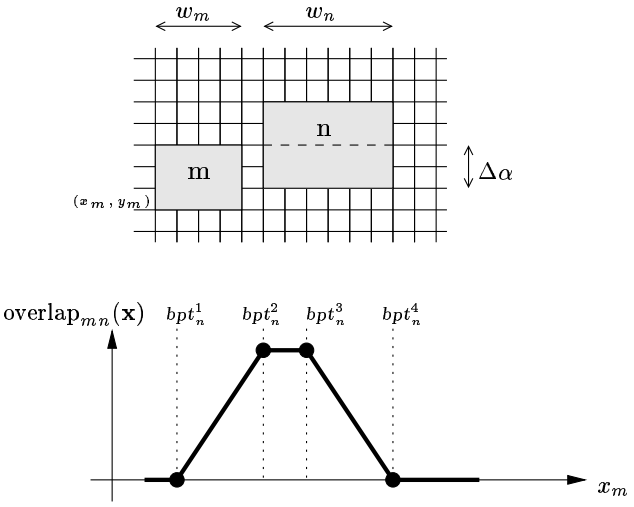

Figure 3: Illustration of the overlap breakpoints. The topmost illustration shows two modules, m and n, which overlap as m is translated along the x-axis. Below is seen the overlap function as m is translated to the right. In the figure $\Delta\alpha$ denotes the height of the overlap. The breakpoints are indicated with dashed lines.

Each of the four terms is a piecewise linear function with possible discontinuities. For the overlap term $A(\mathbf{x})$ the linearity follows from the fact that the overlap of two modules m, n grows linearly as module m is translated along the x-axis (see Figure 3) until the two modules overlap completely or the two modules are disjoint. For the bounding box term $B(\mathbf{x})$ the linearity follows from the fact that as long as the pin in module m is on the perimeter of the bounding box the term will grow linearly as module m is translated along the x-axis (see Figure 4). When the pin in module m is inside the bounding box rectangle, the bounding box term remains unchanged. It can easily be verified in a similar way that the piecewise linearity also holds for the last two terms $C(\mathbf{x})$ and $D(\mathbf{x})$. The sum of piecewise linear functions is

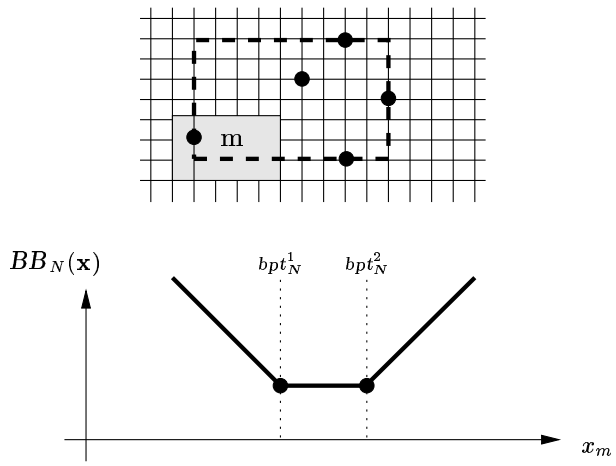

Figure 4: Illustration of the net breakpoints. The topmost illustration shows module m as the shaded box. The pins in net N are marked as black dots and the corresponding bounding box drawn using dashed lines. Below the function $BB_N(\mathbf{x})$ is illustrated as module m is translated along the x-axis.

obviously also a piecewise linear function where the breakpoints and discontinuities correspond to the union of the breakpoints and discontinuities of the individual terms.

For the overlap term $A(\mathbf{x})$, each module $n \neq m$ gives rise to four breakpoints as illustrated in Figure 3, thus the term contributes with at most $4|\mathcal{M}|$ points. The bounding box term $B(\mathbf{x})$ gives rise to two breakpoints for each module $n \neq m$ thus in total we get at most $2|\mathcal{M}|$ points from this term. The last two terms $C(\mathbf{x})$ and $D(\mathbf{x})$ each contribute with $2|\mathcal{M}|$ breakpoints. As we also need to consider the end points of the domain of module m we need to evaluate the objective function in at most $10|\mathcal{M}|+2$ points. To find the minimum of the objective function (17) we sort the breakpoints according to the x-position and then evaluate the objective value in each of the points in constant time per. breakpoint as follows: Starting at the current position of module m we consider the breakpoints to the right of m in increasing order of x-coordinates. Keeping track on the current objective value, the increment in the objective can be determined in constant time for each breakpoint by maintaining the sum of the slopes of the individual terms in the objective. The process is then repeated by moving m to the left. In total the minimum of (17) can be found in time $O((|\mathcal{M}|+|\mathcal{N}|+|\mathcal{P}|)\log(|\mathcal{M}|+|\mathcal{N}|+|\mathcal{P}|))$ where the most expensive part is the sorting of the points.

