An approach to display layout of dynamic windows

Nihar Trivedi
Edith Cowan University

An Approach to Display Layout of Dynamic Windows

By

Nihar Trivedi

A Thesis Submitted in Partial Fulfilment of the Requirements for the

Award of

Master of Science (Computer Science)

Supervisors

Dr. Wei Lai (University of South Queensland)

Dr. Jim Millar (Edith Cowan University)

At the Faculty Science and Technology, Edith Cowan University

Date of Submission : 12th of December, 1997.

Abstract

The development of windows based user interface has introduced a new dimension to the field of human computer interaction. Now a user is able to perform multiple tasks at a time, often switching from one task to another. However windows environment also imposes the burden of manual windows management on the user. Several studies have suggested that manual window management is an unproductive chore often resulting in clutter and confusion on the display screen. Therefore we need a automatic windows layout generator to free the user to perform other useful tasks.

This thesis introduces SPORDAC (Shadow Propagation for Overlap Removal and Display Area Compaction) algorithm. This algorithm aims to remove overlap from the display layout and encapsulate the layout in the finite display area. The SPORDAC prototype integrates the SPORDAC algorithm with simulated annealing to optimise the display area usage. The usefulness and applicability of the SPORDAC approach are illustrated with the implementation of a prototype, samples of generated layouts and analysis of the collected data.

I certify that this thesis does not incorporate without acknowledgment any material previously submitted for a degree or diploma in any institution of higher education; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person except where due reference is made in text.

Signature:

12/21'98

Date:

Acknowledgement

I am very grateful to Dr Jim Cross, Dr Wei Lai, Dr Jim Millar, Dr Craig Standing and my colleagues at CAZ Software for their encouragement, guidance and support towards the completion of this thesis.

TABLE OF CONTENTS

CHAPTER: 1

Introduction

One of the main goals of a good user interface design is to enable the user to cope with information volume. It may happen that users are under stress and often inexperienced for the task at hand. Support must be provided to help the user selectively extract relevant information from available information. Such support must help the user integrate, organise, compare, distil, summarise and apply the information. The development of the windows based environment is an important step in achieving the goal. The concept of windowing grew out of the principle that computers should support the way that people really work. (Funke, Neal and Paul, 1993, p.951)

It was observed that people seldom completed one task in a continuous time frame. Instead, they switched from application to application in response to events happening inside and outside the computing environment. In another study, observations were made on the way people arranged papers on their desktop. It was observed that users frequently rearranged the materials to match changing priorities. Based on these studies the desktop metaphor and the concept of windowing were born. (Funke, Neal and Paul, 1993, p.952)

The development and implementation of windowing have had a dramatic impact on the way people think about and use computers. A new dimension to the human computer interface has been made possible. It is now possible for example, to perform multiple tasks in parallel, and to view the results of one task while performing another. It is even possible to manipulate objects and parameters in one window and simultaneously view the results of that manipulation in another window. Also, information presentation can be managed in a manner that improves continuity of the dialogue and reduces disruption to important material on display. These advantages of a windowing environment have led to its rapid and wide-ranging acceptance. (Funke, Neal and Paul, 1993, p.952; Luders, Ernst and Stille, 1995, p.1183)

However there are costs with windowing environment as well. The user must now assume the burden of managing the windowing environment. Windows must be created and displayed, placed at desired locations, and moved to uncover needed information on other windows, re-sized, exposed and de-exposed, rearranged and 'put away' when no longer needed. These tasks are added to the user's existing application domain tasks. They do not contribute to user productivity. One can view this scenario as challenging a user who is not so confident or competent with multiple different computers.

Several studies have suggested that the burden of window management on user increase the overall time taken to complete a task. Earlier experiments also suggest that a large determinant in the time it takes to solve a problem using windowing system is the time spent manipulating the windows themselves. It was also found that tasks finished were error free but required more time compared to non-windowing system. It was observed that the additional time was spent on window management operations. It is reasonable to conclude that the advantages of windows may be overshadowed by the window management operations that users must perform (Funke, Neal and Paul, 1993, p.953; Luders, Ernst and Stille, 1995, p.1184).

Secondly, an important consideration in the design of windowing system is the choice of window layout approach. The overlapped approach to window layout avoids the problem of restriction on window size but it has its own shortcomings. Overlapped windows cover information on underlying windows, sometimes requiring rearrangement of windows to achieve the desired multi-window view. Also, in an overlapped windowing system, it is possible to accumulate too many windows, making it difficult for users to keep an organised view of what is being presented. This is a serious problem when some windows become completely obscured by overlapped windows (Funke, Neal and Paul, 1993, p.953; Luders, Ernst, 1995,p.1).

From the preceding discussion it is proposed that we need an automatic window positioning system which generates a layout of non-overlapping windows. The system should be interactive to accommodate creation of new windows by the user, closure of any window by the user or re-positioning of a window. It is acknowledged that one of the main problems in display layout generation is to show more details of a portion of layout without hiding the remainder of the layout (Misue, Eades, Lai and Sugiyama, 1995, p.195; Storey and Muller, 1995).

In this thesis, the goal is to have a window occupying more and more display area and other windows giving up their display area depending upon level of interaction of a user with open windows. For example, if a user is working with five windows simultaneously and majority of the user interaction is focused on a particular window then it should generate maximum number of requests to increase it's display area. Similarly, each of the remaining windows should generate requests to reduce their display area. This results in dynamic modification in size of each window. The dynamic modification in the sizes of windows may introduce overlaps to the display layout or create a situation where the display layout is no longer able to fit in the available display area. The final display layout should remove overlaps and encapsulate the layout in available display area.

The research will also investigate the relationship between preserving the mental map of the user and optimising the usage of display area (Misue, Eades, Lai and Sugiyama, 1995,p.186). Misue, Eades, Lai and Sugiyama (1995) have suggested that to preserve mental map of a diagram, the orthogonal ordering, clusters and topology of a diagram should be maintained in the transformed display layout.

The orthogonal ordering is preserved if the horizontal and vertical ordering of objects is maintained. Keeping windows close in the distorted view if they were close in the original view preserves clusters. The topology is preserved if the distorted view of the graph is a homeomorphisim of the original view. Other properties to be preserved include straightness of lines, orthogonality of lines parallel to x and y axis and relative sizes of nodes.

One can see that it is impossible to allocate more space to a portion of layout without distorting one or more of properties described above. One of the main problems in layout adjustment is to show more details of a portion of a layout without hiding the remainder of the layout (Storey and Muller, 1995; Misue, Eades, Lai and Sugiyama, 1995,p.195).

It should be apparent from the above discussion that preserving mental map and optimising display area usage are conflicting goals. In this research an attempt is made to strike a balance between the two by implementing and testing few unique ideas. The prime concern of this thesis is to remove the overlap from the window layout, compact the layout to optimise display area and study the resulting effects on mental-map preservation.

The main problem to be solved by this thesis is similar in nature to graph layout, Very Large Scale Integrated (VLSI) circuit layout, floor plan optimisation and similar problems. Several graph layout algorithms have been developed so far. Many of them translate the graph layout problem into an equivalent mechanical or thermal system or combination of several physical systems with some instability. Then the whole system is brought to equilibrium by applying combination of famous laws of physics or their variants. The stable state is then translated back to actual visual system.

The hypothesis is that it is possible to develop an interactive automatic windows layout manger which can arrange windows in such a way that they do not overlap, fit in the finite display area, optimise the usage of display area and preserve the mental map of the layout to a reasonable degree.

This thesis is organised as follows. Chapter-2 surveys some of the prominent display layout algorithms. Chapter-3 describes relevant VLSI layout algorithms. Chapter-4 formally states the main focus of the thesis. Chapter-5 proposes SPORDAC (Shadow Propagation for Overlap Removal and Display Area Compaction) technique to solve the problem tackled by the thesis. Chapter-6 documents the results and analyses the performance of the SPORDAC algorithm and the SPORDAC prototype. Chapter-7 highlights main research findings, strengths and weaknesses of the SPORDAC technique and concludes the thesis.

CHAPTER: 2

Display Layout Algorithms

This chapter discusses some of the prominent display layout algorithms developed over the years. Many algorithms translate display layout problems into a pseudo-physical system by replacing display objects with some form of a physical entity. Then different types of forces are assumed to apply to the system and the system is brought to equilibrium by applying one or more variations of famous laws of physics. The spring algorithm developed by Peter Eades is one of the first algorithms to implement such an approach (Eades, 1983,p.149).

