DECEMBER 1996

WRL Research Report 96/4

Optimizations and Placements with the Genetic Workbench

Silvio Turrini

digital **Western Research Laboratory** 250 University Avenue Palo Alto, California 94301 USA

The Western Research Laboratory (WRL) is a computer systems research group that was founded by Digital Equipment Corporation in 1982. Our focus is computer science research relevant to the design and application of high performance scientific computers. We test our ideas by designing, building, and using real systems. The systems we build are research prototypes; they are not intended to become products.

There are two other research laboratories located in Palo Alto, the Network Systems Lab (NSL) and the Systems Research Center (SRC). Another Digital research group is located in Cambridge, Massachusetts (CRL).

Our research is directed towards mainstream high-performance computer systems. Our prototypes are intended to foreshadow the future computing environments used by many Digital customers. The long-term goal of WRL is to aid and accelerate the development of high-performance uni- and multi-processors. The research projects within WRL will address various aspects of high-performance computing.

We believe that significant advances in computer systems do not come from any single technological advance. Technologies, both hardware and software, do not all advance at the same pace. System design is the art of composing systems which use each level of technology in an appropriate balance. A major advance in overall system performance will require reexamination of all aspects of the system.

We do work in the design, fabrication and packaging of hardware; language processing and scaling issues in system software design; and the exploration of new applications areas that are opening up with the advent of higher performance systems. Researchers at WRL cooperate closely and move freely among the various levels of system design. This allows us to explore a wide range of tradeoffs to meet system goals.

We publish the results of our work in a variety of journals, conferences, research reports, and technical notes. This document is a research report. Research reports are normally accounts of completed research and may include material from earlier technical notes. We use technical notes for rapid distribution of technical material; usually this represents research in progress.

Research reports and technical notes may be ordered from us. You may mail your order to:

Technical Report Distribution
DEC Western Research Laboratory, WRL-2
250 University Avenue
Palo Alto, California 94301 USA

Reports and technical notes may also be ordered by electronic mail. Use one of the following addresses:

Digital E-net:	`JOVE::WRL-TECHREPORTS`
Internet:	`WRL-Techreports@decwrl.pa.dec.com`
UUCP:	`decpa!wrl-techreports`

To obtain more details on ordering by electronic mail, send a message to one of these addresses with the word ‘‘`help`’’ in the Subject line; you will receive detailed instructions.

Reports and technical notes may also be accessed via the World Wide Web: `http://www.research.digital.com/wrl/home.html`.

Optimizations and Placements with the Genetic Workbench

Silvio Turrini

December 1996

digital **Western Research Laboratory** 250 University Avenue Palo Alto, California 94301 USA

Abstract

The Genetic Workbench (GWB) is a software system built with the intent of investigating evolutionary or non-standard algorithms applied to difficult combinatorial problems. The user is allowed to experiment with various techniques, operators, parameters, strategies and compare the results. In particular the optimal placements of connected components or modules on a plane has been considered, but some of the strategies implemented in the GWB can be applied to other permutation based problems as well. Techniques which generate the best results have also been compared with one of the best commercial tools available, TimberWolf ver. 7, which uses a special simulated annealing algorithm, to highlight the strengths and weaknesses of the different methods. Most of the strategies used in the GWB can be classified as evolutionary or rely on some implementation of a genetic algorithm; this is the reason why the qualifier genetic has been used to name the system. For the placement problem in particular, results of running standard benchmarks are also shown at the end of this report.

What is the Genetic Workbench ?

The Genetic Workbench (GWB) is a system that lets users apply various optimization techniques to specific problems in order to quickly determined the most efficient strategy or the optimal set of parameters that work the best for the case under investigation. Four main optimization engines are part of the tool and can be activated by the user and applied to the problem either individually or in various combinations and order. The system has been written in C++ and runs on DEC workstations and Alpha platforms [see Fig. 1] . Blocks delimited by a solid line are fixed parts of the system and cannot be changed without rewriting the existing code and recompiling the entire system. The shaded block delimited by a dashed line can be changed or added to the system and describes the problem under investigation in terms of an objective function with a certain number of parameters to be optimized. All the algorithms (Clustering, Genetic Engine, etc. etc.) assume an order-base representation of the problem or, as it will be clarified later in this report, can alternatively have a vector representation of its parameters and use special transformations to map them into a permutation space. At this stage of the project there is no sophisticated graphical user interface and options are provided by the standard mechanism in Unix of command line options following the invocation of the program. When dealing with the optimal placement problem, the system uses its own internal format, but routines to interface more popular netlist formats, such as the one used by TimberWolf ver. 7 or higher, have also been developed and can be used for conversions.

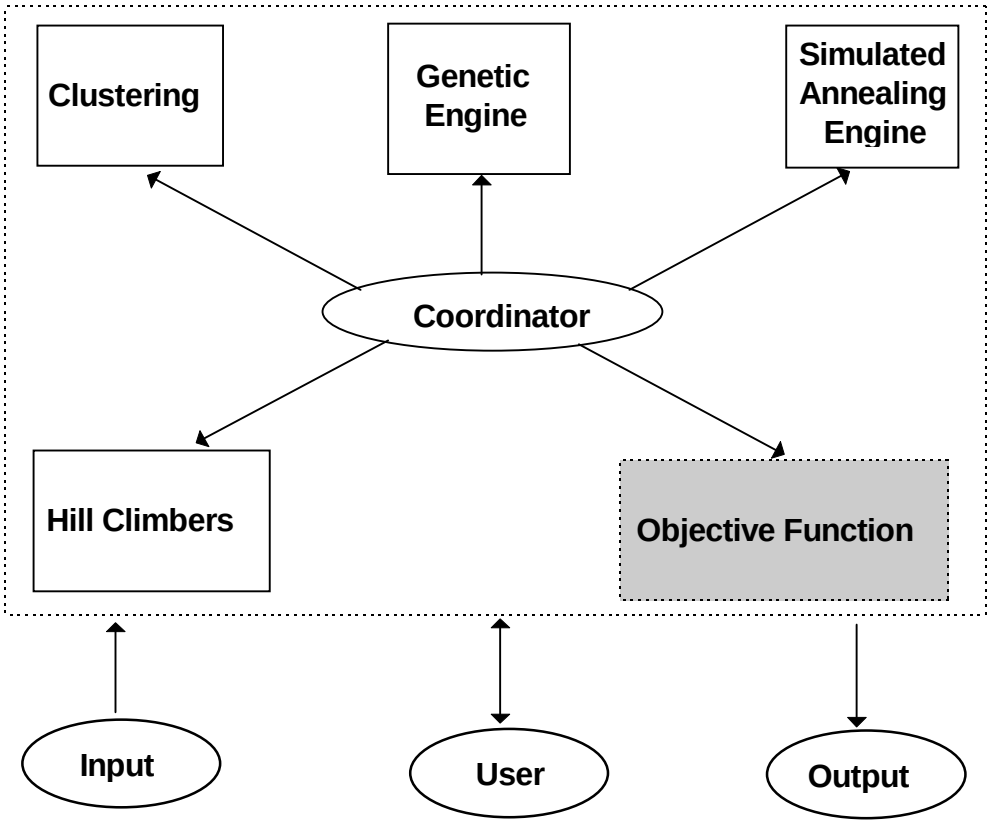


Fig. 1 : block diagram of the system

The diagram in Fig. 1 also shows the four main optimization engines that can be used to optimize a specific objective function either independently or called and subsequently applied to the problem in a user defined order. The engines that use genetic algorithms, simulated annealing techniques and hill-climbers should be familiar to readers and researchers who work in the optimization area and will be described in detail in the next chapters. Clustering techniques are in general used in the first phase of the optimization process to reduce the exploration space by grouping together solutions that share common features with the optimal solution. Engines that are composable [see manual and following chapters], can be applied to the objective function in an order requested by the user. For instance the clustering engine can be applied first and on the resulting configuration a hill-climber can be called for further optimization. Or a genetic algorithm combined with a hill-climber is applied first and further hill-climbing performed on the best solution. Other combinations and optimization sequences can be executed on user demand.

The Genetic Engine and general principles of the Genetic Algorithms (GAs)

In this section the general principles of genetic algorithms (GAs) will be revisited very briefly and problems specific to GWB will be discussed. For further and more detailed descriptions of GAs, their applications and current developments, reading of [Gold89], [Davis91], are strongly recommended.

Genetic algorithms were inspired by nature and they mimic some of the processes observed in natural evolution. The basic concept is very simple: the problem to be optimized is *encoded* as a sequence of bits (in the canonical form this is a binary encoding, but more convenient ones are also used) which constitutes the *chromosome.* In the initial phase a random population of chromosomes is generated and *evaluated* by measuring how worth every chromosome is in the problem context. This measure is called *fitness* of the chromosome and describes how well that particular structure encodes the solution of our problem. From this initial population new chromosomes are created by mating pairs of them according to rules that mimic the natural selection mechanism, where chromosomes that encode successful structures are allowed to reproduce more often than those who do not. The way the new children are created is by using two basic operators : ***crossover*** and ***mutation***. The crossover operators basically copies parts of the parents to the new structures and mutation randomly makes small changes to that information. In other words, the reproduction needs an *imperfect* copy of the information from the two parents. The new chromosomes are evaluated and inserted into the population in place of some other individuals according to some rules (for instance they replace the worst chromosomes in the old population). The simulated evolution continues until some rule says that time is up and the best individual is returned as the best solution of the problem. The process just described is the canonical GA using binary encoding and natural selection. From this basic algorithm an incredible number of variations have been conceived, experimented and

studied, to better fit the nature of the problem under optimization. The selection mechanism, the genetic operators, the encoding, the value of the control parameters, all of those and many more, can drastically change how effective the GA is on a specific problem and there are in practice very few rules that help the user during this process. This is the fundamental reason why the GWB was developed, as an aid to speed up the process just described and to help to experiment new ideas and make changes to any of the basic parts of the algorithm. Another motivation to build the GWB was that order-based problems, such as placement, need special representations and operators that are in general not supported in popular and commercially available GAs packages. Figure 2 illustrate the three fundamental phases of a classic GA that uses binary encoding and standard genetic operators.