4.7 Problem Subdivision for Large Problems

Although much effort has been done to reduce the complexity of each iteration of FLS, 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 [13].

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.

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. Of the three layout styles, the general cell layout best explores the potential of the GLS heuristic with respect to the problem definition given in Section 2. General cell circuits contain rectangular modules of various dimensions, which makes it difficult to produce even feasible placements for circuits with only a few modules. This challenges the inherited packing ability of the GLS heuristic [7].

In order to evaluate the placements produced by GLS three different placement heuristics are used as benchmarks: O-Tree [9], TimberWolf v.1.2 commercial edition (TW) [24] and XQ [30]. The core in TW is a simulated annealing placement heuristic which is specialized for standard cell placement [22]. The XQ placement program uses quadratic optimization and sophisticated partitioning.

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 Computational Environment and 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 LEDA library [16] was used to produce all the graphical output and to some degree to implement complex data types. Circuit data was represented using the XI data model [19], made available by courtesy of the Research Institute for Discrete Mathematics, University of Bonn. To implement the sparse matrix representation, the SparseLib++ [20] and IML++ [11] libraries from NIST were used.

The following benchmark circuits were used in the experiments:

- **MCNC general cell circuits**

  These circuits were considered in a recent paper published on the O-Tree heuristic [9]; 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.

There are five circuits in the MCNC general cell benchmark set. Table 1 describes the characteristics of the circuits. All 8 orientations are possible for the modules and the placement area contains no blockages nor fixed modules.

Circuit	Modules		Blockages	Pins		Nets
	Free	Fixed		Non-I/O	I/O	
Apte	9	0	0	287	73	97
Xerox	10	0	0	698	2	183
Hp	11	0	0	309	45	71
Ami33	33	0	0	520	40	122
Ami49	49	0	0	953	22	396

Table 1: Characteristics of the MCNC general cell circuits.

- **MCNC standard cell circuits**

  The MCNC standard cell circuits were obtained from the recent GSRC release [8]. The circuits are published with benchmark placements produced by TW, which has produced some of the best results published on these circuits [24]. Table 2 describes the characteristics of the MCNC standard cell circuits. Only the basic orientation of the modules is allowed — that is, no rotation nor flipping is allowed.

Circuit	Modules		Blockages	Pins		Nets
	Free	Fixed		Non-I/O	I/O	
Fract	125	0	0	463	24	147
Primary1	752	0	0	2,941	130	903
Struct	1,888	0	0	5,471	64	1,920
Industry1	2,271	0	0	8,837	814	2,593
Primary2	2,907	0	0	11,226	188	3,029
Biomed	6,417	0	0	21,040	97	5,742
Industry2	12,142	0	0	48,404	495	13,419
Industry3	15,059	0	0	68,418	374	21,940
Avqlarge	25,114	0	0	82,752	64	25,385
Golem3	99,932	0	0	338,623	2768	144,950

Table 2: Characteristics of the MCNC Standard Cell Circuits. The first five instances are denoted *small* instances, while the remaining are denoted *large* instances.

- **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. Table 3 describes some of the characteristics of these circuits. The circuits consist of a few very large fixed modules and a large number of (mainly) small standard cells.

The layout of the CLK, DECODER and PU circuits is different from the circuits described earlier. Firstly, all the modules must be placed on coordinates which correspond to standard cell rows — even if their height exceeds the standard cell row height. Secondly, every other row is flipped. That is, only two orientations are allowed on non-flipped rows: The basic orientation and flipping around the y-axis. But on every other row the modules are also flipped around the x-axis. This is a technological constraint which is necessary in order to take into account the power supply. Since this constraint is not implemented in the GLS heuristic it was necessary to run an additional legalization program on the placements produced by the GLS heuristic. This legalization program flips the modules such that they are oriented correctly — at the cost of increased netlength.

