Supervised Floorplanning

Thu Cung, Stephen Tarzia, and Kenneth Wade
March 21, 2008

***Abstract*—In this work, we have attempted to test the usefulness of augmenting a search algorithm with *human supervision* for the floorplan design optimization problem, a classical NP-hard problem in the field of Electronic Design Automation. We have built floorplan visualization tools and a web interface [1] that allows a user to rank a current pool of solutions, thus guiding the search. Results show little quantitative difference between user-guided and metric-guided searches. However, some qualitative differences between results of the two experiments are apparent. We discuss the implications of this experiment and future directions that this research may take.**

I. Introduction

A. Floorplanning

Floorplanning is a decades-old problem in the design of integrated circuits (computer chips). A circuit can be thought of as a set of *components* and a set of connections that must be made between those components, called *nets*. In order to manufacture a circuit on a chip die, the circuit's components must be arranged into the rectangular chip outline. The cost of manufacturing a chip is super-linearly dependent on area because fewer larger chip dies can be replicated on the same silicon wafer and because larger chips are more likely to be damaged by random imperfections on the manufactured wafer. A chip's performance (that is, the maximum frequency at which it can be clocked) and power consumption depend on the length of metal wires interconnecting its components; shorter is better.

The floorplanning problem is a mathematical formalization of the above design scenario [10]. A *floorplan* is a non-overlapping arrangement of rectangles on the plane. The quality of a floorplan is measured by both the area of its bounding-box (the smallest rectangle that contains it) and an estimate of its wirelength. Wirelength cannot be quickly determined, since wire routing is a very complex task involving many interdependent forking and detouring decisions[1]. However, we can estimate it by the sum of each net's half perimiter wirelength (HPWL). HPWL is the height plus width of the bounding box of a net's members. Most often, the goal is of floorplanning is to generate a floorplan while minimizing the a weighted sum of area and HPWL.

Traditional floorplanning algorithms have been very successful at solving small problem instances, up to several hundred blocks. Some of the earliest algorithms were constructive; they queue the blocks and then build a solution by placing one block at a time until all are placed. More sophisticated algorithms consider more than one complete solution. In the *search* approach to optimization, a solution can be permuted

[1]In fact, routing even a single net is an NP-Complete problem known as the minimal Steiner tree problem.

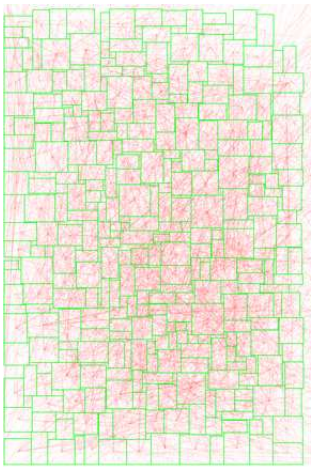


Fig. 1. A good floorplanning problem solution. Problems of this size (300 blocks) are easily solved by simulated annealing.

several ways to produce neighbor solutions. A search considers a sequence of solution neighbors and finally chooses the best one seen during this solution space traversal. The most widely used floorplanning technique is simulated annealing [6]. Simulated annealing is a type of search that initially moves to random neighbors but eventually moves only to *better* neighbors; the transition from random moves to *gradient descent* happens gradually and is controlled by the *temperature* variable.

However, there are more difficult floorplanning variants that have not been adequately solved. The most obvious class of difficult floorplanning problems is very large problems, with thousands of blocks. In the past, floorplanning problems did not enlarge even while the transistor count in circuits exploded; designers simply used higher levels of abstraction to define blocks. However, the advent of macro-blocks, large pre-designed blocks often licensed from a third party, has changed that. There is now a need to place circuits composed of perhaps several macro-blocks and thousands of small blocks representing custom "glue" circuitry; traditional floorplanners cannot solve these problems [9].

Another class of more difficult floorplanning variants includes additional constraints or minimization objectives For example, it may be easier to manufacture a squarer chip; thus, we may have an aspect ratio constraint. Alternatively, we may seek to limit the peak temperature of the chip by spreading out the hottest-running blocks. There may also be some critical path wirelength that must be kept below a certain threshold to meet some timing requirement. There is virtually no limit to the complexities that we can add to a floorplanning problem.