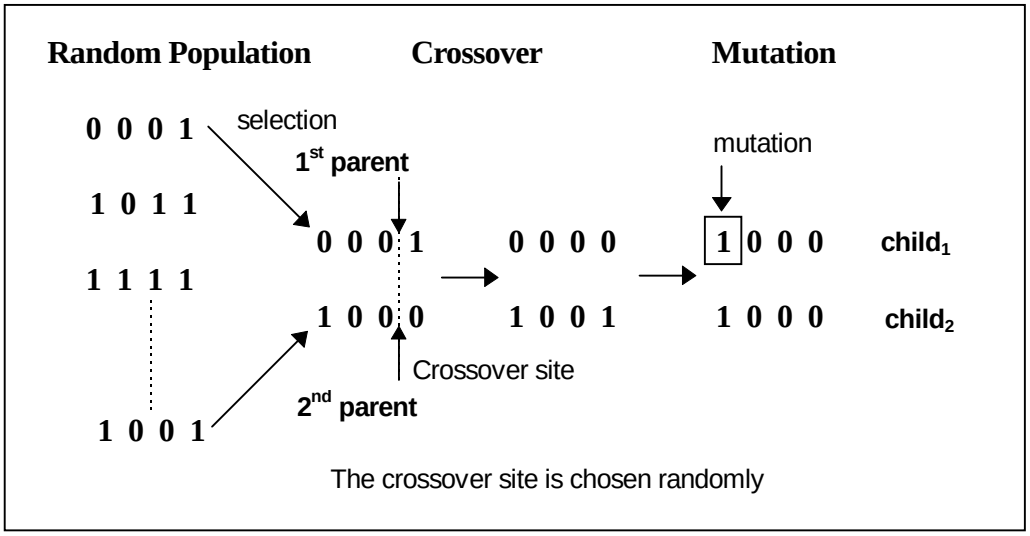


Fig. 2 : the classic genetic algorithm

In the simple binary encoding scheme parameters are represented by a sufficient number of bits and stringed together to form the chromosome. The order these parameters are present in the chromosome might matter for some problems and the genetic operators the GA is using. Because of that and in particular when chromosomes are very long, another genetic operator called ***inversion***, turns out to be useful in practice even if it requires additional overhead and needs some special structure to be supported. In the GWB inversion has been implemented by default and the probability that such operation takes place during reproduction is an additional user controlled parameter. Inversion needs an extended representation where each gene has an additional index associated with it which identifies its position in the string. This allows the genes to be reordered every time the fitness of the chromosome needs to be evaluated. In other words with this representation the fitness of the chromosome does not depend on the order of its genes. When inversion takes place, all the genes between two sites randomly selected along the chromosome are inversely reordered. (For example genes 1 2 3 4 5 before inversion, get changed into 5 4 3 2 1). Finally, after inversion, crossover and mutation are applied to the chromosome before its fitness is evaluated. For more information about inversion, readers are invited to look at [Gold89] pages 166 -170.

GAs in Placement and Order-Based Problems

One possible and natural way to encode order-based information into a linear chromosome is to simply string together the "name" of the objects in the desired order. For simplicity, names can be mapped to integers uniquely so that for instance the chromosome : **1 3 6 5 2 4** defines a placement on a line of the object $_1$ followed by object $_3$, then object $_6$ and so on. On a plane things are a little more complicated and better ways of representing a placement might be conceived, but if a continuous linear structure is still what one wants to use, the chromosome can be bent in order to "cover" all the objects allowed in a certain area. As an example see Fig. 3 on this page, where the darker connected line identifies the chromosome representing the sequence of cells with various size : 1, 2, 3, ..., 25. In this case, a special *placement routine* which knows about module sizes and maximum width, length and number of lines where these module are supposed to be placed, will "bend" the chromosome at the right sites to meet the requirements.

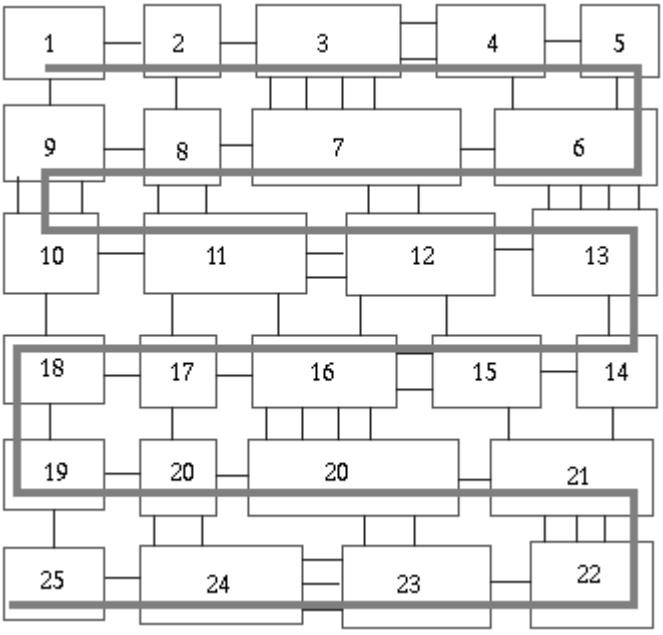
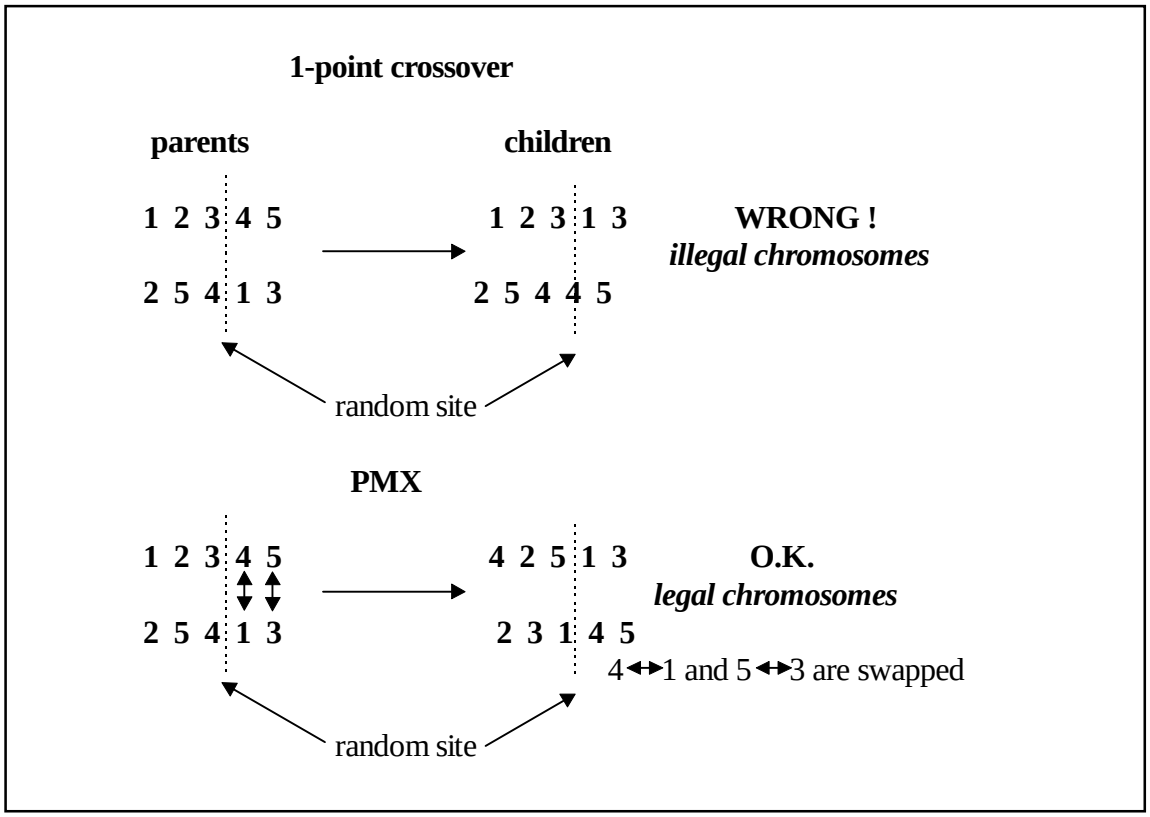


Fig. 3 : representation of placement of connected modules on a plane

One of the goals of an optimal placement is to minimize the total length of all connections among modules and this is what is going to be the objective function we want to optimize. To evaluate a chromosome then, after the placement routine has "unwound" the chromosome according to the user specifications such as row length, cells at fixed locations, etc. etc., special routines do the job given the coordinates **x, y** of each module and the graph which describes how they are connected to each other. It turns out that many interesting order-based problems can be represented by this simple structure : a linear chromosome carrying a sequence of

different objects. The relative order of these objects is what matters to the objective function, so every new configuration is described by a permutation of the original chromosome. Now let us see why whenever the order of different objects is important, the classic genetic operators, crossover, mutation and inversion, cannot be applied the way they were originally defined. It doesn't take too long to convince ourselves that if a chromosome must uniquely determine a permutation of some elements, when crossover takes place, some parts of the chromosome of the two parents cannot be freely copied, otherwise the new children would carry instances of the same objects. By the same token, mutation cannot change every object into any other also. The example 1 shows illegal chromosomes that are generated if a classic crossover operator (this is called 1-point crossover) is applied and how a specialized one (called partial matched crossover or PMX) works fine instead. In some order-based problems there are also other techniques which allow normal genetic operators to be used, but they are in general inefficient and in the case where permutations of elements are considered, as in placement, they wouldn't work at all. Generally these methods either allow illegal chromosomes to be generated, but they *penalize* them according to some criteria, or they try to reestablish the uniqueness of the permutation after the crossover has taken place and degrade the fitness of the new configuration by an amount proportional to the "damage" that had to be repaired. The GBW does not use any of them and new crossover mutation and inversion operators that always generate legal chromosomes have been implemented instead.

Example 1 :



In the example of the previous page, during crossover, the randomly chosen site divides every parent into two pieces : left side and right side. One child is generated by copying the leftmost of $parent_1$ and rightmost of $parent_2$ the other child by copying the leftmost of $parent_2$ and rightmost of $parent_1$. Clearly, illegal configurations are generated in this way. In the PMX, elements at the left side of the random site indicate the swaps to be performed in order to generate the children. In the case described in the example both swaps : **4** and **1, 5** and **3,** take place in the parents and generate the children**.** A mutation is simply implemented by swapping with some small probability, defined by the user, two elements of the same chromosome randomly chosen.