Circuit	Modules		Blockages	Pins		Nets
	Free	Fixed		Non-I/O	I/O	
CLK	29,056	42	354	111,902	414	30,293
DECODER	54,911	17	19	188,275	1,016	59,256
PU	163,960	96	550	617,190	744	184,231

Table 3: Characteristics of the IBM circuits.

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 [6]. 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 $\bar{a}$, was suitable for this purpose.

The *bounding box weight* β in the objective function (8) 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 $\beta_{\text{initial}} = \sqrt{\bar{a}}$ 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 (14) determines to what degree an increased feature penalty modifies the augmented objective value. Suitable values were found to be $\lambda^{of} = \lambda^{cf} = \bar{a}/10$.

With respect to the length of the connection feature memory described in Section 4.4, the best results were obtained for a very short memory of length $t = 3 \cdot$ penalty depth (cf. Section 4.5). 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 4.7 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 general cell circuits described in Table 1 are presented in Table 4. 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. Thus the randomization makes small changes to the placements which may help the algorithm to find new improving solutions.

Circuit		GLS				O-Tree	Impr.
		BB_{5sec}	BB_{10sec}	BB_{30sec}	BB_{60sec}	BB	%
Apte	Min	355,705	355,705	355,705	355,705	317,000	-12.2
	Avr	357,227	356,785	355,925	355,720	347,000	-2.5
	Max	362,373	361,440	360,730	356,088	-	-
Xerox	Min	378,044	371,997	371,121	371,121	368,000	-0.8
	Avr	440,048	*409,228*	*389,176*	*384,370*	426,000	9.8
	Max	617,490	409,228	389,176	384,370	-	-
Hp	Min	*129,477*	*129,477*	*129,449*	*129,347*	153,000	15.5
	Avr	*131,216*	*130,747*	*130,318*	*130,262*	163,000	20.1
	Max	140,034	140,034	131,706	131,706	-	-
Ami33	Min	*43,311*	*43,311*	*40,421*	*39,234*	51,500	23.8
	Avr	*55,846*	*53,710*	*47,217*	*44,786*	57.200	21.7
	Max	93,504	93,504	60,967	52,827	-	-
Ami49	Min	658,597	*558,963*	*551,161*	*546,100*	636,000	14.1
	Avr	815,189	*687,773*	*622,558*	*601,861*	734,000	18.0
	Max	994,846	829,094	696,708	677,851	-	-

Table 4: Results for the MCNC general cell circuits. Rows indicate the minimum, average and maximum results for 100 runs with random initial solutions. Emphasized entries indicate an improved solution value as compared to O-Tree.

For each circuit 100 runs were performed with a time limit of 60 seconds. The results are sampled at different breakpoints and the "GLS" column in Table 4 shows the minimum, average and maximum total netlength after 5, 10, 30 and 60 seconds, respectively.

In the last column 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. [9]. 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.

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 does not find as good solutions as O-Tree. 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 given in Table 2, high-quality placements produced by the TW and XQ heuristics exist. Our initial experiments using GLS as a stand-alone heuristic with the initial solution provided by a simple quadratic optimization and partitioning algorithm were not successful, and are therefore not reported here (for details see [6]). Instead we report on improving the placement provided by TW using the *region-based* GLS algorithm described in Section 4.7.

For the TW results no exact run time information is supplied. Approximate run times were obtained from Madden [17] who reported around 30 hours for the Golem3, an order of an hour for Avqlarge, half an hour for Biomed and 1-2 minutes for Fract on a Sparc 10 with 512MB memory. For the XQ placer the run times were significantly smaller, in the order of minutes for the small instances and a few hours for the Golem3 instance on an Intel Pentium II 500 Mhz with 1GB 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.

In Table 5 we present detailed results for the smallest Fract instance. For this instance we used random initial placements and a time limit of 300 seconds. The time limit in the other instances, which used the TW placement as the initial solution, was 3,600 seconds for the small circuits and 10,800 seconds for the large circuits (cf. Table 2). These results are presented in Table 6 and 7. 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 600 seconds, even for the large Golem3 circuit.

Table 6 and 7 also provide information on the improvements over the placements produced by XQ, which were slightly inferior to the TW placements. It is important that the reader does not use these results to compare TW and XQ and conclude that TW in general produces better results than XQ. The primary reasons are that due to conversion issues the TW results may have changed, and that the XQ results were produced on a very preliminary Linux version which was not stable. The XQ program is also aimed at much larger circuits and is not expected to perform well on such "small" instances.

