Optimizing Windows Layout by Applying a Genetic Algorithm

Nihar Trivedi
PricewaterhouseCoopers LLP.
Level 18, 201 Sussex Street,
Sydney, NSW 2000, Australia
nihar.trivedi@au.pwcglobal.com

Wei Lai
School of Information Technology
Swinburne University of Technology
PO Box 218, Hawthorn
Vic 3122, Australia

Zhongwei Zhang
Dept. of Maths & Computing
University of Southern Queensland
Toowoomba, Qld 4350, Australia
Zhongwei@usq.edu.au

Abstract- Most windows based systems leave the task of windows layout management to the user. This inadvertently tends to decrease the productivity of the user. This paper presents a Genetic Algorithm (GA) based approach to windows layout optimisation. A new shadow-based algorithm is proposed to remove the overlap from the windows and to compact the layout of the windows. The usefulness of this approach is illustrated in this paper with the samples of the generated layouts, the characteristic charts and possible enhancements in the future.

1 Introduction

A good user interface enables a user to cope with the information volume and helps a user to selectively extract the relevant information. Windows based Interface (WI) enables a user to simultaneously perform multiple tasks and to view their results, but the task of windows' layout management is left to the user. Several studies have suggested that the manual window layout management tends to increase the overall time taken to complete a task [3, 8]. Also the window overlaps introduced in the WI tend to decrease the productivity of the user [3, 8].

The problem we are interested in is to devise an approach to automatic, overlap free arrangement of the windows based on the user's interactions with the WI. This problem is similar to the Very Large Scale Integrated (VLSI) circuit layout problem. The line sweeping algorithm, the enhanced plane sweep method, and shift compaction algorithm are some of the relevant algorithms dealing with the problem [1, 4, 11]. However we are unable to apply these techniques to WI because the number and the size of the windows are dynamic and the display area size is limited.

Peter Luders [8] considers the window layout as a NP complete combinatorial optimisation problem. In the first phase of computing the solution, the windows are placed on a grid assuming them to be of the identical size. In the second phase the sizes of the windows are modified to generate the solution layout. However, this method is designed for non-interactive mode and may generate an overlapped windows layout during the second phase of the process.

An evolutionary approach is proposed in this paper. We first discuss some of the characteristics of windows layout problem, followed by the description of our Genetic Algorithm (GA) based solution to the problem. Finally we present the results, the performance analysis and the final summary.

2 Windows Layout Problem

For our windows layout problem, we are interested in developing an algorithm that can arrange any number of windows in a finite display area such that they do not overlap and occupy as smaller area as possible. In applications, each window's size is often changed. For example, when a user interact with a window for editing a text file, he/she wants this focused window's size becomes larger and other windows become smaller. We set up a criteria for arranging these dynamic windows in a limited size of computer screen such that the windows should not overlap each other. This can help the user easily select the windows.

The process of finding the solution of the layout would involve non-determinism. Based on the definition of NP completeness [10], this problem is NP complete. Ample amount of literature can be found that discuss the theory of NP completeness and solving such problems using Genetic Algorithms (GA), Simulated Annealing (SA), etc. Next follows the discussion about our implementation of GA.

3 Hybrid Genetic Algorithm

As a representative of evolutionary paradigm, Genetic Algorithms are adaptive methods that are useful and appropriate for solving search and optimisation problems. Based on the genetic reproducing process of biological organisms. Over many generations, natural populations evolve according to the principles of natural selection and "survival of the fittest", first clearly stated by Charles Darwin in *The Origin of Species*. [2]

0-7803-6657-3/01/$10.00 ©2001 IEEE

The analysis of the SA technique and its comparison with GA by Lin, Kao and Hsu [5, 6, 7] has revealed that incorporating GA with SA results in the design of an annealing schedule that improve the performance of the SA technique [7]. The Annealing Genetic (AG) technique proposed by Lin, Kao and Hsu generates a temporary population of strings at every epoch. Each string is randomly modified and added to the temporary population with some probability derived from the change in the fitness of a string. The genetic operators are then applied to the temporary population to generate the next population. The process continues till 80% of strings in the population are identical [7].