Many variations have been developed on this theme. They either refine the algorithm or consider more variety of forces. The force scan algorithm and COMAIDE are examples of such a category (Misue, Eades, Lai and Sugiyama, 1995,p.190; Dodson, 1993).

Another group of algorithms implement a fish-eye view or a variation on the idea (Noik, 1993, p.336; Misue, Eades, Lai and Sugiyama, 1995,p.200). Over the years many algorithms have been developed on this theme. This chapter describes orthogonal and biform mapping approaches (Misue, Eades, Lai and Sugiyama, 1995,p.200).

SHriMP view technique is a simple method to preserve mental-map while ensuring encapsulation of windows in finite display area (Storey and Muller, 1995). This chapter concludes with discussion of Luders's automatic display layout algorithm (Luders, Ernst and Stille, 1995,p.1194). One can also refer to an excellent bibliography of display layout algorithms prepared by Battista, Eades, Tamassia and Tollis (1994) to investigate variety of algorithms on the subject. Peter Eades has also reported numerous free tree-drawing algorithms (Eades, 1991,p.1).

Spring algorithm

This algorithm attempts to generate a symmetric layout of a graph by applying some of the fundamental laws of physics. The main idea is to replace each node of the graph with a steel ring and each edge with a spring to form a mechanical system. Steel rings repel each other while springs pull the nodes together. Initial layout is generated randomly and the whole system is brought to equilibrium by positioning the nodes in such a way that opposite force cancel out each other. A conventional spring exerts force on a node according to Hooke's law ie., proportional to distance. It is observed in practice that such a spring exerts a large force on nodes that are far apart (Eades, 1984,p.150).

To remedy the situation, spring force is calculated according to following logarithmic formula:

$$F = C \times \log\left(\frac{d}{D}\right)$$

where $C =$ constant,
$d =$ length of the spring,
$D =$ constant

As we can see, force equals to zero if $d = D$.

Non adjacent nodes repel each other by the force calculated using following formula:

$$F = \frac{R}{\sqrt{d}}$$

where $R =$ constant,
$d =$ distance between nodes.

The following algorithm simulates the mechanical system (Eades, 1984, p.150).

```
1.0  Generate random layout
2.0  Repeat M times
3.0      Calculate force on each node;
4.0      Move the node by $k \times (force\_on\_node)$;
5.0  end repeat
6.0  Draw graph
```

where k is a constant.

This algorithm is successful in generating good layouts with less than 30 nodes. It is acknowledged that for dense graphs quality of layout is inferior (Eades, 1984,p.151).

There are practical difficulties in employing this algorithm for this thesis as outlined below.

- This algorithm ensures a spring of non-zero, positive length but it does not seem to handle nodes of different sizes. Hence when the size of a node dynamically changes, the nodes might overlap even though they are connected by a spring.
- It follows from the outline of the spring algorithm that it may generate a layout that does not fit into the available display area.

Co-Operative Multilayer Application-Independent Diagram Environment (COMAIDE)

Dodson (1993) has suggested a method for graph layout in 3D utilising laws of physics. In this approach, each node has mass M and obeys Newton's second law of motion. Each link or edge of the graph behaves as a spring and has negligible mass. It is assumed that whole system is immersed in a viscous liquid. Thus the display layout problem is translated to its equivalent thermo-mechanical system. The algorithm then attempts to bring this system to equilibrium. The problem is solved when the system attains equilibrium.

Dodson (1993) has defined following forces for COMAIDE.

The force exerted by a node is given by,

$$F = m\dot{V}$$

where $\dot{V} = \frac{dV}{dt}$
$V =$ velocity of the node
$m =$ mass of the node.

The force F is sum of motive force on the node and the drag because of viscosity. If we neglect the size of the node then we can say that,

$$m\dot{V} = F_{MOTIVE} - kV$$

where $k =$ viscosity coefficient
$V =$ velocity of the node.

Generally it is assumed that nodes have negligible mass (Dodson, 1993). Which suggests that,

$$V = \frac{F_{MOTIVE}}{k}.$$

The COMAIDE prototype discussed by Dodson (1993) calculates the final layout by arranging display objects in layers. The layout algorithm assumes the existence of variety of 'force-links' between display objects, layers and links themselves. Interested reader can refer to Dodson (1993) for further details on the topic.

The main algorithm of the system (Dodson, 1993) is as follows:

```
1.0   For each node in $n$: Set $V(n)$ {the velocity of $n$} to $[0,0,0]$;
2.0   While $T_s > 0$ { where $T_s$ is the assumed size of time-step in
      seconds };

<<< Force Computation: >>>

3.0       For each node $n$: Set $F(n)$ {the motive force on $n$} to
          $[0,0,0]$;
4.0       For each node $n$: Add its boundary repulsion forces to $F(n)$,
          also,
5.0       For each force couple between two diagram elements:
6.0           For each member $E$ of the pair of elements:
7.0               If $E$ is a node $n$: Add the relevant force to $F(n)$
8.0               Else ($E$ is a link from node $n_1$ to node $n_2$:)
9.0               Add the relevant forces to $F(n_1)$ and $F(n_2)$;

<<< MOTION Computation >>>

10.0      If inertia > 0 then:
11.0      $T_{ss} = \frac{T_s}{5}$;

12.0          For each node $n$, repeat 5 times
13.0              $Posn(n) = Posn(n) + V(n) * T_{ss}$
14.0              $V(n) = V(n) + \left( \frac{(F(n) - V(n) * Viscosity) * T_{ss}}{Mass(n) * inertia} \right)$
15.0      Else For each node $n$:
                  $V(n) = \frac{F(n)}{Viscosity}$;
                  $Posn(n) = posn(n) + V(n) * T_s$;
                  $Viscosity = \min(end\_viscosity, viscosity * \left( \frac{100 + anneal\_rate}{100} \right)$;
```

The above stated algorithm solves the layout problem by assuming that the layout was suspended in a thick fluid, which continually extracts energy from the layout and promotes low energy stable state. The motion and heat that are induced in actual environment are ignored. Notion of inertia in the system controls the effect of mass in the system.

Some of the drawbacks of the algorithm that discourage us from utilising it to solve our problem are listed below.

- It appears from the algorithm proposed by Dodson (1993) that COMAIDE algorithm does not deal with the problem of encapsulation of nodes in a finite area or volume.
- Dodson (1993) acknowledges that the size of a node is ignored in analysis. This may not be a valid assumption for this thesis.
- This algorithm partially relies on user intervention for generating layout (Dodson, 1993). The objective of this thesis is to find a solution where user is relieved from this time consuming unproductive chore.

Simple Hierarchical Multi-Perspective (SHriMP) View method

Preserving orthogonal order of the diagram and fitting the layout in finite display area are the main goals of this algorithm. According to this algorithm other nodes give up their display area to allow a node of interest to grow in size.

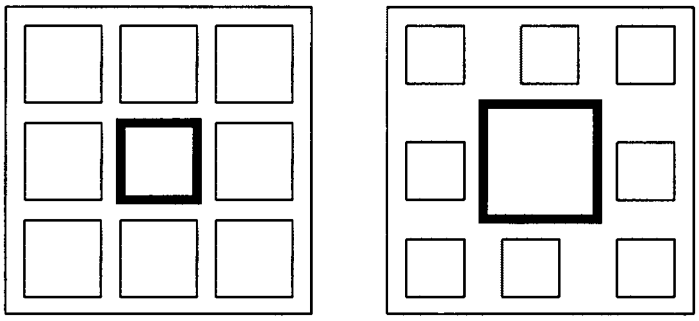


Figure 2.1
Final layout calculated by SHriMP view technique after central node is expanded.

As shown in above diagrams, a node grows by pushing other nodes outwards assuming infinite display space. The nodes are then scaled around the centre point of the display area to fit the available space.

The nodes are pushed outwards by adding a translation vector $[Tx, Ty]$ to its coordinates. Then the nodes are scaled around an arbitrary fixed point. Scale factor is decided by dividing required size of the screen to the requested size of the screen.

The following equations are applied to a node coordinate (X,Y) to translate it to the new position (X',Y').

$$X' = Xp + s*(X + Tx - Xp)$$

$$Y' = Yp + s*(Y + Ty - Yp)$$

Where,

(Xp,Yp) =	coordinates of fixed point
(X,Y) =	coordinates of a node
(X',Y') =	new coordinates of a node
$[Tx,Ty]$ =	translation vector
s =	scaling factor

The magnitude and direction of the translation vector T decides the new positions of nodes when they are pushed (Storey and Muller, 1995).