Circuit		GLS				Impr_{TW}
		$BB_{25\text{sec}}$	$BB_{50\text{sec}}$	$BB_{150\text{sec}}$	$BB_{300\text{sec}}$	%
Fract	Min	*62,431*	*62,378*	*61,191*	*61,191*	5.3
	Avr	67,328	65,720	*63,940*	*63,042*	2.4
	Max	73,718	73,164	70,847	66,835	-3.4

Table 5: Results for the Fract circuit. Rows indicate the minimum, average and maximum results for 50 runs with random initial placements.

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 or XQ. In this section GLS is applied to real-life placements produced by XQ for the three IBM circuits

Circuit	GLS			Impr_{TW}	Impr_{XQ}
	$BB_{600\text{sec}}$	$BB_{1800\text{sec}}$	$BB_{3600\text{sec}}$	%	%
Primary1	955,159	949,587	949,353	3.8	16.6
Struct	756,623	747,627	744,521	4.3	3.8
Industry1	1,742,048	1,695,917	1,634,801	12.4	6.4
Primary2	3,614,241	3,613,442	3,612,800	0.7	11.8

Table 6: Results for the small MCNC standard cell circuits with the TW placement as the initial solution.

Circuit	GLS			Impr_{TW}	Impr_{XQ}
	$BB_{600\text{sec}}$	$BB_{3600\text{sec}}$	$BB_{10800\text{sec}}$	%	%
Biomed	3,457,618	3,447,534	3,442,250	0.7	20.3
Industry2	14,318,272	14,299,904	14,288,855	1.2	15.5
Industry3	42,622,856	42,612,226	42,582,937	0.2	12.6
Avqlarge	6,876,002	6,846,046	6,786,482	1.3	20.6
Golem3	118,341,850	116,347,638	113,614,220	4.2	4.3

Table 7: Results for the large MCNC standard cell circuits with the TW placement as the initial solution.

CLK, DECODER and PU described in Table 3.

For each circuit we let GLS run for 12 hours and output the solution after 3, 6 and 12 hours. As described in Section 5.1, there are some additional technological constraints on the placements of the IBM circuits which are not implemented in GLS. Thus, it was necessary to perform a legalization of the GLS placements by swapping the orientation of some modules which again caused an increase in the netlength. The increase in netlength due to the legalization was between 3% and 4% for CLK, between 4% and 5% for DECODER and less than 2% for PU.

Table 8 compares the results of GLS (after the legalization) using the XQ placements as initial solutions. 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, and after 12 hours the placement is improved by 16.0%. It is interesting to see that larger circuits give rise to larger improvements.

Figure 5 plots the development of the improving solutions over the 12 hours of running time. Solution values before legalization are plotted. For the smallest circuit, CLK, the GLS heuristic converges very quickly and most of the improvement happens within the first 4 hours. For DECODER the improvement converges more slowly, but GLS still manages to produce most of the improvements within the time frame of 12 hours. For the PU circuit the plot shows that 12 hours are not enough for GLS to converge properly.

In Figure 6 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.

Circuit	GLS					
	BB_{3h}	Impr.	BB_{6h}	Impr.	BB_{12h}	Impr.
CLK	4,959,800	6.2%	4,933,650	6.7%	4,903,923	7.2%
DECODER	7,186,454	7.6%	7,082,260	9.0%	7,022,407	9.8%
PU	58,557,990	6.1%	55,763,712	10.6%	52,414,317	16.0%

Table 8: Results for the IBM circuits with the XQ placement as the initial solution.

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 up to 20 percent. Obviously, it is unrealistic to expect the GLS heuristic to outperform the other heuristics on *all* the benchmark circuits. This is especially true when it comes to producing placements of the large standard cell circuits, where both TW and XQ produce high quality solutions. 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 instead 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 generated by current techniques still are quite far from optimum. As we do not have any tight lower bounds for the final 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.

The framework for evaluating the neighbourhood in polynomial time as described in Section 4.6 is to our knowledge new, when applied to the final placement problem. The Bounding Box netmodel lends itself well to this approach, as opposed to the other models considered in Section 2. The fast evaluation of the neighbourhood means that FLS terminates faster, and thus GLS can perform more iterations within the same time limit.