For solving the windows display layout problem, it is necessary that each string in a population represent a likely solution. Hence, each solution string (called a genotype in GA term) in the population must represent a non-overlapping, compact windows layout. For example, suppose there are n windows opened by a user, then an array of integers represents this display layout as

$$id_1, left_1, top_1, width_1, height_1, \ldots, id_i, left_i, top_i, width_i, height_i, \ldots, id_n, left_n, top_n, width_n, height_n$$

Where

id identifies a window,
left and *top* represent the position (x,y) of a window's left-top corner,
width and *height* represent a window's size.

and

$$left_1 + \frac{width_1}{2} \le \ldots \le left_i + \frac{width_i}{2} \le \ldots \le left_n + \frac{width_n}{2}$$

or

$$top_1 + \frac{height_1}{2} \le \ldots \le top_i + \frac{height_i}{2} \le \ldots \le top_n + \frac{height_n}{2}$$

and

$$id_1 \ne id_2 \ne \ldots \ne id_{i-1} \ne id_i \ne id_{i+1} \ne \ldots \ne id_n$$

There are several compact layout scenarios for these windows depending upon the compaction direction, the location and the sizes of the windows. Individual solution strings are likely to have different display area usage. The Annealing Genetic algorithm operates on a set of candidate solution strings by applying Crossover and Mutation genetic operators to find out optimization of the windows display.

Regarding our intention to display disjoint windows, the random initialisation of a solution string may violate this condition. Therefore we have developed Shadow Propagation for Overlap Removal and Display Area Compaction (SPORDAC) genetic operator to correct the faulty solution string. Normal mutation and crossover operators are also customised to adhere to the above condition. Next follows the description of the SPORDAC, mutation and crossover operators.

3.1 The SPORDAC Operator

The SPORDAC is a one-dimensional technique that removes overlap and compacts the layout in the compaction direction at a time. The SPORDAC assumes that a window extends its shadows in both horizontal and vertical directions as shown in the following figure.

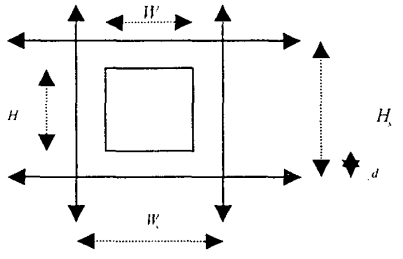


Figure 1 Shadows extended by a window of height H and width W with inter-window gap d.

$H_s = H + 2d$ ....(1)

$W_s = W + 2d$ ....(2)

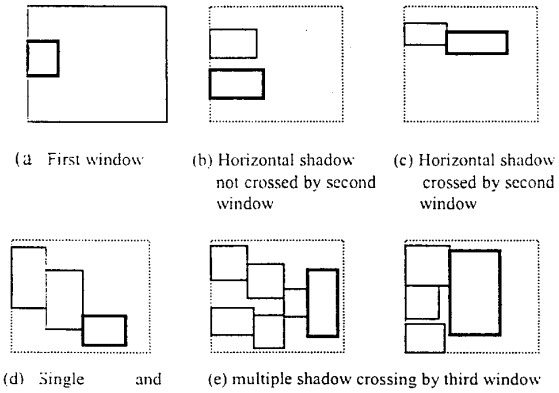
The SPORDAC only considers the shadows extended in the compaction direction. The shadows extended in the non-compaction direction are ignored. If two windows with their centres at (X_1, Y_1) and (X_2, Y_2) are overlapping then we can say that,

$$|X_1 - X_2| < \frac{W_1}{2} + \frac{W_2}{2} + d$$

This equation is utilised for overlap removal. A similar equation can be derived for the Y co-ordinates.

3.1.1 Overlap removal with SPORDAC

The SPORDAC initially places all windows on a virtual display area of indefinite height and width and then maps the virtual layout on the physical layout using a simple scaling operation. To generate the virtual layout, the SPORDAC begins with sorting the windows in the increasing value of compaction direction coordinate (ie. X or Y) of their centres. Next, each window is scanned from the sorted list and placed on the virtual display. This process continues till the sorted list is exhausted. Before positioning a window in the virtual display, the SPORDAC checks whether the current window crosses the compaction direction shadows of already placed windows or not. If such shadows are crossed then a short-list of windows is prepared that are on the periphery of the already placed windows in the virtual display. The current window is then placed next to the right most or the bottom most window in the short-list, depending upon the compaction direction.