Genetic Options in the GWB

In the GWB a variety of reproduction strategies, genetic operators, mutations and control parameters has been implemented and can be selected by the users. Only new ideas or non-standard GAs techniques will be described here. Information about the more conventional operators, selection strategies and techniques in general, can be found in good GAs literature and in this paper wherever appropriate there will be pointers to the source of information. More interested users of the system can also refer to the user manual for a detailed description of some of the options and features of GWB.

Partial List of some of the GWB options :

Genetic operators and schemes followed by **[*]** will be explained in this section. For all the others see references or [Gold89] , [Davis91] and any other good literature on GAs.

Reproduction strategies :

1. Generational Reproduction	The entire population is replaced by a new one generated by using the classic reproduction scheme
2. Elitist	The new population always has the best chromosome generated so far.
3. Steady State	During reproduction new generated children replace the worst chromosome of the population.
4. CHC	Parents are selected according to their diversity (Hamming distance in their genotype). When this diversity cannot be maintained any more, a new population is regenerated from the

old one by changing only a small percentage of the genes in each chromosome.

5. Directional [StorPri95] — Parents and children define a "direction" in the multidimensional search space and new chromosomes are generated along these directions. It's a sort of discrete version of standard line optimization methods.

6. Simulating Annealing [MahGold92] — This scheme uses simulated annealing combined with crossover and mutation.

7. Deterministic Crowding [Mahfoud95] — This selection mechanism uses phenotypic crowding to keep diversity in small populations.

Selection schemes:

1. Scaling — It is used to avoid premature convergence at the beginning of the run and to enhance selection at the end of the run. This is accomplished by scaling the chromosomes' fitness proportionally to its difference from the average value.

2. Directional [KuoHwa93] — This is the classic scheme where chromosomes' fitness is compared to the population's average.

3. Disruptive " — The best and worst chromosomes are selected with the same probability.

4. Boltzmann [MahGold92] — A simulated annealing cooling schedule is applied to the selection procedure to improve diversity in the population and avoid premature convergence toward a local optima.

Genetic Operators (crossover) :

1. Partial Matched Crossover (PMX)
2. Cycle Crossover (CX)
3. Order Crossover (OX)
4. Uniform Order Crossover (UOX) [Davis90]

These are extensively covered in all books about GAs.

Genetic Operators (mutation) :

1. Simple	A simple swap of randomly selected genes.
2. Block	Random groups of chromosomes are swapped.
3. Permutation [Davis90]	A certain number of genes get randomly permuted.

In the case a FITT transformation is used [*] additional operators are available :

Additional operators for transformed chromosomes :

1. 1-point crossover	Extensively explained in classic GAs books.
2. 2-point crossover	"
3. Uniform crossover	"
4. 1-2-point crossover	1-point and 2-point crossover are both applied with a certain probability (user defined).
5. Reduce Surrogates [Genesis90]	This is used to make the probability of destroying one particular schema [see Holland71] during crossover independent of its length.

In addition to the above, genetic operators and reproduction schemes can be combined with hill-climbers, which can be applied to the best individuals of the final population, to the best child after each reproduction iteration or to the entire population at start.

Hill-climbers [*] Only a brief description is given. For more details refer to the GWB user manual.

1. Random Hill-climber
2. Random Inversion Hill-climber
3. Multiple operator Hill-climber
4. Dynamic Hill-climber
5. Net Hill-climber

All hill-climbers work under the assumption that an operator, such as swapping two elements in a chromosome, or a combination of those operators, is applied to the chromosome first and the new configuration evaluated. If the new value turns out to be better or the same of the previous one, then the new chromosome becomes the one on which to continue to iterate the procedure for a certain number of times.

Random Hill-climber	Elements in the chromosome at locations randomly selected are swapped.
Random Inversion Hill-climber	The same as before except that a genetic inversion is applied first.
Multiple Operator Hill-climber	Same as the random hill-climbers except that not only a swap, but also right-shift and left-shift operators are used. User parameters control the amount of shift and the probability that these operators are selected at every iteration.
Dynamic Hill-climber	This can only be applied if FITT is used. See [*] and [Yuret94]
Net Hill-climber	This is specific to the placement problem. Given a chromosome, one element (module) and one of its nets are randomly chosen. Then local moves of connected modules inside a given window are tried in order to minimize the net length.[see Fig. 4]

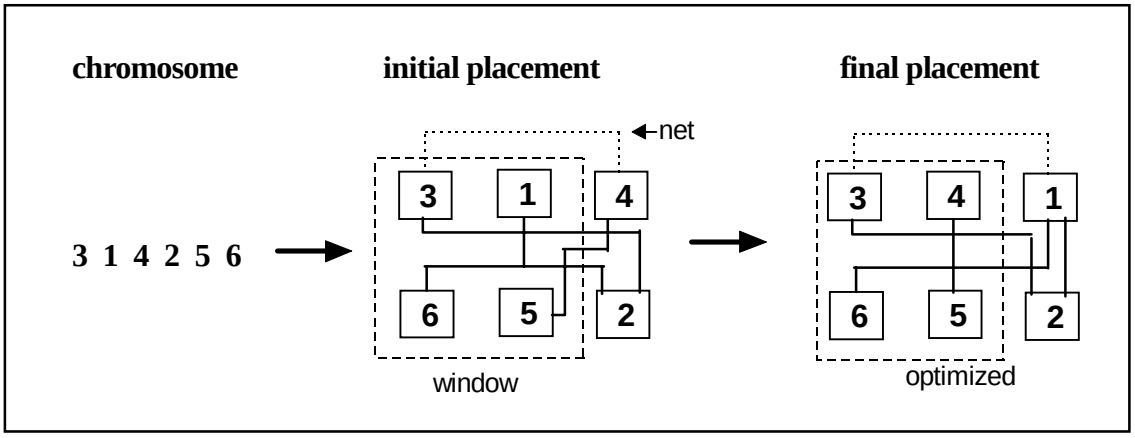


Fig. 4 : how a Net Hill-climber operates

The FITT (Fast Inversion Table Transform)

In this chapter a brief description of a new technique applied to GAs will be given. For more detailed information readers are referred to [Turr96]. This fast transform with its inverse are *O(n log n)* algorithms which map permutation domains into multidimensional cartesian (vector) domains and back. In GAs a chromosome which represents a permutation of *n* distinct elements, can be transformed into a vector with *n - 1* components and back. Using this idea order-based problems, after being transformed, can be solved by GAs that use standard genetic operators without introducing any more complexity. The inverse FITT is used every time the chromosome needs to be evaluated and the objective function does not have correspondent simple operations in the linear space. The Dynamic Hill-climber also uses this technique and has been modified to work on discrete spaces instead of continuous ones as originally proposed by its authors [MazYur94]

The Placement Problem and its Objective Function

One and perhaps the most difficult requirement for an optimal placement of connected modules on a board or components on a chip is to minimize the total length of all the connections. Other parameters such as density of wires, number of vias and prioritization of critical paths are also important and can be implemented as simple modifications of the previously mentioned objective function. For simplicity, all the examples and later the results reported here will have the total length of all connections as the only objective function to be optimized. Specific details such as how to handle separate unconnected circuits, non connected pins and many other tedious problems that unfortunately are present in real circuits and tend to break standard algorithms, have been fixed in the implementation to be able to run the suits of standard benchmarks. The details of the implementation will not be considered here. As good approximation of the objective function we like to minimize, the minimal spanning tree of the graph that describes the entire network of circuits to be placed can be computed every time a new placement is evaluated by executing a version of the popular Kruskal's algorithm [Krusk90]. Because this algorithm is somewhat expensive especially when dealing with large circuits, most of the time further approximation for multiple connected nets is used instead. In general, unless otherwise specified by the user, only one calculation of the minimal spanning tree is performed out of a hundred approximated evaluations. The multiple connected nets are approximated by using the popular method of computing half the perimeter of the net bounding box [see Fig. 5].

Our experiments support the strategy of only one exact evaluation of the total net length out of a hundred approximated ones, which turns out to be sufficient to avoid error propagation problems that could fool the optimization procedure.

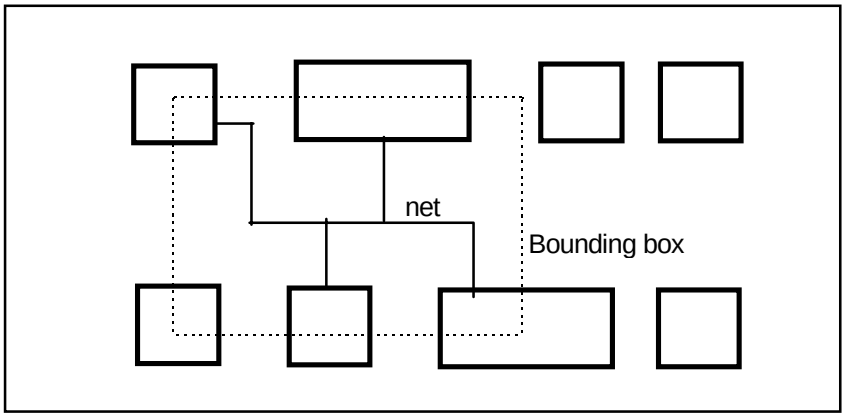


Fig. 5 : bounding box approximation

Every iteration of the method used to optimize a placement, has 4 main phases :

1. Change of representation (in a GA selection crossover and mutation are applied)
2. Cells are placed in rows.
3. The total net length is evaluated.
4. According to the value and some criteria, the new configurations are accepted or discarded.