Three variants on above theme are designed by modifying the translation vector T.

<u>SHriMP view (variant 1)</u>

In this method, the graph is partitioned into nine different sections by extending the edges of the scaled node. The translation vector for each node is calculated according to the partition containing its centre.

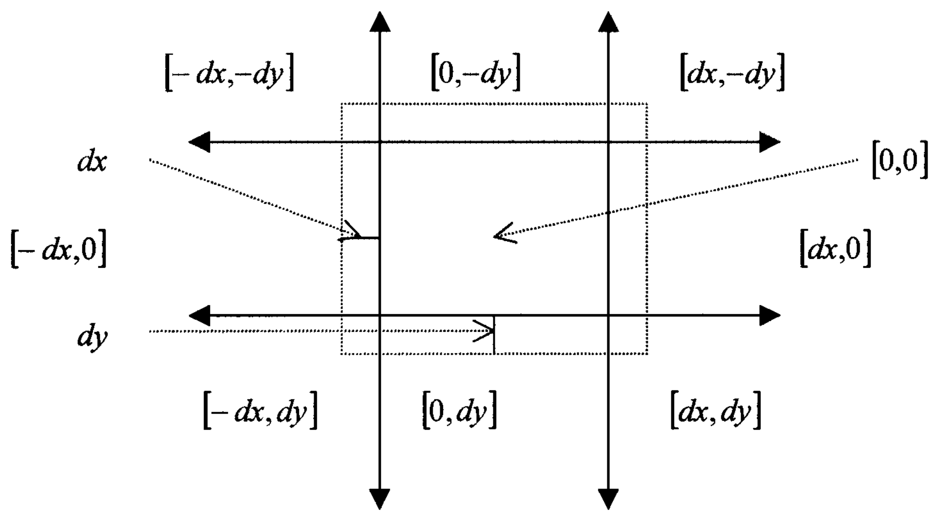


Figure 2.2
Translation vectors for each sibling node is determined by the partition containing its centre.

In the above diagram, *dx* and *dy* are *x* and *y* direction differences between the new size of the scaled node and its previous size. The dotted square represents new size of an expanded node. As we can see, the layout is divided in nine partitions with corresponding translation vectors. All the nodes are pushed by the same amount in both directions to maintain orthogonal relationships (Storey and Muller, 1995).

SHriMP view (variant 2)

In the second approach each node stays on the line connecting its centre to that of the node being re-sized. When a node is re-sized, it pushes a sibling node outward along this line. This technique preserves proximity relationships. The direction of each sibling node's translation vector is equal to the direction of the line connecting the centres. The magnitude of this vector is equal to the distance that a corner point of the scaled node moves as it is enlarged.

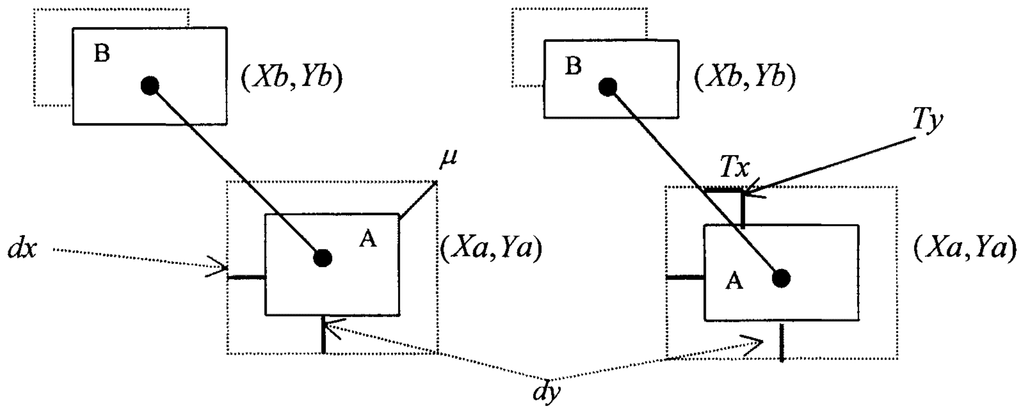


Figure 2.3(a) Figure 2.3(b)

Fig 2.3(a) A sibling node, B , is pushed outward along the line connecting its centre and the centre of A, the node being scaled. Each node is pushed out by distance μ.

Fig 2.3(b) A sibling node, B, is pushed along the vector between its centre and that of A, the node being scaled. The distance it is pushed along this vector is determined by the displacement of the intersecting node's edge as it moved along the vector.

The equations used in this approach are as shown below.

$$\mu = \sqrt{dx^2 + dy^2}$$

$$Tx = \mu \frac{Xa - Xb}{\sqrt{(Xa - Xb)^2 + (Ya - Yb)^2}}.$$

The second variant sacrifices orthogonal relationships to some extent but nodes, which were close in the original view, remain close in the transformed view as well (Storey and Muller, 1995).

<u>SHriMP view (variant 3)</u>

In the third variant, the direction of the translation vector remains the same but the magnitude is not the same for all sibling nodes. A node pushes out sibling nodes according to the displacement of the scaled node's edge as it moved along the line connecting their centres.

$$\begin{aligned}
&Tx = \frac{1}{m}(Yb \pm dy - Ya) + Xa - Xb && \text{if } |m| \ge 1 \\
&Tx = 0 && \text{if } |m| = 0 \\
&Tx = \pm dx && \text{otherwise} \\
&Ty = m(Xb \pm dx - Xa) + Ya - Yb && \text{if } 0 < |m| < 1 \\
&Ty = 0 && \text{if } |m| = 0 \\
&Ty = \pm dy && \text{otherwise}
\end{aligned}$$

Where $(Xa, Ya) =$ coordinates of the node being expanded,
$m =$ slope of the line connecting centres of expanding and pushed nodes.

The strengths and weaknesses of SHriMP view method are as mentioned below:

- Storey and Muller (1995) claim SHriMP view technique to be fast in execution and easy to implement.
- It is also claimed that SHriMP view technique preserves orthogonal relationships and the proximity of nodes and fits the final layout within the display area (Storey and Muller, 1995).
- However it seems from the mathematical formulas used for different variants of SHriMP view technique that this method does not attempt to optimise usage of the display area or remove overlap.

Fish Eye View layout algorithms

The Fish Eye View (FEV) algorithm designed by Furnas (cited in Storey and Muller, 1995) aims to view and navigates detailed information while providing the user with important contextual cues. This display method is based on the fish eye lens metaphor where the objects in the centre of the view are magnified and the objects further from the centre are reduced in size. In Furnas' formulation, each point in the display structure is assigned a priority calculated using a degree of interest function. Objects with a priority below a certain threshold are filtered from the view (Storey and Muller, 1995).

Several variations on this theme have been developed to deemphasise the information of lesser interest by using the size, position, colour or shading along with filtering (Storey and Muller, 1995; Misue, Eades, Lai and Sugiyama, 1995,p.199; Noik, 1993, p.336). A method proposed by Sarkar and Brown (Cited in Storey and Muller, 1995) magnifies the objects of interest and demagnifies the objects of lower interest around focal point. Therefore nodes further away from the focal point look smaller (Storey and Muller, 1995).

An alternative is the continuous zoom algorithm designed by Ho et. al., (Cited by Storey and Muller, 1995) that allows the user to expand and shrink nodes while navigating a diagram.

In the orthogonal stretching algorithm suggested by Sarkar (Cited by Storey and Muller, 1995) the user stretches a square region of the display area in X and Y directions. The Objects within this region are stretched while the objects outside this region are contracted uniformly

Misue et. al., (1995) has described three variations on the FEV method. They are Biform Display method (BF), the Fish Eye display method (FE) and Orthogonal Fish Eye method (OFE).