B. Hypothesis

We hypothesize that a user will be able to predict the long-term quality of block arrangements under continued search when presented with a meaningful visualization; we believe that feedback from the user can be used to more quickly find good solutions in the course of an optimization search. We proceed as follows. In section I-C, we summarize influential past work; we describe our experiment and results in sections II and III; finally, we draw conclusions and propose extensions to this work in sections IV and V.

C. Related Work

a) Floorplan visualization: There is a long tradition of floorplan visualization in the literature. Most papers presenting floorplanning algorithms include sample illustrations of the floorplans that they produce in addition to the standard collection of area and wirelength statistics. This practice seems to imply a recognition in the community that metrics alone do not entirely capture the quality of a floorplan. There has also been some work on visualization for manual chip design tools [4].

b) Supervisory control: The *human factors* community has defined a mode of human-machine interaction called *supervisory control* [12]. This scheme is based on the recognition that humans and computers each excel at different types of tasks [11]. The machine controls low-level functions and provides both visual feedback and control "knobs" to the human. The human interprets the machine and environment state and directs the machine accordingly. Such a system has the advantage of being both adaptive and perhaps simpler to build than a fully automated system. Along the axis of increasing problem difficulty there is a point beyond which the time needed to program a fully-automated algorithm exceeds the aggregate time needed for humans to solve each problem manually.

c) Artificial intelligence: von Ahn's work demonstrates the efficacy of human computation in solving AI problems [13]–[16]. However, von Ahn's formal model of a human algorithm game as an input to output mechanism is limiting [13, chapter 7]; it presupposes a well defined, and thus easily checkable, problem. Many practical problems are ill-posed; the value of a particular solution depends on a vast amount of context which cannot be effectively encoded into any known algorithm. In these cases, the primary difficulty is in refining the problem definition into one that has a unique solution which is most valuable in the current context. This refinement step is a hard AI problem and is thus amenable to human computation. A human in the loop can play the role of supervisor, adjusting goals and evaluating progress of some underlying computation.

d) Human directed search: Lin and Dinda's work on virtual machine scheduling [7] entails the use of direct user input to improve scheduling decisions. Users provided feedback regarding their "discomfort levels" while carrying out tasks on everyday applications in a controlled environment. The feedback gathered furthered scheduling procedures in that it is clear users tolerance levels vary drastically depending on the user and task being performed. Achieving a *universally* optimal scheduling model is thus paradoxical an optimal scheduling solution cannot be universal as it must adapt to specific user preferences, which have few commonalities. Optimization can be achieved by tailoring scheduling preferences to users based on his/her individual preferences.

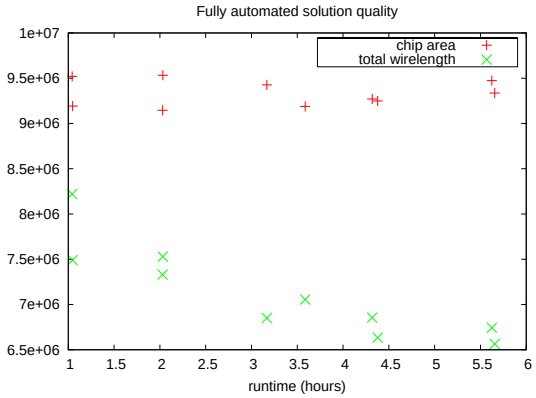


Fig. 2. Long runtime solution quality

II. Methodology

To test our hypothesis, we perform a simple experiment using an existing floorplanner and set of custom visualization and user interface scripts. Using a single problem input, we generate a pool of twenty candidate solutions. The quality of the solutions is then ranked; in the experimental setup a human user does this ranking and in the the control setup solutions are ranked by wirelength. The best-ranked five candidates are then used to create the next pool of twenty candidates; each of the five candidates is used as a starting point for the floorplanner four times. The process is then repeated beginning with the ranking of the new pool of twenty candidates. In the following subsections we describe our experimental setup in more detail.

A. Floorplanning framework

In our experiments we use the Parquet floorplanner, a high-quality academic code [3]. Parquet uses simulated annealing and is highly configurable. In our experiments, we set the algorithm runtime to ten minutes and set the annealing temperature $T = T_o/2^r$ where r is the current round of ranking. By lowering the initial temperature every round, we are enforcing less randomness in the search in later rounds.