During the phase 2 the coordinates of the cells are computed and stored in appropriate fields of the structure representing every single gene of the chromosome. In the GWB it has been decided to have in every gene three integer fields which, in the placement problem, are used to keep the cell name and the coordinates *X* and *Y* of the center of the cell. It turned out that with a specialized algorithm, inversion can also be implemented without using one more field in the gene structure. Also, in other order-based problems that have been investigated, this structure seemed to be sufficient to keep all the information that was needed. In the event that additional information stored in the genes is required, the code must be changed and the system recompiled. A trivial example of that could be another placement problem, that instead of being defined in a bidimensional space needs to be solved in the space where one more coordinate is needed. The routine responsible for the placement also takes care of other user defined constraints, such as placing certain cells at fixed locations, inserting feedthroughs and others that are typical in VLSI or in circuit board design.

The Simulated Annealing Engine

Simulated annealing is the simplest of the engines in the GWB, where the least amount of effort in experimenting and trying new solutions was applied. In fact, the real reason why such an engine has been included in the GWB system is to be able to prove some results when comparing the GBW techniques with TimberWolf. We also followed some of the ideas about the cooling schedule from the paper about parallel Boltzman selection schemes for GAs [see

MahGold92]. TimberWolf uses a very elaborated simulated annealing cooling schedule that they claim to be very efficient when applied to this problem. In addition to all sorts of tricks, quadratic estimators, windowing operations and more, its cooling schedule is embedded in a hierarchical clustering algorithm which run by default on top of simulated annealing engine. As it will be shown later TimberWolf delivers very good results on the benchmarks we tried, especially in circuits of large complexity, but our intuition was that they depended in largely on the clustering and the incremental evaluation of the placement, rather than the special simulated annealing schedule. To support this idea a simple cooling schedule was implemented and a faster selection mechanism was used when changing the configuration in order to decrease the time spent during the evaluation. The cooling schedule is based on the original Metropolis algorithm and is controlled by three main temperatures : at the initial state of the placement, at an intermediate point and at its final state.[see Fig. 6]. After a swap of two cells chosen according some rules explained later, the new configuration is accepted with probability expressed as a function of the difference in cost from the previous configuration and the one at the current temperature. Temperatures are updated by multiplying them by constants computed after some statistical analysis and considerations similar to those described in the paper about Boltzmann selection [MahGold92]. The equations that relate ***p***, the probability of accepting a new configuration, the temperature and the cost of the placement are :

$$p(\Delta C) = 1, \quad if\ (\Delta C \geq 0)$$

$$p(\Delta C) = e^{\frac{\Delta C}{T}}, \quad if\ (\Delta C < 0)$$

Metropolis cooling schedule

where $\Delta C = C_{new} - C_{old}$, T = *current temperature*

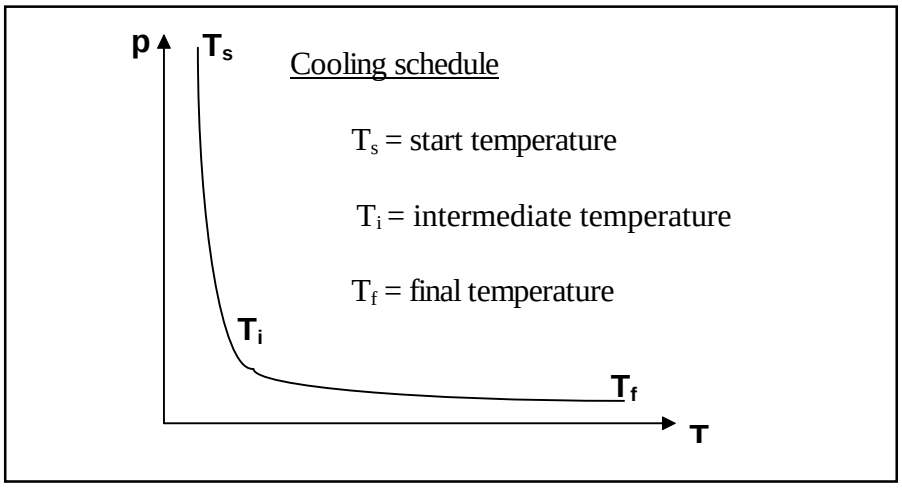


Fig 6 : GWB cooling schedule

Let us take now a closer look at what happen during the evaluation of a placement of modules of various sizes arranged by row on a plane [see Fig. 7]. A general observation about any simulated annealing engine is that no matter which operation is used to update a configuration, to be statistically meaningful, the number of changes and evaluations at each temperature must

be very large, especially in VLSI placements where circuits contain a large number of cells or modules. It goes without saying that if results are expected in a reasonable amount of time, the modification itself and the evaluation of the new configuration must be as fast as possible. Clearly the simplest operation that simulated annealing can use in order to preserve the permutation is to swap randomly two cells and compute the cost of the new placement. TimberWolf uses a fancier replacement policy and even one cell moves and rotations are tried, but we will not consider that here. The problem with swapping cells of *different* sizes is illustrated in Fig. 7. Because in a placement by row one of the user parameters is the maximum length of each row where the cells are packed together, if the two elements to be swapped belong to different rows, a large number of nets between the two gets shifted and must be recalculated. It turns out that in almost all circuits there are many instances of the same type of cells, for instance 2-input nand gates, 4-way multiplexers, flip-flops, etc.. which have the same size. Therefore if we grouped together cells with the same size, when for instance, we needed to change a configuration with a simple swap operation, we could pick the two from the same group. This means that only nets that belong to the two cells need to be updated, no shifting will occur between rows, cutting down the amount of computation necessary in each evaluation. Clearly these restricted swaps will not allow a random and uniform sampling of the search space for all possible legal configurations, so once in a while two cells which belong to different sets, therefore with different sizes, will be tried also.

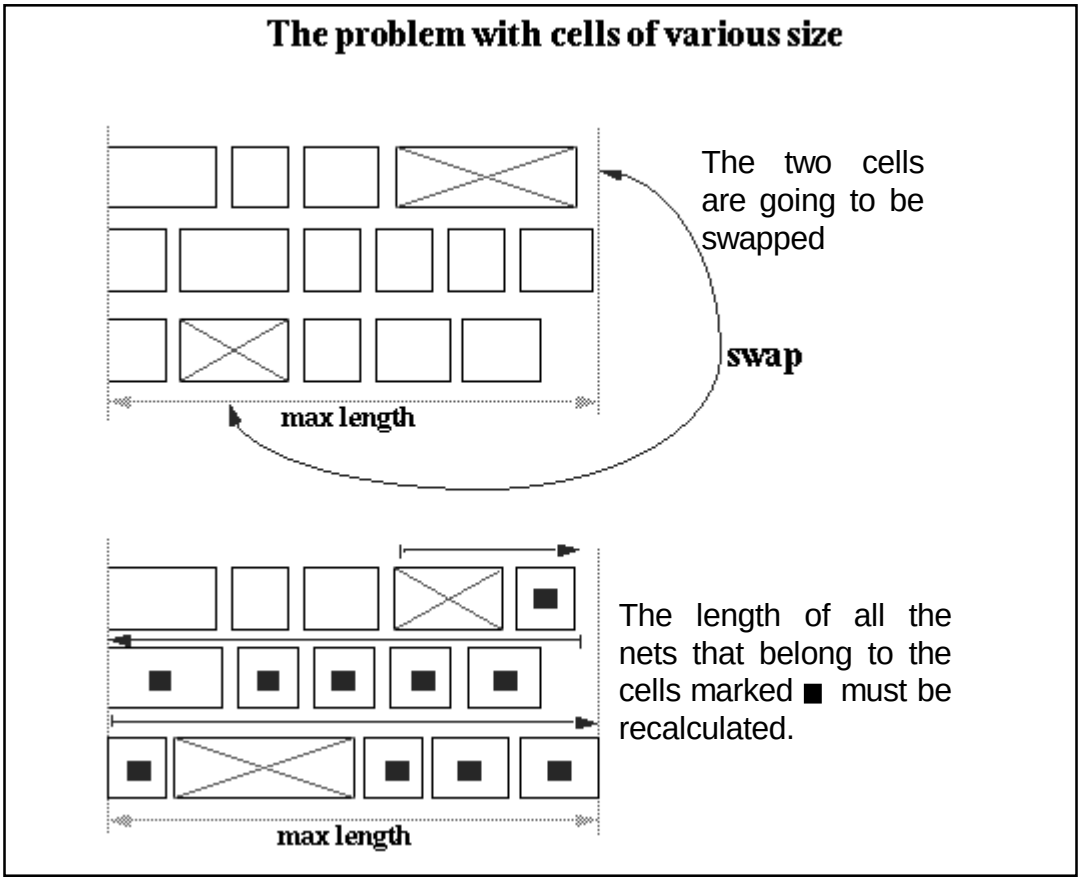


Fig. 8 : swapping cells of different sizes in a placement by row

As we anticipated, the results of several experiments on various circuits support the policy that at least one out of a hundred iterations requires swapping cells of different sizes, but in the GWB this can also be changed on demand. No windowing mechanism or special estimators have been added to improve the selection phase before the iteration of the new placements and the results reported later will reflect the simple implementation just described.

Clustering

In the placement problem, clustering is a technique which aggregates different cells according to some specific objectives. One objective, for instance, can be to group together cells that share many connections, because they need to be placed the best, in order to minimize the total net-length. Another way to see clustering is that once these groups of cells that share some properties are formed, other optimization engines can be applied to them as if they were single macro cells. Anybody can see here that clustering, if effectively done, can dramatically improve any optimization procedure that can be now applied to a smaller number of groups of cells. Once the groups are optimally placed, then the same engine can optimize placements inside each single cluster of cells, assuming that the two phases to a certain degree are independent of each other. Clearly there are dependencies and perfect clustering cannot be expected by any technique invented so far, but to anticipate one of the conclusions of this report, this seems to be the only known method that allows the large placements needed in VLSI to be performed in a reasonable amount of time and with good quality. As previously said TimberWolf combines clustering with simulated annealing and in the version we tested they could not be separated or disabled independently. Clustering is the part of the GWB least developed and where more work needs to be done to fully exploit the potential that these techniques seem to have. Before describing the new approach that we used in the GWB a general observation that motivated it should be articulated and explained. Almost every placement we know of have some optional constraints that must be satisfied such as cells placed at fixed locations or oriented in some way. The reason is that circuits have inputs and outputs so somehow they have to communicate with the external world through preferred buses and wires located at fixed coordinates. Even if an entire chip is asked to be placed by one of this tool and not only parts of it, signals must be connected to the external pads, which implicitly defines preferred areas where the input and output cells must be located. So every placement starts with a set of constraints and might be thought as a multi-phase process where connected elements are optimally placed relative to the constraints defined in the previous phase. These partial placements are clearly done according to a greedy algorithm and we cannot certainly say there is a global view of the entire process. Nonetheless even without additional optimizations, the selection strategy used to place connected cells according to the constraints of the previous configurations by itself produced results that are sometimes better than either GAs or other methods. Moreover, because a complete evaluation of the total net length is not needed during the phases just described, this algorithm is extremely fast: at least 3 orders of magnitude faster than other techniques we experimented.