Orthogonal Fish Eye method:

This method computes display layout along both axes independent of each other and preserves orthogonal ordering of the display objects. It also preserves straightness of lines parallel to X and Y axis. The following equations are used to move the point (X,Y) to (X',Y').

$$X' = \frac{1}{n}\sum_{i=0}^{n-1}\Phi_i$$

$$Y' = \frac{1}{n}\sum_{i=0}^{n-1}\psi_i$$

$$\Phi_i = \frac{r}{\Pi}\tan^{-1}\frac{x-p_i}{s_i}$$

$$\psi_i = \frac{r}{\Pi}\tan^{-1}\frac{y-q_i}{s_i} \qquad (i = 0,1\ldots\ldots,n-1)$$

$$\theta_i = \tan^{-1}\frac{y-q_i}{x-p_i}$$

$$l_i = \frac{2r}{\Pi}\tan^{-1}\frac{\sqrt{(x-p_i)^2+(y-q_i)^2}}{s_i}$$

where (p_i,q_i)= view point
Φ = polar angle from view point
l_i = new distance of point(X,Y) from a view point(p_i,q_i)
s_i = constant to control magnification ratio at view point(p_i,q_i)

The coefficient $\frac{r}{\Pi}$ ensures that the objects stay within a square of side r. This method can theoretically display an infinite domain on a finite area but in practice they tend to crush surrounding areas infinitely and make them invisible (Misue, Eades, Lai and Sugiyama, 1995, p.201).

Biform Mapping:

This method claims to overcome shortcomings of previous method by using 'view areas' instead of viewpoints. This method preserves the aspect ratio of the rectangular frames of display objects. The display objects are magnified uniformly in each view area and demagnified uniformly outside the view areas (Misue, Eades, Lai and Sugiyama, 1995, p.201).

Force Scan Algorithm

The Force Scan Algorithm uses the principle similar to the spring algorithm to move nodes in both horizontal and vertical directions to avoid overlaps. The main idea is to apply force F_{uv} between two pairs of u,v nodes so that overlap of node u and v can be removed. The force is applied in both directions (Misue, Eades, Lai and Sugiyama, 1995, p.191).

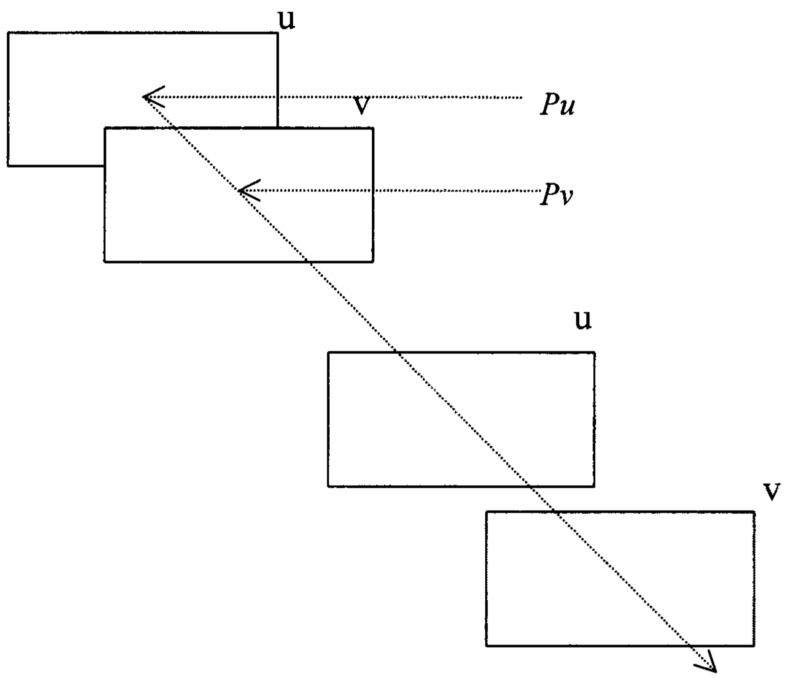


Figure 2.4 Actual and desirable distances between two windows.

The force F_{uv} is applied along the line connecting the centres of nodes. The magnitude of the force is the difference between the actual distance D_{uv} and the desirable distance K_{uv} between the node images for u and v. The force is analogous to Hooke's law (Misue, Eades, Lai and Sugiyama, 1995, p.191).

The actual distance D_{uv} is the Euclidian distance between P_u and P_v. The desirable distance K_{uv} is the distance required between centres of both nodes to remove overlap. Let $r = (X, Y)$ be the first point along the line from P_u to P_v for which either $|X - Xu| \geq \frac{(Wu + Wv)}{2}$ or $|Y - Yv| \geq \frac{(Hu + Hv)}{2}$.

Then the desirable distance K_{uv} is the Euclidean distance between P_u and r (Misue, Eades, Lai and Sugiyama, 1995, p.191).

If the nodes u and v overlap, then the magnitude of the force F_{uv} is $K_{uv} - D_{uv}$ (Misue, Eades, Lai and Sugiyama, 1995, p.191). Thus,

$$F_{uv} = \max(0, K_{uv} - D_{uv}) \times u$$

Where u = unit vector in the direction from P_u to P_v

In practice a constant value g can be added to K_{uv} to force a gap of size g between nodes.

The Force Scan Algorithm applies forces in two scans. The first is in the horizontal direction preserving horizontal order of nodes. The second scan is in the vertical direction and preserves the vertical order of nodes. Nodes are sorted in ascending order of x coordinates of their centre (Misue, Eades, Lai and Sugiyama, 1995, p.191).

Misue, Eades, Lai and Sugiyama (1995) outline the horizontal scan algorithm as follows.

```
1.0   $i = 1;$
2.0   While $i < |V|$ do
3.0       Suppose that $x_i = x_{i+1} = \cdots\cdots = x_{k+1};$
4.0       $\delta \leftarrow \max_{i \le m \le k < j \le |V|} f^{x}{}_{v_m v_j};$
5.0       for $j \leftarrow k+1$ to $|V|$ do  $x_{v_j} \leftarrow x_{v_j} + \delta;$
6.0       $i = k + 1;$
```

A similar scan is applied in opposite direction.

A variation of force scan algorithm called push - pull algorithm uses following force equation (Misue, Eades, Lai and Sugiyama, 1995, p.192).

$$Fuv = (Ku - Duv) * U$$

With this modification the force Fuv will be positive if the desirable distance is more than actual distance and it will be negative when nodes overlap. The positive value of the force pulls the diagram together and negative value removes overlap (Misue, Eades, Lai and Sugiyama, 1995, p.192).

However, this algorithm does not always fit the diagram in finite display area and it is acknowledged that it may not always produce non-overlapping layout (Misue, Eades, Lai and Sugiyama, 1995, p.195).

Luders's Automatic Display Layout method

Luders's approach considers display layout generation of hierarchical objects as a combinatorial optimisation problem. This approach achieves a final display layout in two phases.

In the first phase, all display objects are assumed to be of the same size and a grid based display layout is generated. The grid locations are calculated on the basis of dimensions of display area, and the number of objects that can be displayed side by side and vertically tiled. After the grid is generated, the size of display objects is modified in such a way that they do not overlap. A simulated annealing algorithm is used to generate the grid based layout. The cost function for simulated annealing algorithm is designed to accommodate hierarchical nature of the objects. The cost function tends to optimise the number of edge crossings, length of edges, edges running across objects, etc (Luders, Ernst and Stille, 1995, p.1189).

In the second phase, a modified version of force directed algorithm is applied in two parts:

(a) a force-directed part, which is responsible for minor changes of the placements of objects, and

(b) a pressure directed part, which actually introduces different object sizes etc (Luders, Ernst and Stille, 1995, p.1193).

Force-directed part:

In force-directed part it is assumed that certain static spring forces and dynamic forces are applied to the nodes. Applicable static spring forces are defined as follows:

(a) the attracting force $F_{stat,E}$ between objects which are connected by edges

(b) the repulsive force $F_{stat,O}$ between each pair of objects, and

(c) the repulsive 'border force' $F_{stat,B}$ between each object and the borders of the placement area (in each direction $+x,-x,+y,-y$) etc (Luders, Ernst and Stille, 1995, p.1193).

The magnitudes of these forces are dependent on the distance between the objects and on the distance between an object and the border of the placement area etc (Luders, Ernst and Stille, 1995, p.1189).

The calculation of the repulsive forces is done in such a way that the force tends towards infinity with decreasing distance. Therefore, an overlapping of objects or the placement of an object outside the placement area is impossible. However, overlap may occur if the size of the rectangle representing the object changes (Luders, Ernst and Stille, 1995, p.1194).

The following dynamic forces (Luders, Ernst and Stille, 1995, p.1194) are defined for the system:

(a) an attracting force $F_{dyn,RAbs}$ between the current and the reference placement of an object, and

(b) the force $F_{dyn,R\mathrm{Re}l}$, which pushes an object in order to keep the relative position of two objects in two succeeding layouts.