The sample problem *ibm02* that we use is taken from a recent publication on difficult floorplanning instances [2], [9]. It contains 1471 blocks, 8508 nets, and the ratio of largest to smallest block area is 3004.3. Figure 2 indicates the difficulty of this problem; the simulated annealing search still makes slow progress after several hours of runtime.

A MySQL database stores an entry for each candidate with columns for solution statistics and user rankings.

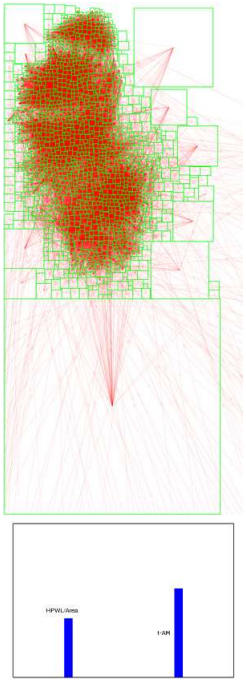


Fig. 3. A floorplan visualized. More examples can be seen at the website [1] and in figure 4.

B. Visualization

The Parquet floor planner we used considered area and wire length when generating floor plans. Thus an optimal floor plan would have minimal area and require the least amount of wire length. If these two metrics were minimized, white space (area which components have not been assigned to) would be optimized as well. Aspect ratio (how "square" the dimensions of the floor plan are) would not necessarily be optimized. Parquet provided numerical values for these four metrics for each floor plan. Unfortunately, the changes in area and wire length from floor plan to floor plan were not easily distinguishable. In order to allow for effective integration of user rankings, we needed a method to depict these metrics visually, so as to successfully engage users.

To create the visualizations, we used gnuplot 4.0, a command-driven, function and data plotting program. gnuplot is most used for its ability to take data as input and then output the data graphically. We used both gnuplots ability to generate graphs and also used the program to generate visualizations of our floor plans.

Visual depictions of the area and wires, whether through images or graphical, were difficult. Because the images and graphs were capturing such a large quantity of information, oftentimes showing these numbers visually resulted in cluttered and, consequently, useless images. The visualizations thus needed to relay these metrics in such a way that users can easily distinguish how the metrics vary between the different floor plans.

To illustrate both area and wire length, our visualizations displayed both the layout of the individual components as well as their corresponding wires (see Figure 3, top). Users could see how the components were organized and also see the extension and intersection of wires. To generate these images, a script modifies the basic gnuplot visualization that Parquet already provides.

When comparing later generations of floor plans to earlier ones, it is easy to distinguish between the different floor plans (see Figures 4 a, c). There is a noticeable difference between the saturation of the red of the wires. However, when comparing floor plans of the same generation (see Figure 4 a, b), it is difficult to visually determine the changes of the metrics between different floor plans of the same generation.

To capture the differences in area and wire length of the visualizations, we decided to include graphs. One idea was to display a scatter plot of wire length against the number of intersections per wire. However, it was difficult to visually determine differences in the graphs of the floor plans.

We decided to use bar graphs (see Figure 3, bottom). The bar graphs depict two measurements: the ratio of total wire length (HPWL) to total area and the aspect ratio (variation of aspect ratio from 1). These graphs visualize the actual values of the metrics, allowing users to compare the floor plans based on each designs measurements. The bars of these graphs managed to capture the metric differences of the floor plans. We used gnuplot and C++ to create the bargraphs. The C++ program takes a file and values as input and generates .plt and .dat files, which gnuplot uses to create the graphs themselves.

A PHP script calling ImageMagick was used to resize and join the gnuplot figures.

C. User interface

In order to get the user input to guide the creation of the next generations of the floorplan, the website allows a user to rate a current generation's floorplan [1]. For a given benchmark, the system randomly selects four sets of visualizations and bar graphs and presents them to the user. Each visualization illustrates to the user the actual layout of the components and corresponding wires, while the paired bar graph has information about the total wire length, the total area, and variation of the aspect ratio.

Upon inspection of the visualization and mentally processing the data presented by the bar graph, the user is able to make a decision about which of the four presented floorplans occupies the smallest area and whose wire length is also minimized. The best of the four is marked "best" and the worst of the four is marked "worst." The remaining floorplans are simply left alone for that iteration.