The PQ Clustering

The name of PQ clustering we use comes from the *Priority Queue*, which is the structure used to operate the selection of the elements that are placed one by one during the process. In order for the priority queue to know which element to pay attention to and knowing that we want to cluster together cells that will only have minimum length nets, we introduce this general idea also used by other tools : during the partial placements, cells which share the larger number of connections will be grouped together. By using this approach we form clusters around cells at the fixed locations, optimized for the maximum number of intra-connections and minimum number of inter-connections. In reality things are a little more complex, because of the multi-pin nets real circuits have to deal with. In order to solve this problem we borrowed an idea used in TimberWolf as well as other clustering methods. The definition for ***strength*** of a connection is needed : the strength ***w*** of a net connected to ***n*** pins is the reciprocal of ***n - 1***.
Or in mathematical notation :

$$w = \frac{1}{n-1}$$ ***strength of a net connected to n pins***

[see Fig. 9] where some examples of connections and their strengths are shown.

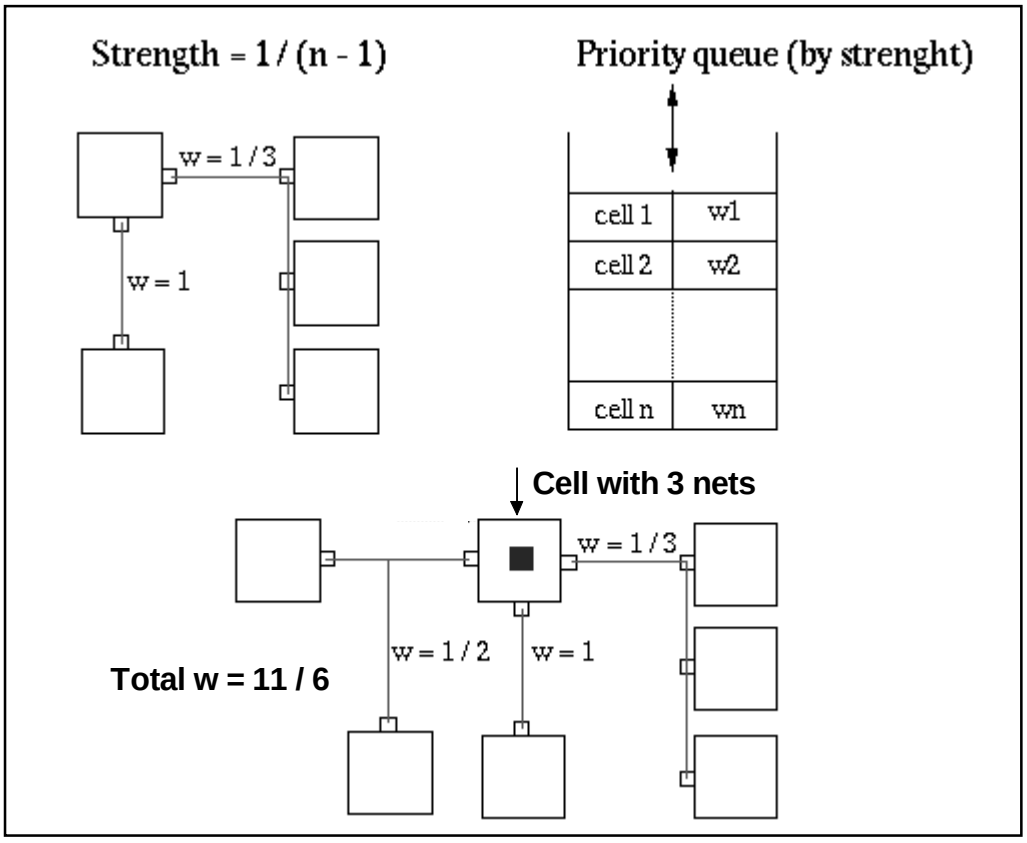


Fig. 9 : strength of various connections

Notice that by this definition every point to point connection has the maximum strength of one. We can then think of our circuit as described by nets with weights associated to them, which

are the strengths we just defined. Cells to be placed are kept in a priority queue and new cells are selected from this pool according to the strength of their connections. In general, every cell can have multiple nets connected to it, therefore the value stored in the priority queue will be the sum of the weights of all its nets. The simplest implementation the PQ clustering algorithm can be summarized as follows :

Phase1 - All cells at fixed locations are pushed into the PQ.

Phase 2 - The cell with the highest strength is extracted from PQ , placed in a row and the cells connected to it examined.
The new connected cells, that are not in PQ already, are pushed into the stack and their strength computed, if they are already in the stack the strength of the new connection is added to the previous value.

Phase 3 - Phase 1 and 2 are iterated, placing cells and forming new rows when the maximum length of the current row is reached, until PQ is emptied.

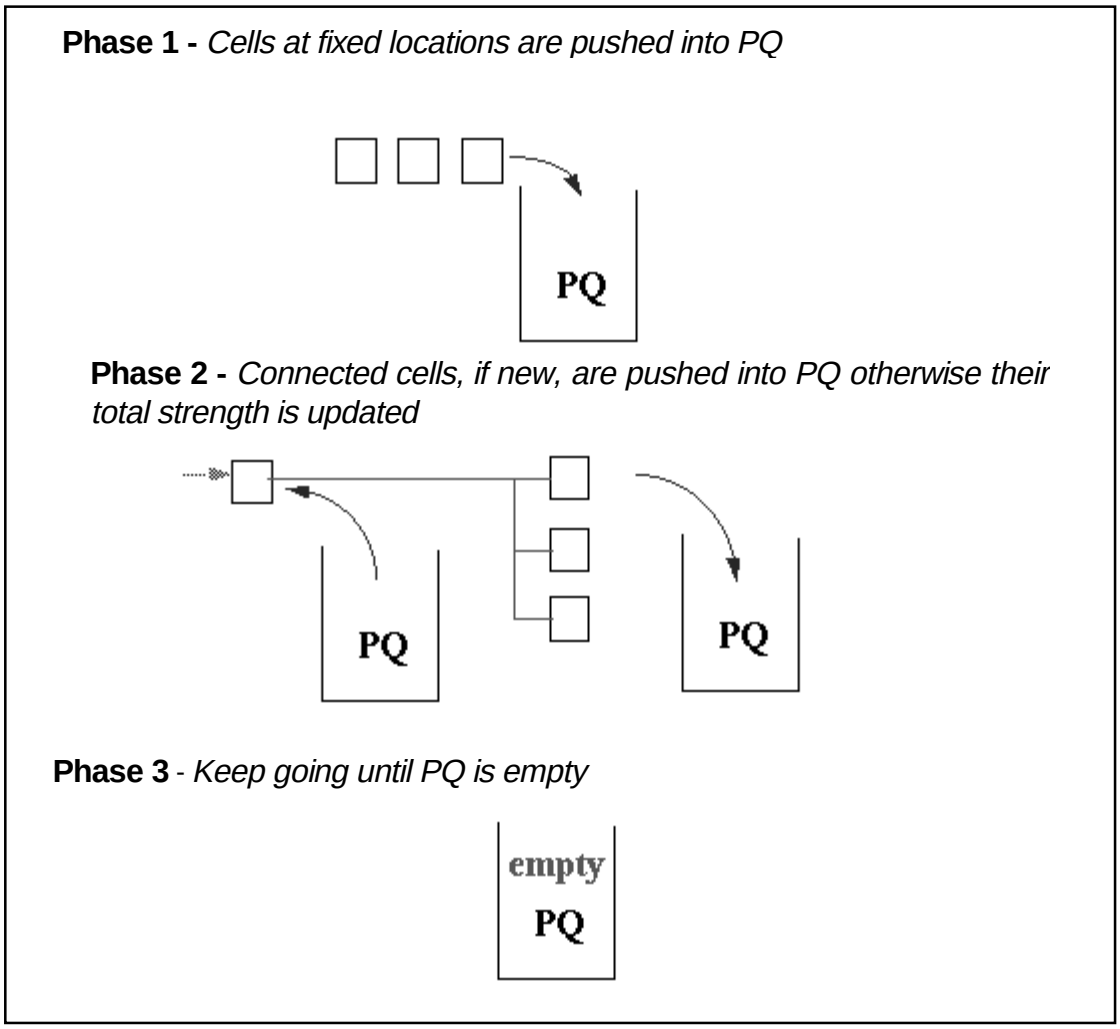


Fig. 10 : PQ clustering by row

An illustration of the simple procedure is also shown in Fig. 10. After this procedure we are left with a certain number of rows (clusters) that can be placed by an optimization engine such as GAs, a hill-climber, or anything the user decides to use. If not specified otherwise, the GWB iterates by default a hill-climber for ten times the average number of cells in a row.

As it will be apparent from the results reported in the next chapter, PQ clustering is very fast with large VLSI placements if compared to any other methods used in the GWB and TimberWolf as well. The quality of the final placement is also very good if compared to other methods in the GWB, but worse than TimberWolf, especially when placing very large circuits.
One of the reasons is that the cells extracted from the PQ are placed close to each other in the same row until its maximum length is reached; clearly clusters optimized in only one dimension are built in this way. A two-dimensional approach should be used instead, but at the time this code was written we did not have any good ideas of how to implement an algorithm that could take the second dimension into account. Some new ideas came up recently, but they have not been implemented yet so for now we have no results to show how much better quality and faster placements can be generated on large applications by exploiting these new methods.
Our intuition is that they should improve greatly the quality without slowing down the execution time substantially.