The forces effective on an object i are calculated by determining linear combination of all forces effective on i. The final location of each object i is determined from these forces.

The pressure directed part:

This part modifies sizes of objects in order to introduce objects of different sizes. To achieve this, *inner* and *outer* pressures of an object are introduced. The sizes of objects are modified observing Boyle and

Mariotte's law which says that product of volume V and pressure P of a gas is constant: $P \times V = const$ (Luders, Ernst and Stille, 1995, p.1196). Luders, Ernst and Stille (1995) describe their algorithm for pressure directed part as outlined below.

```
1.1   Determine initial placement
1.2   Determine forces effective on each node
1.3   Determine $\Delta P = |P_{outer} - P_{inner}|$ for each node
1.4   Repeat
1.5         for $i = 0; i < I_D; i++$ begin
1.6             Determine node $j$ with maximal $\Delta P$
1.7              If $P_{innerj} - P_{outerj} > 0$ then
1.8                     Reduce $j$ in size
1.9              else
1.10                    enlarge node $j$
1.11           Modify pressure differences $\Delta P$
1.12        end
1.13        for $i = 0; i < I_F; i++$ begin
1.14                    Determine node $j'$ with maximal $F_j$
1.15                    Move $i$ in the direction of $F_j$
1.16                    Modify forces $F$
1.17        end
1.18  until stop criterion
```

This algorithm terminates if the maximum number of iterations is reached or if the improvement of the pressure or force differences drops below a lower limit (Luders, Ernst and Stille, 1995, p.1198).

It is claimed that this method effectively generates 'good' layouts for windows containing different types of data and hierarchical display objects (Luders, Ernst and Stille, 1995, p.1198).

However, this method is designed for non-interactive mode and may generate an overlapped layout if display object size is modified in force directed layout generation.

One of the main objectives of this thesis is to develop an interactive display layout algorithm, which can generate non-overlapped layout under all conditions in dynamic environment.

This chapter has surveyed some of the prominent display layout algorithms and evaluated their strengths and weaknesses. The next chapter discusses how the VLSI layout problem relates to the display layout problem and investigates some of the relevant VLSI layout algorithms.

CHAPTER : 3

VLSI Layout Algorithms

This chapter begins with comparison of the display layout and the VLSI layout problems followed by discussion of some of the relevant VLSI layout algorithms. The chapter describes few VLSI layout generation methods, which operate on 'constraint' graph generation principle. We will discuss Horizontal Shuffle, Line Sweeping Algorithm, Enhanced Plane Sweep method, Shift Compaction method and Shape Optimising method in this chapter. These methods generate a 'constraint' graph to represent objects and constraints to be observed. The constraint graph is then manipulated and final solution layout is computed.

Display layout and VLSI layout problems

There are several similarities between the problem of windows layout generation and that of VLSI layout. In both cases rectangular shapes need to be arranged in finite area and uphold certain constraints. It is the prime objective of this thesis to remove overlap and arrange the layout in such a way that display area is optimally utilised. Similarly, overlap removal and optimised usage of chip area could be one of the objectives of VLSI layout generation algorithm. However, there are some significant differences between the problems.

For VLSI layout generation it is generally assumed that enclosing area is big enough to accommodate all the objects and objects do not change their size. For windowing, the size of each window is dynamic and it keeps changing depending upon the user's interaction with the window. Hence the finite display area may not be enough to accommodate all windows in all situations. This would require scaling of the windows as and when necessary.

A second difference between VLSI and display layout problems is that the constraints to be applied in both cases may not be identical. The constraints to be observed would depend upon application specific details. This would determine whether some algorithmic operations could be allowed in the system or not. For example, changing the orientation of a cell could be a very useful operation for VLSI layout generation but may not be recommended for windows layout generation as it may generate a layout that becomes difficult to comprehend by a user. Similarly, scaling is a legitimate operation for window layout generation but can not be applied for VLSI layout generation, etc.

Thirdly, VLSI layout generation process does not involve any user interaction, as the final output is not visual in any way. But in a window environment, user interaction might change the number of windows open and hence we need an interactive algorithm.

Several VLSI layout algorithms have been designed over the years, which utilise graph theory principles to solve the problem. Many such algorithms construct a 'constraint graph' and 'solve' it to generate the final layout. A constraint graph represents objects in the layout as nodes and physical constraints amongst the objects as edges.

Horizontal Shuffle

In this algorithm, a node in the graph represents every cell or window. A hypothetical source and sink node is added to the diagram. The layout is compacted in the direction from source to sink node, ie., left to right or top to bottom (Lai,1993,p.100).

An edge between a pair of nodes is added if they overlap. The weight of the edge is the minimum distance required between x coordinates of centres of both nodes (Lai, 1993,p.101).

Every node is also connected to the source and sink nodes with weight equal to sum of half of the width of the node and gap (Lai, 1993,p.101).

An algorithm to compute maximum weight path is used to determine final position of each node in horizontal direction. This generates a non-overlapping layout. This algorithm removes overlap but the constraint graph may contain redundant edges (Lai, 1993,p.102).

The same process can be applied in the opposite direction to compact the diagram in both directions.

Line Sweeping Algorithm

The Line Sweeping Algorithm described by Hsiao and Feng (1990) generates a constraint graph as described below.

When considering X compaction, a vertical line scans the diagram from left to right. The sweeping line jumps from left to right of the source layout step by step. At each step, it should stop whenever the sweeping line exactly encounters one or more edges of the layout rectangles. Moreover, every edge of a layout rectangle must be encountered during the sweeping operation (Hsiao and Feng, 1990, p.78).

At each sweeping step, the sweeping line intersects with a set of rectangles, which are then called active rectangles. Some of these rectangles' left or right edge may coincide with the sweeping line. These rectangles are dealt one by one in the graph generation algorithm. Whenever any of these rectangles is the one currently being considered, it is called the master rectangle. Each time after all right and left edges of encountered rectangles have been processed, the sweeping line automatically jumps to the next step using a special algorithm (Hsiao and Feng, 1990, p.79).

Hsiao and Feng (1990) describe Line Sweeping Algorithm as follows.

```
1.0  {
2.0          Generate an empty adjacency list and several
             temporary buffers;
3.0          Start to scan the source layout from left to right;
4.0          While (at each sweeping step)
5.0          {
6.0                  Choose the master rectangle from current
                     active rectangles one by one;
7.0                  For (each master rectangle)
8.0                  {
9.0                          Refer to the buffers to avoid trivial
                             considerations of the constraints that
                             have been dealt with before;
10.0                         Solve constraints;
11.0                         Rearrange the unnecessary constraints
                             generated in the previous sweeping steps
                             from the buffers;
12.0                 }
13.0                 Update the generated adjacency list of the
                     constraint graph;
14.0         }
15.0         Compact the adjacency list;
16.0 }
```

Optimal constraint graph generation using Enhanced Plane Sweep method

The Enhanced Plane Sweep method is an improved constraint graph generation method. A constraint graph contains objects as its nodes and directed edges as constraints applicable to the layout.

It is often found that most of these constraints are redundant and result in deterioration of performance. This algorithm aims to generate required constraints only (Awashima, Sato and Ohtsuki, 1993, p.507).

Y compaction using this algorithm can be explained as follows.

This algorithm aims to generate a constraint graph $G = (V, E)$, where each vertex Vi represents a corresponding layout object and each directed edge $e_{ij} \in E$ represents any existing constraint between vertices V_i and V_j. Since we are discussing Y compaction, the direction of separation edges is upward, that is, from lower objects to upper objects. Each separation edge is weighted according to a minimum spacing rule between two vertices connected by the edge.

An edge e is said to be redundant if and only if at least one directed path other than e exists that connects two vertices connected by e. In other words, a redundant edge e is a shortcut of a constraint graph (Awashima, Sato and Ohtsuki, 1993, p.508).