The best and work ratings are translated into rankings using the following formula:

$$(best\_count - worst\_count)/total\_count$$

where $total\_count$ is the number of times that the candidate was presented to the user for rating. If $total\_count$ is zero, then that candidate is ranked at the bottom of the list of best solutions. The interface was implemented in PHP on an Apache web server with connections to the MySQL database described above.

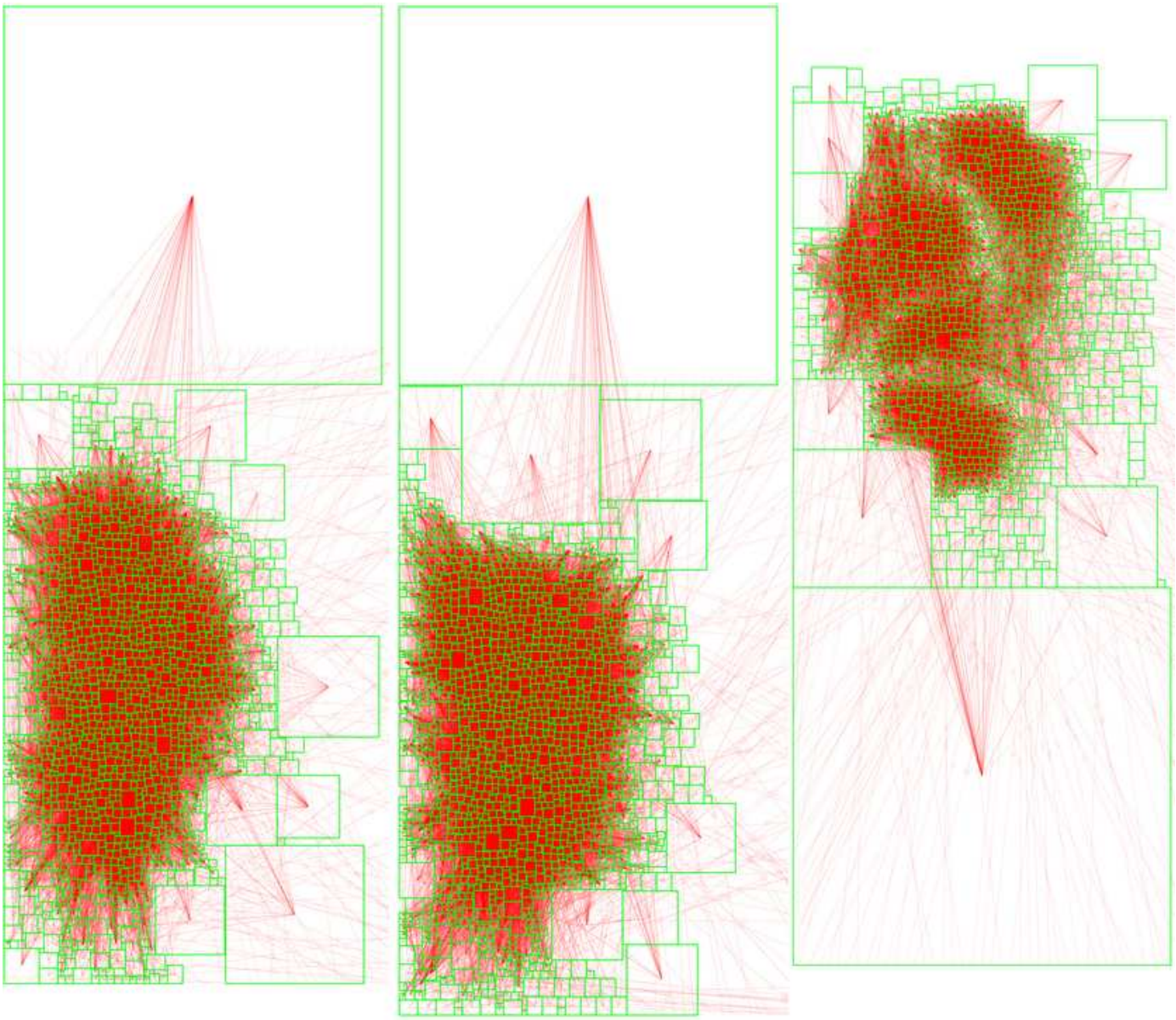
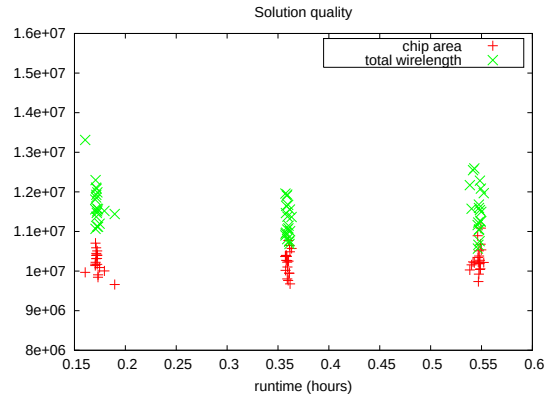
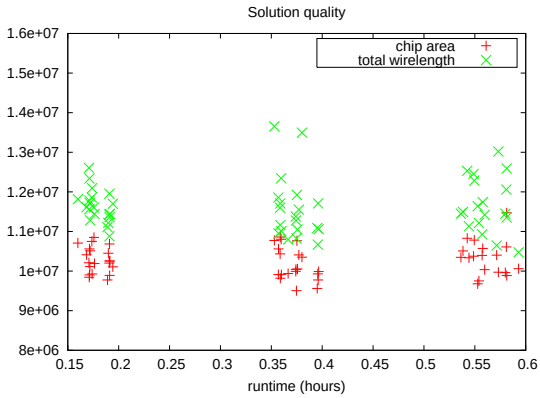