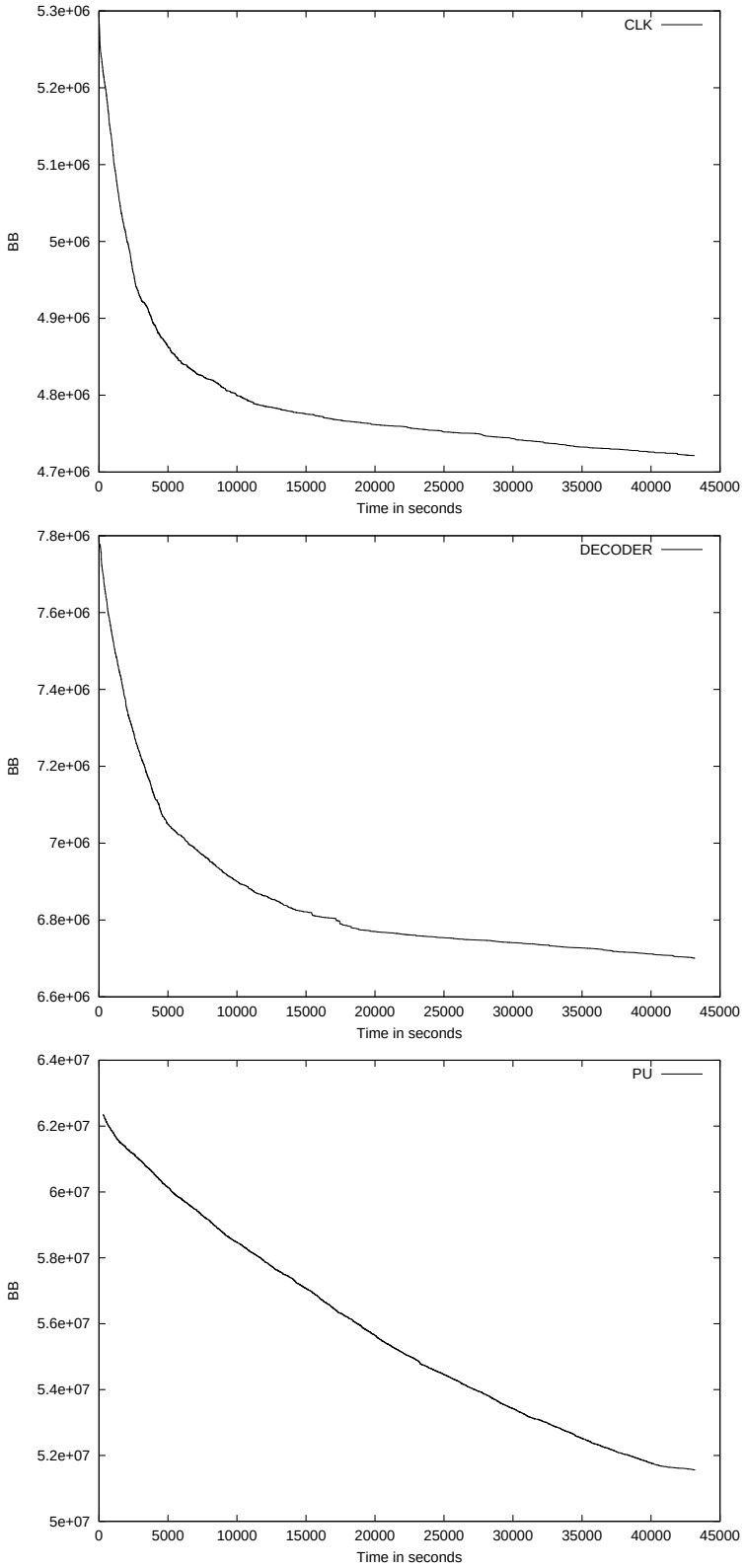

Figure 5: Plots which show the improving solution of GLS for the CLK, DECODER and PU circuit. The y-axis measures the total BB netlength in grid line units before legalization.