Placement results

TimberWolf ver.7 was available to us only in executable form for a DecStation 5000, therefore the GWB was also tested on the same platform with 300 Mbytes of memory. Also, execution times for the GWB are between 3 - 4 times faster for the version running on the AlphaStation 400, 4/233 than the ones reported here and used for the comparison with TimberWolf. In the first set of tables TimberWolf results are the *best* obtained out of ten runs on the same circuit meanwhile the results for the GWB are the *worst* also out of ten runs with a specific optimization method. On average, the difference between the best and worst runs is between 5 - 10 % on all the circuits we tried, on both the GWB and TimberWolf. Moreover, for the GWB only the best combinations of genetic algorithm and hill-climber have been reported and they don't include any clustering up front, meanwhile TimberWolf always does clustering when the circuit to be optimized has more than a few hundreds cells. Results from the GWB simulating annealing engine have also been reported to compare our different approach with the one used by TimberWolf. The netlists of the circuits that have been tested are either from the standard benchmark used by the CAD community or generated at WRL [see Fig. 11] from real circuits and complexities that go from a few cells to a few thousands. In separate tables the results of PQ clustering has been also provided. As already explained the execution times suggest that further research on better bidimensional cluster placement should be undertaken in the future if a truly VLSI placement with hundreds of thousands of cells is sought.

circuit	# cells	# nets	# pins
fish100	100	180	360
fish1000	1000	1930	3860
p1	752	831	2821
p2	2907	2961	11038
biomed	6417	5711	20943
avqsmall	21854	22116	76167
avqslarge	25114	25376	82687
industrial_2	12142	12949	47909
industrial_3	15059	21808	68044

Fig. 10 : some of the standard benchmarks

Genetic algorithms : the winner

The best results consistently generated over all the possible combinations of parameter values, selection strategies, genetic operators and hill-climbing techniques have been obtained by running a combination of a GA and a net-hill-climber previously described with the following parameters :

selection scheme : steady-state elitist with replacement of the worst two individuals of the population

genetic cross-over operator : Uniform Order-based cross-over (UOX) with probability $p_c = 0.8$ [see Davis90, pages 80 - 81]

genetic mutation operator : simple swap with probability $p_{swap} = 0.8$. This correspond to the standard mutation rate per gene proportional to the inverse of the number of cells (chromosome length) or $p_m \cong \frac{0.8}{\#cells}$ [Ex. 100 cells : $p_m \cong 0.008$]

hill-climber : net-hill climber with a window 3 columns high and 4 times as large as the minimum cell width. Net hill-climbing was applied to the best individual after cross-over, mutation and inversion.

Inversion : with probability $p_i = 0.3$

population size : 100 individuals if not otherwise specified

The other strategy that performed almost as good as the one reported was a GA that used Deterministic Crowding [see Mahfould95, pag. 139 - 152], as the selection mechanism during the run. Boltzmann selection also did very well in terms of quality of placement by been able to keep the chromosome diversity in the population until the end of the run, but it required much larger execution times. Our opinion is that this selection mechanism can be effectively used for the placement problem and this representation only if the GA is running on a multiprocessor system [MahGold92]. The population size has been limited to 100 individuals for the circuits in Table 1 and 200 individuals for the circuits in Table 2, if not otherwise specified.

Hill-Climbers : the winner

All the hill-climbers in the GWB accept a new configuration only if its fitness is superior or equal to the previous one after certain operations have been performed. The general problem with hill-climbers is that they only explore locally and they do not keep any global information that can be used during the search for the optimum. They can only generate new configurations that are better, or with the same cost, than the previous ones. Simulated annealing can be seen in this context as a particular hill-climber that sometimes, based on statistical measures, decides to accept a worst configuration with a probability that at the beginning is very high and at the end of the run becomes almost negligible. Another way of looking at simulating annealing is as an almost random search at the beginning and a straight hill-climber in the end. For this reason hill-climbers in the GWB can be used in combination with regular GAs or other techniques that allow global optimization, to improve and accelerate convergence toward local optima, once interesting regions of the search space have been identified. The two techniques that turned out to be the most successful during the experiments with the GWB where :

1. *dynamic operator change* : multiple operations that applied to the old configuration generate the new one are allowed. They are randomly or deterministically changed during the run.

2. *transformation from permutation into vectors and back* : this allows techniques successfully used in linear (vector) spaces to be applied to order-based problems, such as placement in our case. A distance between points in the search space can be defined in the transformed space allowing a more effective search.

In one of the hill-climbers that belong to the first category we just described, three operators are allowed :

1. *swapping* randomly chosen cells (same and different sizes)
2. *shifting-right* by a certain amount a randomly chosen cell
3. same as the previous one, but *shifting-left* instead.

Each operator is used until a certain number of iterations has been reached or when no improvement has been reported for too many iterations. The other technique is of more general use and allows algorithms such as Dynamic Hill-Climbing (DHC) to be applied to order-based problems. In the GWB the DHC algorithm [see MazYur94 and Yuret94] used for optimizing continuos functions has been adapted to work also in the discrete spaces generated by transforming permutations into multidimensional vectors. Readers interested in the details of the algorithm in addition to the De La Maza and Yuret's suggested paper should also refer to the author's technical report [Turr96] available on the Web too. From the tables, results show that DHC performs quite well even if compared with the combined clustering and simulated annealing algorithm in TimberWolf. Because DHC showed constantly better results than any other hill-climber we tried, in the tables only the DHC results have been reported.

Simulated Annealing : results

The results indicate that it is actually clustering that is providing very good results and not so much the elaborated annealing schedule that TimberWolf uses. In fact the simple simulated annealing schedule implemented in the GWB with the addition of the group selection scheme used to swap, most of the time, cells with the same size, performs even better than TimberWolf on small circuits. In larger circuits the effect of clustering is clearly overwhelming.

Clustering : results

In the last two tables the results of comparing PQ clustering with TimberWolf are shown. The only purpose is to show the execution times of the PQ-clustering are more than 3 order of magnitude better than TimerWolf special clustering and annealing. The quality of placement is not bad either for circuit of medium size, but is clearly not as good as TimberWolf for larger sizes. As mentioned before, a better strategy for optimizing the clusters in the two dimensions instead of by row only, in our opinion should produce much better placements and should be at least investigated in the future.

Legend : *genetic* : the best GWB Genetic Algorithm; *DHC* : Dynamic Hill-Climber

SA : GWB Simulated Annealing; *TW* : TimberWolf ver. 7

all values are in the form of : ***cost / time*** [read smaller, better],
the solution with the lowest cost will be shown with shading

algorithm	**28 cells (c/t)**	**100 cells (c/t)**	**200 cells (c/t)**
genetic	1624 / 400.0 $_{pop1}$	552 / 1020.0 $_{pop2}$	1400 / 4000 $_{pop3}$
DHC	1700 / 10.2	580 / 100.0	1380 / 380.0
SA	1700 / 11.0	552 / 40.0	1100 / 170.0
TW	1814 / 54.0	680 / 65.0	1480 / 200.0

pop1 : 100 individuals; pop2 : 600 individuals; pop3 : 1200 individuals

Table 1 : placement (all without clustering)

Algorithm	**752 cells (c/t)**	**1,000 cells(c/t)**	**2,907 cells(c/t)**
genetic	3050000 / 5100 $_{pop4}$	too large population	too large population
DHC	1120000 / 2200	140350 / 9850	too much time
SA	789000 / 620	785450 / 1560	6003450 / 5100
TW	821000 / 899.7	839850 / 1883	4009870 / 3672

pop4 : 2000 individuals; too large population means impractical

Table 2 : placement (only TW with clustering)

algorithm	**752 cells (c/t)**	**1,000 cells (c/t)**	**2,907 cells (c/t)**	**6417 cells (c/t)**
PQ clust. only	109000 / 3.1	1010200 / 1.12	5400000 / 4.2	6160072 / 10.1
TW SA + clust.	821000 / 899.7	839850 / 1883	341000 / 3672	3222672 / 9120

But, look at the difference in execution time !

Table 3 : just clustering (TW + SA)

How close the optimal solution is ?

In order to have at least the feeling of how close the solution generated by one of the best engine is to the real absolute optimum, in the last table [Tab. 4] a medium size circuit with 1000 gates and a highly symmetrical structure (a fishnet structure) has been automatically generated. The optimal solution ,with all the symmetrical ones, was known by construction as well as its cost. The experiment consisted on running TimberWolf allowing the simulated engine to operate for an increasing number of seconds and see how well the results were approaching the known optimal placement. The cost of the optimal placement was of 520,000 units and from the table one can see that even after more than 19 hours and a half the job was running on a DecStation 5000, the solution produced was still almost 35 % worse than the real global optimum. Needless to say that larger circuits performed even worse than that. If not for anything else this simple result is showing that optimized placements produced by the best tools available today are far from being acceptable even for circuits of modest size and more research and better methods should be sought.

all values are in the form of : ***cost (time)*** [read smaller, better]

1,000 cells	**cost (t = 1883s)**	**cost (t = 7252s)**	**cost (t = 71203s)**
TimberWolf ver. 7	839,850	812,240	698,600

Table 4 : TW cost and running time

Conclusions

A few general conclusions can be drawn from the results of this research :

- Clustering, maybe combined with some other optimization technique, seems to be a good way for dealing with large placements required in VLSI and there is space for improvements.
- For the simple genetic representation described, genetic algorithms are the slowest and require large populations if they run on single processor architectures. They do deliver good quality placements for circuit of small complexity and can be used in those cases.
- New techniques that work on vectors, instead of permutations, should be investigated considering that the new method based on analytic transformations (FITT) showed very promising results with a simple hill-climbing algorithm and could be used for this purpose.
- Smarter schemes can do better than sophisticated annealing cooling schedules.

References

Davis90 "Handbook of Genetic Algorithms" by Lawrence Davis, Van Nostrand Reinhold 1991.

Genitor93 Genetic free package from University of Colorado
FTP : ftp.cs.colostate.edu:/pub/GENITOR.tar
see Chapter 4.1 for reference.

Gold89 "Genetic Algorithms in Search Optimization & Machine Learning", David E. Goldberg, Addison-Wesley, 1989.

Holland71 "Processing and processors for schemata" in E. L. Jacks (Ed.), Associative information processing (pp. 127-146). New York: American Elsevier.