Fig. 4. (a) and (b) are visualizations from the initial candidate pool. (c) is from a much later stage.



(a)



(b)

Fig. 5. Automated results (a) and Supervised results (b)

III. Results

We captured the area and total wire length of each floor plan at different generations for both the automated, Parquet-generated floor plans as well as those which involved user-input. Graphs of these results are shown in Figures III a (automated) and III b (user-input).

The performance overhead of generating the visualizations was significant, but not overwhelming. Total runtime was about 65 seconds. The major bottleneck in visualization was in rendering an EPS figure to a very high resolution bitmap. This operation took about 45 seconds and was required in order to get the very fine lines representing floorplan wires. Image and floorplan storage overhead is minimal: about 1.5 Mbyte per candidate.

IV. Conclusions

There is relatively little difference between the resulting metrics of the automated approach and the user-ranked approach. Although the user rankings for the later generation are not as uniform as the automated rankings, the values of the chip area and cumulative wire length align very closely to the values that resulted from Parquet.

While we had anticipated that user rankings would allow Parquet to arrive at an optimized solution faster, our results indicate otherwise. User involvement in the process was about equally effective as the automated approach, in arriving at an optimal solution.

Although this does not align with what we had hypothesized, it does not disprove that user-involvement throughout floor planning is not useful. If anything, user involvement can be as equally effective as automated programs in designing floor plans.

In addition, the user driven results seem to be *qualitatively different*. In particular, their aspect ratios were squarer. The average and maximum AR of the user driven results was 0.467 and 0.847, respectively. This compares favorably to the values of 0.439 and 0.805 obtained in the control group. This is a good example of the problem objectives being influenced by the human supervisor; aspect ratio was not explicitly a floorplanning goal for Parquet.

Looking at the results of figure III, the lack of quantitative solution improvement over the three generations is troubling. This seems to indicate a flaw in our experimental setup. Different values for the generation runtime, initial temperature and temperature dropoff rate may be beneficial.

V. Future Work

Improving the visualizations is an area we would like to explore. Visualizations should allow for easier distinction between the level of optimization of each floor plan.

To better capture area, we may want to allow users the option of only viewing the outline of the component layout, as opposed to each individual component. If we were to only give an outline of the chip area overall, we would include the value of whitespace for each floor plan as well, so that users would have a way of knowing how tightly packed the components were.

With the current visualizations, we would like to capture how saturated each image is with the color redessentially the wire length of the floor plan. Because saturation lessens with each generation of floor plans, depicting the level of saturation alongside each floor plan within a given generation could prove useful.

Additionally, creating animated GIFs of the floor plans could also help users. Rather than depicting all of the wires on a single image, we could create separate images for short, medium, and long wires. Color-coding these wires and allowing users to layer these images could prove useful.

As an additional experiment, one could let the automated run choose random starting solutions for the next generation rather than those with the shortest wirelength. Also, the user could play a different role, choosing the runtime parameters for floorplanner in next stage; as we have said, the Parquet floorplanner is highly configurable.