(a First window (b) Horizontal shadow not crossed by second window (c) Horizontal shadow crossed by second window

(d) Single and (e) multiple shadow crossing by third window

Figure 2 Horizontal direction overlap removal and compaction by SPORDAC

3.1.2 The SPORDAC Algorithm

The SPORDAC operator is applied to every string in the population in both directions to produce a non-overlapping, compact display layout string.
The SPORDAC algorithm for X direction compaction is as outlined below.

1.0 Sort all windows in ascending order of X coordinate of their centre.
2.0 While (total scanned windows <= total windows)
  2.1 Read next window from the sorted window list.
  2.2 Prepare a short list of windows whose X direction shadows are crossed by current window.
  2.3 Find the window with highest value of right edge from the short list.
    Place the current window next to this window.
  2.4 If the short list found in step 2.2 is empty then initialise the left coordinate of the current window with the value of inter window gap.
  2.5 Add current window to scan list.
3.0 wend

The SPORDAC operator is applied to every string in the population in both directions.

3.2 Mutation Operator

Mutation is a way of generating a new solution by changing a part of an old solution. The solution string is represented as an array of integers holding the left and top edges' coordinates. The mutation operator randomly selects a window and randomly modifies the left and top edges' coordinates of the window. This may generate an invalid solution string. The SPORDAC operator is applied to the sibling string to compact the layout.

The graphical operation of the mutation operator is as shown in the figure 3 below. For example, suppose that the mutation operator is to operate on a string that represents following display layout.

The initial windows layout.

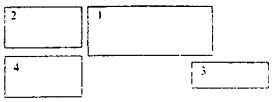
The genetic string representing above display layout could be as shown below

2,0,0,100,50,4,0,60,100,1,110,0,200,60,3,320,20,100,40.

The mutation operator randomly selects a window and randomly modifies either left or top edge value of the selected window. Suppose the operator selects the last window and its top edge and left edge values to modify, the modified integer string is shown below.

2,0,0,100,50,4,0,60,100,1,110,0,200,60,3,280,40,100,80.

Then the corresponding windows layout is as below.



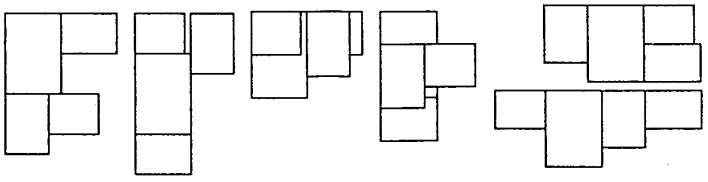
Layout after a mutation on window 3.

Figure 3. Mutation

3.3 Crossover Operator

Crossover is a process that generating new solutions by exchanging part of old solutions. It is understandable that every solution string represents the same number of windows. However, because of the SPORDAC operator applied to each string, it is possible that windows are placed in different order in different solution strings. Therefore implementing the crossover operator on the basis of string location may result in incorrect offspring. Hence the crossover operator is designed to operate on the basis of window ID. For example, if there are four windows in the string and if second window is selected as crossover point then the windows from the third window onwards are swapped to generate offspring.

The following figure gives graphical illustration of crossover operator.



(a) Parent strings (b) Offsprings (c) Compaction

Figure 4 Crossover

3.4 Fitness Assessment of Windows Layout

The fitness of each string is determined by the formula:

$$Cost\_of\_String = \frac{Void\_area}{Display\_area}$$

The display area is the area of the bounding rectangle for the layout represented by the string and the void area is the unoccupied area in the display area. The above mentioned genetic operators were integrated with the Annealing Genetic (AG) approach suggested by Lin, Kao and Hsu for windows layout optimisation. [7]