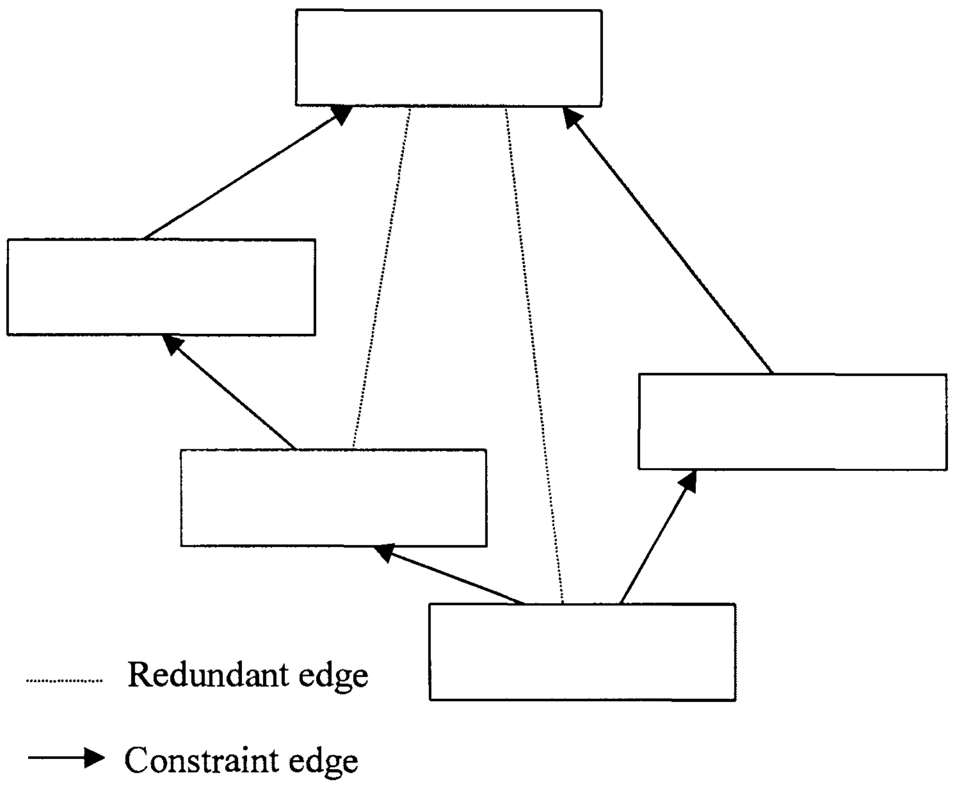


Figure 3.1 An example of constraints among objects

The algorithm maintains a list called PB (Previous Boundary). Here the direction of plane sweep is vertical which means that the horizontal scan line sweeps layout objects from top to bottom. PB is the list of horizontal boundary edges of previously swept objects that are vertically visible from current scan line. During plane sweep, a CSL (Current Scan Line) buffer is maintained to store objects crossing the current scan line generation (Awashima, Sato and Ohtsuki, 1993, p.509). Horizontally adjacent objects can be detected by searching this buffer. Vertically adjacent objects can b detected by searching PB. CSL is not necessary during vertical constraint graph generation (Awashima, Sato and Ohtsuki, 1993, p.509).

PB can be thought of as a conjunction of shadow fronts propagated from objects currently on the scan line and from objects that will be swept afterwards. Candidate objects for generating constraints can be detected by searching PB within a range that is derived from the size of a same object on the scan line. After enumerating candidate objects that are visible from a source object, a redundancy check is done in a simple way, so that redundant constraints are neglected and never generated (Awashima, Sato and Ohtsuki, 1993, p.508).

The input to the algorithm is a list of n objects stored in the Event List (EL) and a value (D) specifying the minimum distance between objects. The algorithm operates on the inputs to produce an optimal constraint graph. Awashima, Sato and Ohtsuki (1993) explain enhanced plane sweep algorithm as outlined below.

```
1.0   Sort EL in descending order of Y coordinate of upper edges of
      objects. Set the value for $D$. Initialise a work list PB.
2.0   If EL is empty then stop. Otherwise read next object S from EL.
3.0   Search PB within the range determined by extending upper edge $h$
      of current object S with minimum spacing value D in both left and
      right directions and enumerate candidate objects.
4.0   For each candidate object, one of updating operations of PB is
      applied. Constraint is generated if necessary.
5.0   Insert current object S into PB according to X coordinate of left
      end point of upper edge $h$. Go to step 1.0.
```

The update operation on PB is followed by a simple redundancy check. Then a constraint is added if it is not redundant. Redundancy checks are performed as described below. Here a candidate edge in PB is denoted by g and source edge on current scan line by h. The following redundancy checks stand on the fact that a constraint from h to g is redundant if and only if at least one object exists between h and g (Awashima, Sato and Ohtsuki, 1993, p.509).

(a) g covers the left end point of h

If the right end point of g is the upper right corner of the corresponding object then a constraint from h to g is required. Update x coordinate of the right end point of g by x coordinate of the left end point of h (Awashima, Sato and Ohtsuki, 1993, p.509).

(b) g covers the right end point of h

If the left end point of g is the upper left corner of the corresponding object then a constraint from h to g is required. Update X coordinate of the left end point of g by x coordinate of the right end point of h (Awashima, Sato and Ohtsuki, 1993, p.508).

(c) h covers g

If the left end point of g is the upper left corner of the corresponding object and the right end point of g is the upper right corner of the corresponding object then a constraint from h to g is required. Delete g from PB (Awashima, Sato and Ohtsuki, 1993, p.508).

(d) g covers h

A constraint from h to g is always required. Duplicate g into gl and gr. Set x coordinate of the right end point of gl by x coordinate of the left end point of h. Set x coordinate of the left end point of gr by x coordinate of the right end point of h (Awashima, Sato and Ohtsuki, 1993, p.508).

It is shown that time complexity of the algorithm is $O(n * \log n)$. This means that the amount of time spent by this algorithm to solve the problem is proportional to $n * \log n$. Here n represents the problem size. It is also claimed that traditional constraint graph generation algorithms have complexity of $O(n^2)$ hence this method is better (Awashima, Sato and Ohtsuki, 1993, p.510).

At the end of constraint graph generation stage, we get an overlap free layout compacted in one direction. We can repeat the operation in the opposite direction to compact the layout in both X and Y directions. Even after we compact the diagram in both directions, we may still not get the best solution. There are a few VLSI layout algorithms that operate on constraint graphs in both directions and attempt to compact the layout. These algorithms work in both directions simultaneously. Two such algorithms worth mentioning here are Shift compaction and zone refining method. Both methods are described below.

Shift Compaction Algorithm:

It is argued that two independent compactions in both directions do not necessarily give sufficient compaction in every situation. Therefore a better method is sought.

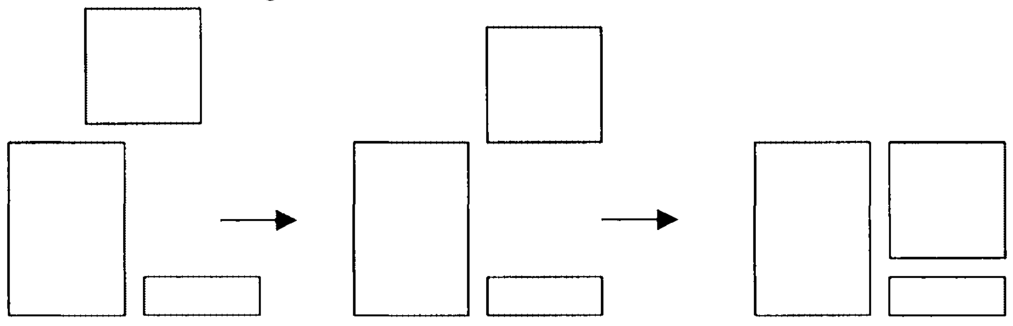


Figure 3.2(a) Figure 3.2(b) Figure 3.2(c)

Figure 3.2(a) A result of one dimensional compaction. (b) Shift operation (c) Result of Y compaction.

Let us consider the layout of blocks suggested in the figure. The usual one-dimensional compaction algorithms can not compact the layout. However, shifting both blocks A and B in X direction followed by a Y compaction operation produces a better quality layout as evident from the diagrams. The compaction method based on such an idea is described as follows (Sakamoto, Onodera and Tamaru, 1990, p.41).

(1) The layout compacted in one direction is taken as input.

(2) The critical paths are cut off by shifting those layout elements that lie on critical paths in the direction perpendicular to the compaction direction.

(3) Compact the layout again by using a one-dimensional compaction algorithm (Sakamoto, Onodera and Tamaru, 1990, p.41).

In this method, objects lying on the critical path determine the layout width. This method shifts the objects lying on the critical path in the non-compaction direction to reduce the length of critical path in compaction direction. The shift direction is perpendicular to compaction direction (Sakamoto, Onodera and Tamaru, 1990, p.41).

Sakamoto, Onodera and Tamaru (1990) explain their algorithm as outlined below.

Input: One dimensional compacted layout in Y direction .