Krusk90 "Introduction to Algorithms", by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, editor McGrow Hill, MIT Press, 1990, pages 504 - 505.

KuoHwa93 "A Genetic Algorithm with Disruptive Selection" from the 5th Proceedings on Genetic Algorithms by Stephanie Forrest 1993, pag. 65 - 69.

MahGold92 "Parallel Recombinative Simulated Annealing : A Genetic Algorithm IlleGAl report n. 93006 July 1993 by Samir W. Mahfoud & David E. Goldberg.

Mahfoud95 "Niching Methods for Genetic Algorithms" by Samir W. Mahfoud at University of Illinois Urbana-Champaign, 1995.

MazYur94 "Dynamic Hill Climbing", article on AI Expert, March 1994, pag. 26-31.

StorPri95 "Differential Evolution - A simple and efiicient adaptive scheme for global optimization over continuous spaces", by Rainer Storn and Kenneth Price, 1995, International Computer Institute, Berkeley.

Turr96 "Optimization in Permutation Spaces", by Silvio Turrini, WRL Technical Report 96/1, Digital Equipment Corporation, Western Research Laboratory.
[http://www.research.digital.com/wrl/techreports/abstracts/96.1.html]

Yuret94 "From Genetic Algorithms To Efficient Optimization" by Deniz Yuret at Massachusetts Institute Of Technology Master in Science in Electrical Engineering and Computer Science 1994.

WRL Research Reports

‘‘Titan System Manual.’’ **Michael J. K. Nielsen.** WRL Research Report 86/1, September 1986.

‘‘Global Register Allocation at Link Time.’’ **David W. Wall.** WRL Research Report 86/3, October 1986.

‘‘Optimal Finned Heat Sinks.’’ **William R. Hamburgen.** WRL Research Report 86/4, October 1986.

‘‘The Mahler Experience: Using an Intermediate Language as the Machine Description.’’ **David W. Wall and Michael L. Powell.** WRL Research Report 87/1, August 1987.

‘‘The Packet Filter: An Efficient Mechanism for User-level Network Code.’’ **Jeffrey C. Mogul, Richard F. Rashid, Michael J. Accetta.** WRL Research Report 87/2, November 1987.

‘‘Fragmentation Considered Harmful.’’ **Christopher A. Kent, Jeffrey C. Mogul.** WRL Research Report 87/3, December 1987.

‘‘Cache Coherence in Distributed Systems.’’ **Christopher A. Kent.** WRL Research Report 87/4, December 1987.

‘‘Register Windows vs. Register Allocation.’’ **David W. Wall.** WRL Research Report 87/5, December 1987.

‘‘Editing Graphical Objects Using Procedural Representations.’’ **Paul J. Asente.** WRL Research Report 87/6, November 1987.

‘‘The USENET Cookbook: an Experiment in Electronic Publication.’’ **Brian K. Reid.** WRL Research Report 87/7, December 1987.

‘‘MultiTitan: Four Architecture Papers.’’ **Norman P. Jouppi, Jeremy Dion, David Boggs, Michael J. K. Nielsen.** WRL Research Report 87/8, April 1988.

‘‘Fast Printed Circuit Board Routing.’’ **Jeremy Dion.** WRL Research Report 88/1, March 1988.

‘‘Compacting Garbage Collection with Ambiguous Roots.’’ **Joel F. Bartlett.** WRL Research Report 88/2, February 1988.

‘‘The Experimental Literature of The Internet: An Annotated Bibliography.’’ **Jeffrey C. Mogul.** WRL Research Report 88/3, August 1988.

‘‘Measured Capacity of an Ethernet: Myths and Reality.’’ **David R. Boggs, Jeffrey C. Mogul, Christopher A. Kent.** WRL Research Report 88/4, September 1988.

‘‘Visa Protocols for Controlling Inter-Organizational Datagram Flow: Extended Description.’’ **Deborah Estrin, Jeffrey C. Mogul, Gene Tsudik, Kamaljit Anand.** WRL Research Report 88/5, December 1988.

‘‘SCHEME->C A Portable Scheme-to-C Compiler.’’ **Joel F. Bartlett.** WRL Research Report 89/1, January 1989.

‘‘Optimal Group Distribution in Carry-Skip Adders.’’ **Silvio Turrini.** WRL Research Report 89/2, February 1989.

‘‘Precise Robotic Paste Dot Dispensing.’’ **William R. Hamburgen.** WRL Research Report 89/3, February 1989.

‘‘Simple and Flexible Datagram Access Controls for Unix-based Gateways.’’ **Jeffrey C. Mogul.** WRL Research Report 89/4, March 1989.

‘‘Spritely NFS: Implementation and Performance of Cache-Consistency Protocols.’’ **V. Srinivasan and Jeffrey C. Mogul.** WRL Research Report 89/5, May 1989.

‘‘Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines.’’ **Norman P. Jouppi and David W. Wall.** WRL Research Report 89/7, July 1989.

‘‘A Unified Vector/Scalar Floating-Point Architecture.’’ **Norman P. Jouppi, Jonathan Bertoni, and David W. Wall.** WRL Research Report 89/8, July 1989.

''Architectural and Organizational Tradeoffs in the Design of the MultiTitan CPU.'' **Norman P. Jouppi.** WRL Research Report 89/9, July 1989.

''Integration and Packaging Plateaus of Processor Performance.'' **Norman P. Jouppi.** WRL Research Report 89/10, July 1989.

''A 20-MIPS Sustained 32-bit CMOS Microprocessor with High Ratio of Sustained to Peak Performance.'' **Norman P. Jouppi and Jeffrey Y. F. Tang.** WRL Research Report 89/11, July 1989.

''The Distribution of Instruction-Level and Machine Parallelism and Its Effect on Performance.'' **Norman P. Jouppi.** WRL Research Report 89/13, July 1989.

''Long Address Traces from RISC Machines: Generation and Analysis.'' **Anita Borg, R.E.Kessler, Georgia Lazana, and David W. Wall.** WRL Research Report 89/14, September 1989.

''Link-Time Code Modification.'' **David W. Wall.** WRL Research Report 89/17, September 1989.

''Noise Issues in the ECL Circuit Family.'' **Jeffrey Y.F. Tang and J. Leon Yang.** WRL Research Report 90/1, January 1990.

''Efficient Generation of Test Patterns Using Boolean Satisfiablilty.'' **Tracy Larrabee.** WRL Research Report 90/2, February 1990.

''Two Papers on Test Pattern Generation.'' **Tracy Larrabee.** WRL Research Report 90/3, March 1990.

''Virtual Memory vs. The File System.'' **Michael N. Nelson.** WRL Research Report 90/4, March 1990.

''Efficient Use of Workstations for Passive Monitoring of Local Area Networks.'' **Jeffrey C. Mogul.** WRL Research Report 90/5, July 1990.

''A One-Dimensional Thermal Model for the VAX 9000 Multi Chip Units.'' **John S. Fitch.** WRL Research Report 90/6, July 1990.

''1990 DECWRL/Livermore Magic Release.'' **Robert N. Mayo, Michael H. Arnold, Walter S. Scott, Don Stark, Gordon T. Hamachi.** WRL Research Report 90/7, September 1990.

''Pool Boiling Enhancement Techniques for Water at Low Pressure.'' **Wade R. McGillis, John S. Fitch, William R. Hamburgen, Van P. Carey.** WRL Research Report 90/9, December 1990.

''Writing Fast X Servers for Dumb Color Frame Buffers.'' **Joel McCormack.** WRL Research Report 91/1, February 1991.

''A Simulation Based Study of TLB Performance.'' **J. Bradley Chen, Anita Borg, Norman P. Jouppi.** WRL Research Report 91/2, November 1991.

''Analysis of Power Supply Networks in VLSI Circuits.'' **Don Stark.** WRL Research Report 91/3, April 1991.

''TurboChannel T1 Adapter.'' **David Boggs.** WRL Research Report 91/4, April 1991.

''Procedure Merging with Instruction Caches.'' **Scott McFarling.** WRL Research Report 91/5, March 1991.

''Don't Fidget with Widgets, Draw!.'' **Joel Bartlett.** WRL Research Report 91/6, May 1991.

''Pool Boiling on Small Heat Dissipating Elements in Water at Subatmospheric Pressure.'' **Wade R. McGillis, John S. Fitch, William R. Hamburgen, Van P. Carey.** WRL Research Report 91/7, June 1991.

''Incremental, Generational Mostly-Copying Garbage Collection in Uncooperative Environments.'' **G. May Yip.** WRL Research Report 91/8, June 1991.

''Interleaved Fin Thermal Connectors for Multichip Modules.'' **William R. Hamburgen.** WRL Research Report 91/9, August 1991.

''Experience with a Software-defined Machine Architecture.'' **David W. Wall.** WRL Research Report 91/10, August 1991.

‘‘Network Locality at the Scale of Processes.’’ **Jeffrey C. Mogul.** WRL Research Report 91/11, November 1991.

‘‘Cache Write Policies and Performance.’’ **Norman P. Jouppi.** WRL Research Report 91/12, December 1991.

‘‘Packaging a 150 W Bipolar ECL Microprocessor.’’ **William R. Hamburgen, John S. Fitch.** WRL Research Report 92/1, March 1992.

‘‘Observing TCP Dynamics in Real Networks.’’ **Jeffrey C. Mogul.** WRL Research Report 92/2, April 1992.

‘‘Systems for Late Code Modification.’’ **David W. Wall.** WRL Research Report 92/3, May 1992.

‘‘Piecewise Linear Models for Switch-Level Simulation.’’ **Russell Kao.** WRL Research Report 92/5, September 1992.

‘‘A Practical System for Intermodule Code Optimization at Link-Time.’’ **Amitabh Srivastava and David W. Wall.** WRL Research Report 92/6, December 1992.

‘‘A Smart Frame Buffer.’’ **Joel McCormack & Bob McNamara.** WRL Research Report 93/1, January 1993.

‘‘Recovery in Spritely NFS.’’ **Jeffrey C. Mogul.** WRL Research Report 93/2, June 1993.

‘‘Tradeoffs in Two-Level On-Chip Caching.’’ **Norman P. Jouppi & Steven J.E. Wilton.** WRL Research Report 93/3, October 1993.