There are also alternatives to simulated annealing that might be better suited to human involvement. A genetic algorithm [5] [8] keeps a pool of solutions that are “bred” to create the next generation. Traditionally, a cost function determines which solutions produce offspring and which perish; a human could make these decisions instead.

A. Floorplanning as a game

Currently, the solution provided to the floorplanning problem sparks limited interest outside of an experienced floorplanner that has an idea of what they want. Presenting this problem as a game that is analogous to a simplified city planning game can allow for more difficult floorplanning with additional constraints and/or minimization objectives. For example, crime rate could represent temperature at various points on the board, roads and rails could represent the wire length or density and so on.

Since the floorplanning problems we are solving will be complex, large scale cities will be presented to the user in a particular state instead of being built from the ground up. The user would not directly control building placements, but they would instead influence the evolution of the city.

The goal of a typical city planning game is to make as much money as possible with a single city that the user has constructed over a long period of time. For our game, there can and will be several different goals for different stages. The user would be able to identify problem regions within the city that are preventing the goal from being achieved. If the primary goal for a stage were to lower the crime rate in all areas of the city that were above a certain rate without adding more Police stations, the user could select types of buildings that attract crime and let the system focus on changing them through the city’s next evolution. If the primary goal for a stage were to maximize traffic flow between certain “high priority” regions, the user could select road segments and rules that would prioritize the change

More points would be awarded if various city goals are achieved in fewer evolutions of the city. The evolutions could be presented to the user as either the same city at the same time in a parallel dimension, or the same city at a point not

to far in time. Instead of directly manipulating buildings and infrastructure, it would be that the user's policies will affect the way the city evolves in the near future. We could also limit the amount of real-world time that a user can spend making decisions through the use of a countdown timer, which would give the user some since of urgency and allow them to be more efficient at playing the game as they become more familiar. In addition to the time restriction, a limit could be placed on the number of "policies" that can be applied to a generation for it's next evolutionary iteration.

B. Beyond Floorplanning

Floorplanning was chosen as a target problem for supervised search simply because of one authors past experience with the problem. However, we do not claim that it is the only or the best problem for supervised search. We believe that there is plenty of exciting work to be done on supervised search in other computing fields as well.

REFERENCES

[1] http://belmont.eecs.northwestern.edu/cgi-bin/floorplan/index.php.

[2] http://vlsicad.eecs.umich.edu/bk/ispd06bench.

[3] S. N. Adya and I. L. Markov. Fixed-outline floorplanning : Enabling hierarchical design. *IEEE Trans. on VLSI Systems*, 11(6):1120–1135, December 2003.

[4] D.L. DeMaris. Visualization in a vlsi design automation system. *IBM J. Res. Develop.*, 35(1/2):238–243, January/March 1991.

[5] D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[6] S. Kirkpatrick, Jr. C.D. Delatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, May 1983.

[7] B. Lin. *Human-driven Optimization*. PhD thesis, Northwestern Univ., 2007.

[8] S. Nakaya, T. Koide, and S. Wakabayashi. An adaptive genetic alforithm for vlsi floorplanning based on sequence-pair. In *proc. Intl. Sym. on Circuits and Systems*. IEEE, May 2000.

[9] A.N. Ng, I.L. Markov, R. Aggarwal, and V. Ramachandran. Solving hard instances of floorplacement. In *proc. Intl. Sym. Physical Design*. ACM, April 2006.

[10] S.M. Sait and H. Youssef. *VLSI Physical Design Automation*, chapter 3: Floorplanning. World Scientific, 1999.

[11] T.B. Sheridan. Functional allocation: algorithm, alchemy or apostasy? *Int. J. Human-Computer Studies*, 52:203–216, 2000.

[12] T.B. Sheridan. *Handbook of Human Factors and Ergonomics*, chapter 38: Supervisory Control, pages 1025–1052. Wiley, 3rd edition edition, 2006.

[13] L. von Ahn. *Human Compuation*. PhD thesis, CMU, December 2005.

[14] L. von Ahn. Games with a purpose. *IEEE Computer*, 39(6), June 2006.

[15] L. von Ahn and L. Dabbish. Labeling images with a computer game. In *proc. CHI 2004*, 2004.

[16] L. von Ahn, R. Liu, and M. Blum. Peekaboom: A game for locating objects in images. In *proc. CHI 2006*, 2006.