4 Results

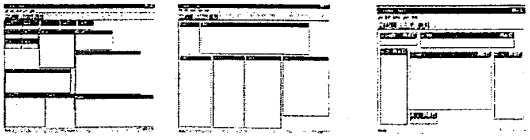
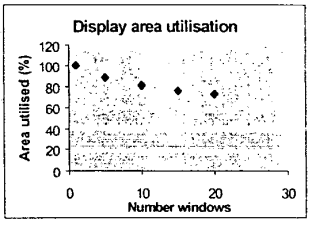


Figure 5 Sample layouts

The software prototype was implemented on a 133 MHz, Pentium machine. The sample layouts generated by our approach are shown in the figure 5. When a user interacts with a window, its width and height are increased by 15%. The other windows decrease their widths and heights by 15%. The layout optimisation is performed next. We can observe from the figure that our approach has been successful in producing compact, non-overlapping windows



layout.

5 Performance

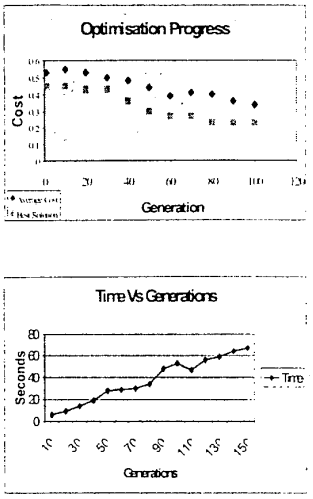


Figure 6 Performance characteristics

The figure 6 depicts the optimisation characteristics observed for our approach. The figure suggests that the optimisation achieves about 93% display area utilisation in the best case. The display area utilisation was observed to be 65% in the worst case. The optimisation plot was generated with the values of $P_c = 0.05$ and $P_m = 0.005$. The optimisation progress curve resembles in nature to the plot produced by Lin, Kao and Hsu for set partitioning problem.

6 Summary

The development of a new 'shadow' based algorithm for overlap removal and display layout compaction and its successful integration with GA for window layout optimisation has been our main achievement. The extension of the SPORDAC genetic operator to simultaneously update the layout in two dimensions could be an interesting topic for further research in the area. The optimisation function could be enhanced to include other relevant constraints such as mental-map preservation [9] for the change of windows layout.

References

[1] Awashima, T., Sato, M., and Ohtsuki, T. (1993). Optimal consraint graph generation algorithm for layout compaction using enhanced plane-sweep method. *IEICE Transactions on fundamentals of electronic, 76(4)*. 507-512.

[2] Beasley, D., Bull, D., Martin, R. (1993). An Overview of Genetic Algorithms : Part 1 Fundamentals. *University Computing, 15(2)*. 58-69

[3] Funke, D., Neal, J., and Paul, R. (1993). An approach to intelligent automated window management. International journal of man-machine studies, 38. 949-983.

[4] Hsiao, P., and Feng, W. (1990). New algorithms based on multiple storage quadtree for hierarchical compaction of VLSI mask layout. *Computer aided design, 22(2)*. 74-80.

[5] Jain, S., Gea, H., (1996). PCB Layout Design Using a Genetic Algorithm. *Journalof Electronic Packaging, 118(1)*. 11-15.

[6] Koza, J. (1992). *Genetic Programming : On the Programming of Computers by Means of Natural Selection.*. Massachusetts Institute of Technology.

[7] Lin,F., Kao, C., and Hsu, C. (Nov. 1993). Applying genetic approach to simulated annealing in solving some NP hard problems. *IEEE Transactions on systems, man and cybernetics, 23(6)*. 1753-1767.

[8] Luders, P., Ernst, R., and Stille, S. (1995). An approach to automatic display layout using combinatorial optimisation algorithms. *Software-Practice and experience, 25(11)*. 1183-1202.

[9] Misue, K., Eades, P., Lai, W., and Sugiyama, K., (1995). Layout Adjustment and the Mental Map. *Journal of Visual Languages and Computing*, 6, 183-210.

[10] Pfleeger, C. (1989). *Security in Computing*. Prentice Hall Inc. New Jersey.

[11] Sakamoto, M., Onodera, H., and Tamaru, Keikichi. (1990). Shiftcompaction – Quasi-Two-Dimensional compaction method for symbolic layout. *Electronics and communications in Japan, Part 3, 73(9)*. 40-51.