Output: Compacted layout in Y direction.

```
1.0   {
2.0        Make $G_y$ and solve it
3.0        Make $G_x$ and solve it
4.0        Repeat {
5.0             Find the critical path in $G_y$
6.0             Cut the critical path in $G_y$ by shifts
7.0             If (possible) {end}
8.0             Determine X positions according to $G_x$ with shift
                constraint edges.
9.0             Make $G_y$ and solve it.
10.0       } until (The critical path length of $G_y$ is less than the current
           Y width)
11.0       Compact the layout in the y direction according to $G_y$.
12.0  }.
```

In above algorithm, G_x and G_y represent constraint graphs in X and Y directions. Solving a graph means locating the critical path in the graph.

Y shift compaction algorithm attempts to control the width of the layout. Similarly, it is possible to derive a X shift compaction algorithm to control the height of the layout.

However the authors acknowledge that an attempt to reduce width in one direction may result in increase in width in opposite direction. Because of this drawback, this algorithm is not utilised to solve the problem addressed by this research (Sakamoto, Onodera and Tamaru, 1990, p.41).

Shape Optimisation Algorithm

The main goal of this algorithm is to find the optimal shape and position of each layout element from the initial layout. The idea is to limit shape optimisation to only those objects that lie on the critical path. The shape of an element is then optimised to generate compact layout. This process is repeated until no further shape optimisation is possible (Okada, Onodera and Tamaru, 1995, p.170).

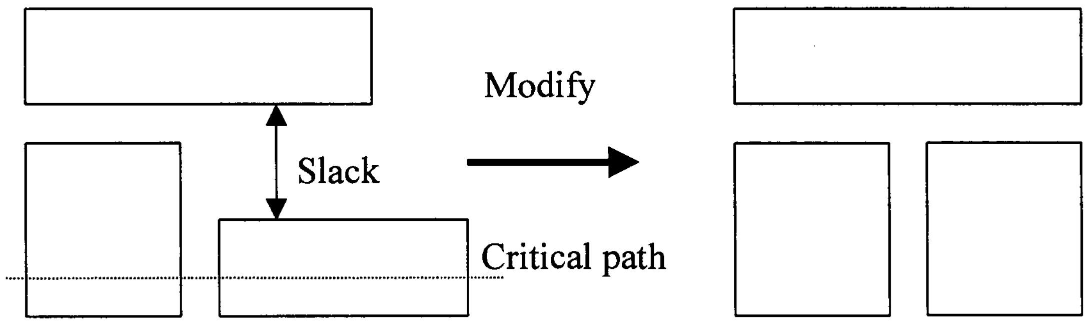


Figure 3.3 Conceptual illustration of shape optimisation.

In this algorithm, an element on the critical path is replaced with another element of smaller width to reduce the width of the critical path. The height of the whole layout remains unchanged if the increase in height of the element is less than the amount of Y slack (Okada, Onodera and Tamaru, 1995, p.5).

Okada, Onodera and Tamaru (1995) outline their shape optimisation algorithm as shown below.

```
1.0  {
2.0       Make X graph and solve it
3.0       Repeat
4.0       {
5.0            Make Y graph
6.0            Select element
7.0            Modify element
8.0            Solve Y graph
9.0            Make X graph and solve it
10.0      }until (No more possibility)
11.0 }
```

This algorithm selects an element in the same way it is selected in shift compaction algorithm. The selected element is modified in such a way that length of the critical path decreases without increasing the length of the critical path in the opposite direction.

This algorithm eliminates the drawback of the shift compaction algorithm but there are some problems if this VLSI layout algorithm is used for dynamic windows' layout.

As mentioned earlier in Chapter-1, the SPORDAC algorithm calculates the size of each window depending upon the level of user's interaction with each window. The Shape Optimisation Algorithm modifies the height and the width of objects to optimise the layout area. If this algorithm is utilised for windows layout generation, it could result in dramatic changes in aspect ratios of the windows. This may make it difficult for the user to understand new layout.

It is important to clarify that, in a dynamic environment like windows, at some stage windows may not fit in the finite display area. Hence the SPORDAC prototype should apply uniform scaling to all open windows without sacrificing their individual aspect ratios.

This chapter completes the survey of display layout and VLSI layout algorithms and methodologies. The next chapter states research questions and main research focus for this thesis and discusses some of the algorithmic aspects.

CHAPTER : 4

The Problem

Prominent Research Questions

We can restate our research hypothesis as described below.

The main objective of this thesis is to develop an interactive automatic windows display layout manager to relieve the user from unproductive chore of manual window management. The user can open as many windows as he or she may wish. Depending upon the level of interaction with each window we would like to automatically re-size and reposition all open windows in such a manner that they do not overlap, optimally utilise finite display area and preserve the user mental-map to fair degree. Our main focus would be on removing the overlap, encapsulating all open windows in display area and optimising the usage of display area.

It is clear that the problem addressed by this thesis is similar to floor plan area optimisation, VLSI layout generation, a special case of graph layout generation (Battista, Eades, Tamassia & Tollis, 1994, p.7) or any layout generation problem where rectangular objects need to be arranged in pre-defined space. The algorithm to be proposed by this thesis should be capable of handling dynamic enclosing area.

One can find an ample amount of literature that addresses the problem of facility layout planning (Suzuki, Fuchino, Muraki and Hayakawa, 1990,p.226; Proth and Souilah, 1992, p.227; Shih, Enkawa and Itoh, 1992, p.2839; Yaman, Gethin and Clarke, 1993, p.413; Bozer, Meller and Erlebacher, 1994, p.918; Bland and Dawson, 1994, p.500; Rebaudengo and Reorda, 1996, p.943).

Discussion about NP completeness

Pfleeger (1989,p.79) has explained that there are some problems that could be solved within the time bounded by a polynomial function of the size of the problem. For example, determining whether an item exists in a list or not can be done in time proportional to the size of the list. These types of problems are known to belong to class P.

By contrast, there are some problems that could be solved within the time bounded by a polynomial function of the size of the problem provided the problem-solving algorithm has the ability to '*guess*' the solution. This guessing is known as non-determinism (Pfleeger, 1989, p.79).

There are problems known to be solvable deterministically in polynomial time (P) and there are problems known not to have a polynomial time solution (*EXP*). The class *NP* complete fits somewhere between *P* and *EXP* (Pfleeger, 1989, p.79).

It is acknowledged that graph layout generation is a NP complete problem depending upon the aesthetic criteria to be satisfied (Eades, 1984,p.149). One of the recent researchers has recognised that window layout generation is a combinatorial optimisation problem (Luders, Ernst and Stille, 1995, p.1183).

In this thesis the question of whether the problem is NP complete or not is not considered. Interested researcher can refer to Joy and Smith (1995) for further information on the topic.

Tsuchida (1995, p.907) discusses the complexities involved with graph drawing and concludes that graph drawing is a NP complete problem under certain conditions.

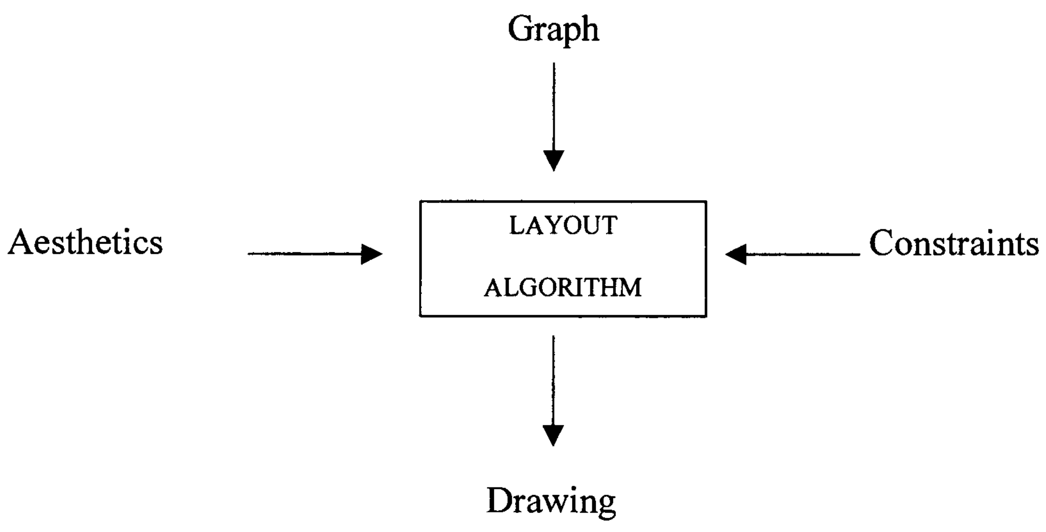


Figure 4.1 Overview of display layout process.