‘‘Unreachable Procedures in Object-oriented Programing.’’ **Amitabh Srivastava.** WRL Research Report 93/4, August 1993.

‘‘An Enhanced Access and Cycle Time Model for On-Chip Caches.’’ **Steven J.E. Wilton and Norman P. Jouppi.** WRL Research Report 93/5, July 1994.

‘‘Limits of Instruction-Level Parallelism.’’ **David W. Wall.** WRL Research Report 93/6, November 1993.

‘‘Fluoroelastomer Pressure Pad Design for Microelectronic Applications.’’ **Alberto Makino, William R. Hamburgen, John S. Fitch.** WRL Research Report 93/7, November 1993.

‘‘A 300MHz 115W 32b Bipolar ECL Microprocessor.’’ **Norman P. Jouppi, Patrick Boyle, Jeremy Dion, Mary Jo Doherty, Alan Eustace, Ramsey Haddad, Robert Mayo, Suresh Menon, Louis Monier, Don Stark, Silvio Turrini, Leon Yang, John Fitch, William Hamburgen, Russell Kao, and Richard Swan.** WRL Research Report 93/8, December 1993.

‘‘Link-Time Optimization of Address Calculation on a 64-bit Architecture.’’ **Amitabh Srivastava, David W. Wall.** WRL Research Report 94/1, February 1994.

‘‘ATOM: A System for Building Customized Program Analysis Tools.’’ **Amitabh Srivastava, Alan Eustace.** WRL Research Report 94/2, March 1994.

‘‘Complexity/Performance Tradeoffs with Non-Blocking Loads.’’ **Keith I. Farkas, Norman P. Jouppi.** WRL Research Report 94/3, March 1994.

‘‘A Better Update Policy.’’ **Jeffrey C. Mogul.** WRL Research Report 94/4, April 1994.

‘‘Boolean Matching for Full-Custom ECL Gates.’’ **Robert N. Mayo, Herve Touati.** WRL Research Report 94/5, April 1994.

‘‘Software Methods for System Address Tracing: Implementation and Validation.’’ **J. Bradley Chen, David W. Wall, and Anita Borg.** WRL Research Report 94/6, September 1994.

‘‘Performance Implications of Multiple Pointer Sizes.’’ **Jeffrey C. Mogul, Joel F. Bartlett, Robert N. Mayo, and Amitabh Srivastava.** WRL Research Report 94/7, December 1994.

‘‘How Useful Are Non-blocking Loads, Stream Buffers, and Speculative Execution in Multiple Issue Processors?.’’ **Keith I. Farkas, Norman P. Jouppi, and Paul Chow.** WRL Research Report 94/8, December 1994.

‘‘Drip: A Schematic Drawing Interpreter.’’ **Ramsey W. Haddad.** WRL Research Report 95/1, March 1995.

‘‘Recursive Layout Generation.’’ **Louis M. Monier, Jeremy Dion.** WRL Research Report 95/2, March 1995.

‘‘Contour: A Tile-based Gridless Router.’’ **Jeremy Dion, Louis M. Monier.** WRL Research Report 95/3, March 1995.

‘‘The Case for Persistent-Connection HTTP.’’ **Jeffrey C. Mogul.** WRL Research Report 95/4, May 1995.

‘‘Network Behavior of a Busy Web Server and its Clients.’’ **Jeffrey C. Mogul.** WRL Research Report 95/5, October 1995.

‘‘The Predictability of Branches in Libraries.’’ **Brad Calder, Dirk Grunwald, and Amitabh Srivastava.** WRL Research Report 95/6, October 1995.

‘‘Shared Memory Consistency Models: A Tutorial.’’ **Sarita V. Adve, Kourosh Gharachorloo.** WRL Research Report 95/7, September 1995.

‘‘Eliminating Receive Livelock in an Interrupt-driven Kernel.’’ **Jeffrey C. Mogul and K. K. Ramakrishnan.** WRL Research Report 95/8, December 1995.

‘‘Memory Consistency Models for Shared-Memory Multiprocessors.’’ **Kourosh Gharachorloo.** WRL Research Report 95/9, December 1995.

‘‘Register File Design Considerations in Dynamically Scheduled Processors.’’ **Keith I. Farkas, Norman P. Jouppi, Paul Chow.** WRL Research Report 95/10, November 1995.

‘‘Optimization in Permutation Spaces.’’ **Silvio Turrini.** WRL Research Report 96/1, November 1996.

‘‘Shasta: A Low Overhead, Software-Only Approach for Supporting Fine-Grain Shared Memory.’’ **Daniel J. Scales, Kourosh Gharachorloo, and Chandramohan A. Thekkath.** WRL Research Report 96/2, November 1996.

‘‘Efficient Procedure Mapping using Cache Line Coloring.’’ **Amir H. Hashemi, David R. Kaeli, and Brad Calder.** WRL Research Report 96/3, October 1996.

‘‘Optimizations and Placement with the Genetic Workbench.’’ **Silvio Turrini.** WRL Research Report 96/4, November 1996.

WRL Technical Notes

‘‘TCP/IP PrintServer: Print Server Protocol.’’ **Brian K. Reid and Christopher A. Kent.** WRL Technical Note TN-4, September 1988.

‘‘TCP/IP PrintServer: Server Architecture and Implementation.’’ **Christopher A. Kent.** WRL Technical Note TN-7, November 1988.

‘‘Smart Code, Stupid Memory: A Fast X Server for a Dumb Color Frame Buffer.’’ **Joel McCormack.** WRL Technical Note TN-9, September 1989.

‘‘Why Aren’t Operating Systems Getting Faster As Fast As Hardware?.’’ **John Ousterhout.** WRL Technical Note TN-11, October 1989.

‘‘Mostly-Copying Garbage Collection Picks Up Generations and C++.’’ **Joel F. Bartlett.** WRL Technical Note TN-12, October 1989.

‘‘Characterization of Organic Illumination Systems.’’ **Bill Hamburgen, Jeff Mogul, Brian Reid, Alan Eustace, Richard Swan, Mary Jo Doherty, and Joel Bartlett.** WRL Technical Note TN-13, April 1989.

‘‘Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers.’’ **Norman P. Jouppi.** WRL Technical Note TN-14, March 1990.

‘‘Limits of Instruction-Level Parallelism.’’ **David W. Wall.** WRL Technical Note TN-15, December 1990.

‘‘The Effect of Context Switches on Cache Performance.’’ **Jeffrey C. Mogul and Anita Borg.** WRL Technical Note TN-16, December 1990.

‘‘MTOOL: A Method For Detecting Memory Bottlenecks.’’ **Aaron Goldberg and John Hennessy.** WRL Technical Note TN-17, December 1990.

‘‘Predicting Program Behavior Using Real or Estimated Profiles.’’ **David W. Wall.** WRL Technical Note TN-18, December 1990.

‘‘Cache Replacement with Dynamic Exclusion.’’ **Scott McFarling.** WRL Technical Note TN-22, November 1991.

‘‘Boiling Binary Mixtures at Subatmospheric Pressures.’’ **Wade R. McGillis, John S. Fitch, William R. Hamburgen, Van P. Carey.** WRL Technical Note TN-23, January 1992.

‘‘A Comparison of Acoustic and Infrared Inspection Techniques for Die Attach.’’ **John S. Fitch.** WRL Technical Note TN-24, January 1992.

‘‘TurboChannel Versatec Adapter.’’ **David Boggs.** WRL Technical Note TN-26, January 1992.

‘‘A Recovery Protocol For Spritely NFS.’’ **Jeffrey C. Mogul.** WRL Technical Note TN-27, April 1992.

‘‘Electrical Evaluation Of The BIPS-0 Package.’’ **Patrick D. Boyle.** WRL Technical Note TN-29, July 1992.

‘‘Transparent Controls for Interactive Graphics.’’ **Joel F. Bartlett.** WRL Technical Note TN-30, July 1992.

‘‘Design Tools for BIPS-0.’’ **Jeremy Dion & Louis Monier.** WRL Technical Note TN-32, December 1992.

‘‘Link-Time Optimization of Address Calculation on a 64-Bit Architecture.’’ **Amitabh Srivastava and David W. Wall.** WRL Technical Note TN-35, June 1993.

‘‘Combining Branch Predictors.’’ **Scott McFarling.** WRL Technical Note TN-36, June 1993.

‘‘Boolean Matching for Full-Custom ECL Gates.’’ **Robert N. Mayo and Herve Touati.** WRL Technical Note TN-37, June 1993.

‘‘Piecewise Linear Models for Rsim.’’ **Russell Kao, Mark Horowitz.** WRL Technical Note TN-40, December 1993.

‘‘Speculative Execution and Instruction-Level Parallelism.’’ **David W. Wall.** WRL Technical Note TN-42, March 1994.

‘‘Ramonamap - An Example of Graphical Groupware.’’ **Joel F. Bartlett.** WRL Technical Note TN-43, December 1994.

‘‘ATOM: A Flexible Interface for Building High Performance Program Analysis Tools.’’ **Alan Eustace and Amitabh Srivastava.** WRL Technical Note TN-44, July 1994.

‘‘Circuit and Process Directions for Low-Voltage Swing Submicron BiCMOS.’’ **Norman P. Jouppi, Suresh Menon, and Stefanos Sidiropoulos.** WRL Technical Note TN-45, March 1994.

‘‘Experience with a Wireless World Wide Web Client.’’ **Joel F. Bartlett.** WRL Technical Note TN-46, March 1995.

‘‘I/O Component Characterization for I/O Cache Designs.’’ **Kathy J. Richardson.** WRL Technical Note TN-47, April 1995.

‘‘Attribute caches.’’ **Kathy J. Richardson, Michael J. Flynn.** WRL Technical Note TN-48, April 1995.

‘‘Operating Systems Support for Busy Internet Servers.’’ **Jeffrey C. Mogul.** WRL Technical Note TN-49, May 1995.

‘‘The Predictability of Libraries.’’ **Brad Calder, Dirk Grunwald, Amitabh Srivastava.** WRL Technical Note TN-50, July 1995.

WRL Research Reports and Technical Notes are available on the World Wide Web, from `http://www.research.digital.com/wrl/techreports/index.html`.