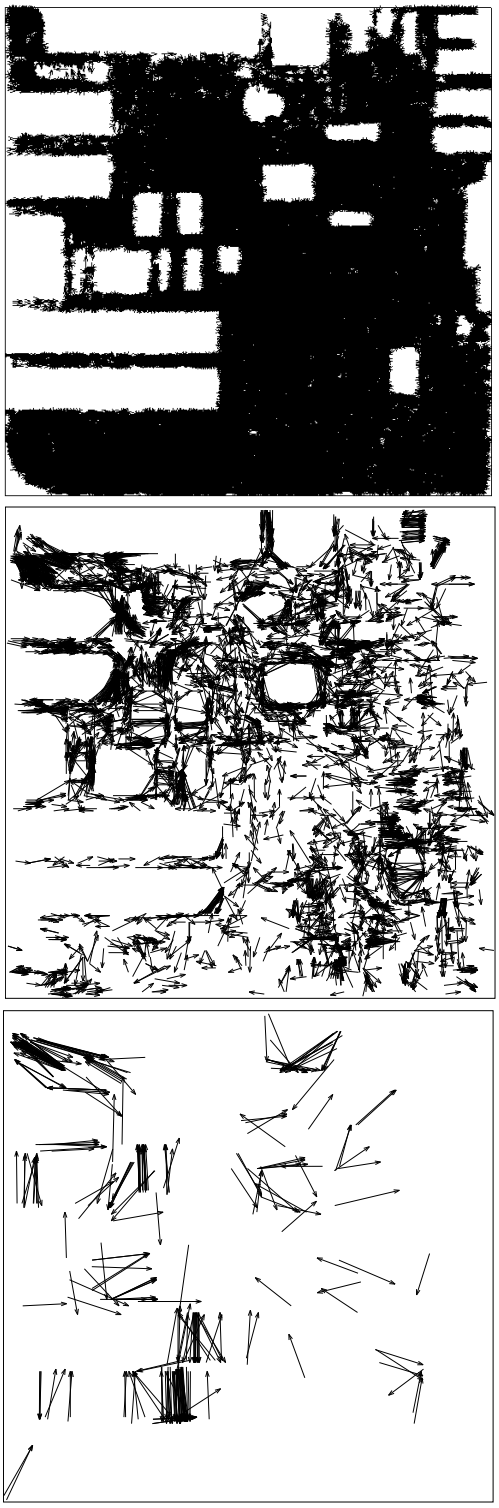

Figure 6: Displacements made by GLS on the PU circuit after 12 hours. At the top are shown the displacements from 0% to 2% of the maximum displacement. In the middle the displacements from 2% to 6% and at the bottom the displacements which are longer than 6%.

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. A final topic of further research is to verify that the routability of the GLS solutions is not worse compared to other algorithms. As described in Section 5.3, the preliminary results seem promising.

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 Köhl 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] C. J. Alpert, J.-H. Huang, and A. B. Kahng. Multilevel circuit partitioning. In *Proceedings of the 34th ACM/IEEE Design Automation Conference*, pages 530–533, 1997.

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

[3] U. Brenner and J. Vygen. Worst-case ratios of networks in the rectilinear plane. Technical Report 00904, Research Institute for Discrete Mathematics, University of Bonn, 2000.

[4] F.R.K. Chung and R.L. Graham. On Steiner trees for bounded point sets. *Geom. Dedicata*, 11:353–361, 1981.

[5] K. Dowsland. Some experiments with simulated annealing techniques for packing problems. *European Journal of Operational Research*, 68:389–399, 1993.

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

[7] 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.

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

[9] 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.

[10] A. Hetzel. *Verdrahtung im VLSI-Design: Spezielle Teilprobleme und ein sequentielles Lösungsverfahren.* PhD thesis, Research Institute for Discrete Mathematics, University of Bonn, 1995.

[11] IML++ v. 1.2a. `http://math.nist.gov/iml++/`.

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

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

[14] B.W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49:291–307, 1970.

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

[16] LEDA v. 4.1. `http://www.mpi-sb.mpg.de/LEDA/`.

[17] Patrick Madden, 2000. Personal communication.

[18] 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.

[19] Karsten Muuss. *XI Data Model.* Research Institute for Discrete Mathematics, University of Bonn, 1996.

[20] NIST SparseLib v. 1.5c. `http://math.nist.gov/sparselib++/`.

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

[22] 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.

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

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

[25] M. Upton, K. Samii, and S. Sugiyama. Integrated placement for mixed macro cell and standard cell designs. In *27th ACM/IEEE Design Automation Conference*, pages 32–35, 1990.

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

[27] 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.

[28] 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.

[29] 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.

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

[31] J. Vygen. *Plazierung im VLSI-Design und ein zweidimensionales Zerlegungsproblem.* PhD thesis, Research Institute for Discrete Mathematics, University of Bonn, 1997.

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