The above figure shows a very general idea of display layout process. A display layout algorithm operates on description of window layout along with some predefined constraints and aesthetics and produces final layout of windows.

This thesis assumes that the display layout problem is NP complete. In recent years, nature based optimisation methods have been popular in attempts at solving NP complete problems. Simulated annealing and genetic algorithms are such methods. Many of research papers cited in this chapter utilise one or both of these methods.

This thesis proposes a unique shadow propagation technique, SPORDAC (Shadow Propagation for Overlap Removal and Display Area

Compaction), to remove the overlap and compact the layout at the same time. This thesis utilises the SPORDAC algorithm, simulated annealing and genetic algorithm to generate solution display layout. The following chapter explains the design and implementation of the SPORDAC algorithm and the SPORDAC prototype to generate solution display layout.

CHAPTER:5

The Solution

The Solution

This chapter begins with a general outline of the solution proposed by this thesis followed by an explanation of the Annealing Genetic (AG) approach proposed by Lin, Kao and Hsu (1993) and its relevance to the development of the SPORDAC prototype. This is followed by the explanation of SPORDAC algorithm and how it is integrated with the AG approach. The chapter concludes with the description of the object model of the SPORDAC prototype.

Overview of the solution:

As it is stated earlier in Chapter-1 and Chapter-4, the SPORDAC prototype should be capable of handling dynamic changes in sizes of windows and display area. It should also calculate the size of each window depending upon the level of user interaction with each window. This results in dynamic window sizes. The underlying display area that contains every window could be dynamic as well. The SPORDAC prototype assumes a single parent window upon which user creates and manipulates numerous child windows. The SPORDAC prototype recognises following user interactions:

- The user is allowed to change the size of the parent window and add or remove a window at any point in time. Hence the number of windows open in the display area may change dynamically.
- The user should be able to create windows at will and position them at will. This means that the user could introduce overlaps at any point in time. Hence the SPORDAC prototype should provide a mechanism for the user to generate overlapping windows and the SPORDAC algorithm should be able to remove the overlap.

It is stated earlier in Chapter-1 that every interaction with a window enables that window to generate a request to increase its area and the remaining windows generate requests to reduce their display area. An interaction with a window can be recognised as an occurrence of any specific event. The event could be a window receiving input focus or getting activated, etc. Which event to select for triggering the display layout process is an application dependent issue. The SPORDAC prototype has selected the double-click event of left mouse button for simplicity.

The SPORDAC prototype should remain in a neutral state while it is not computing the solution display layout and wait for user interaction. In this neutral state the user will be free to modify system parameters, sizes and number of windows, display area dimensions, or invoke the display layout procedure by double clicking on a window.

The SPORDAC algorithm proposed by this thesis is a one-dimensional method that removes overlap from the layout and compacts it at the same time. The SPORDAC prototype applies this method in both directions to get a compact layout in both directions. The compact layout is then passed on to the genetic algorithm and a simulated annealing based controlling procedure to optimise usage of the display area. The best solution found is then scaled and mapped on to the available display area.

The SPORDAC prototype is implemented in Microsoft Visual C++ V4.2 under Windows'95 environment. Interested readers can refer to Appendix for a brief discussion of Object-Orientation and Document/View architecture proposed by Microsoft.

Genetic Algorithm

Genetic algorithms (GA) are search algorithms based on the mechanics of natural selection and natural genetics. They combine survival of the fittest among string structures with a structured yet randomised information exchange to form a search algorithm with some of the innovative flair of human search (Lin, Kao and Hsu, 1993,p.1752).

GAs are simple yet powerful in their search for improvement. They are not limited by restrictive assumptions about the search space like the existence of derivatives, uni-modality and other matters.

GAs are different from calculus based, enumerative and random search methods as follows (Genetic Algorithms, n.d., p.3; Jain and Gea, 1996, p.12):

- GAs work with a coding of the parameter set not the parameters themselves.
- GAs search from a population of points, not a single point.
- GAs use objective function information, not derivatives or other auxiliary knowledge.
- GAs use probabilistic transition rules, not deterministic values.

A simple genetic algorithm is composed of three operators.

Reproduction

Reproduction is a process in which individual strings are copied according to their objective function values. We can think of the objective function as some measure of profit that we wish to maximise (Genetic Algorithms, n.d., p.5). Copying strings according to their fitness values means that strings with a higher value have a higher probability of contributing one or more offspring in the next generation. This operator is an artificial version of natural selection. The objective function value of each string determines which string will be selected. The reproduction operator may be implemented in algorithmic form in a number of ways. Implementing a biased coin or a roulette wheel could be one such method (Jain and Gea, 1996, p.12).

Crossover

After reproduction, simple crossover may proceed in two steps. First, members of newly reproduced string in the mating pool are mated at random. Second, each pair of strings undergoes crossing over as follows, an integer position *k* along the string is selected uniformly at random between 1 and the string length less one (ie. $[1, l-1]$), where l is the length of the bit string). Two new strings are created by swapping all characters between position $k+1$ and l inclusively. For example, let us assume $A = 011011$ and $B = 110010$.

If the mutation position is four then after a crossover operator is applied on this pair of strings, we get $A = 011010$ and $B = 110011$ (Jain and Gea, 1996,p.13).

- <u>Mutation</u>

Mutation is an important operator because, even though reproduction and crossover effectively search and recombine different likely solutions, occasionally they may become over zealous and lose some potentially useful genetic material. In artificial genetic systems, the mutation operator protects against such an irrecoverable loss. In simple genetic algorithms, the mutation operator is applied with small probability by selecting one of the strings and randomly modifying one of its locations.

Mutation rates are comparatively small in natural systems. We can consider mutation as secondary mechanism of genetic algorithm adaptation (Jain and Gea, 1996,p.12).

A general outline of a simple genetic algorithm can be given as follows (Lin, Kao and Hsu, 1993,p.1755).

```
1.0   Initialise the parameters of the genetic algorithm;
2.0   Randomly generate the $old\_population$;
3.0   For $generation = 1$ to $\max\_generation$
4.0   Clear the $new\_population$
5.0   Compute the fitness of each individual in the $old\_population$;
6.0   Copy the highest fitness of individual to the $solution\_vector$;
7.0   While the $noof\_individual < population\_size$ do
8.0               Select two parents form the $old\_population$ based
                  on their fitness values;
9.0               Perform the crossover of the parents to produce
                  two offspring;
10.0          Mutate each offspring based on $mutation\_rate$;
11.0          Place the offspring to $new\_population$;
12.0  End while
13.0  Replace the $old\_population$ by $new\_population$.
14.0  End for
15.0  Print out the $solution\_vector$ as the final solution.
```

We can summarise several key features of genetic algorithms as follows (Lin, Kao and Hsu, 1993,p.1754).

1. Genetic algorithms work from a population instead of a single state. By maintaining a population of well-adapted states, the probability of becoming trapped in a local minimum is greatly reduced (Lin, Kao and Hsu, 1993,p.1754).

2. The crossover operation tries to retain genetic information from generation to generation, assuming that the genetic information always contains important substructures of the quality solutions. The average performance of the next generation is better than the previous (Lin, Kao and Hsu, 1993,p.1754).
3. Although the mutation operation has the effect of destroying the structure of a solution; there is still a chance of producing a better one (Lin, Kao and Hsu, 1993,p.1754).
4. Selecting parents based on their fitness values means that parents with a higher value always have a higher probability of contributing one or more offspring in the next generation. Even so, parents with a lower fitness value still have a chance to reproduce. Thus, the probability of escaping from local minima increases (Lin, Kao and Hsu, 1993,p.1754).
5. If the average cost of the population is decreased from generation to generation, one can assure a faster convergence ratio for the genetic algorithm (Lin, Kao and Hsu, 1993,p.1754).

Simulated Annealing

Simulated Annealing (SA) is another nature-based optimisation technique like genetic algorithm. Genetic Algorithms are based on natural evolution while SA is based on thermodynamics. In SA, a control parameter called temperature is used to control the minimisation search, which may occasionally move uphill. The mean and the variance of the cost function are decreasing during the course of the search process. Theoretically, the SA can be viewed as an algorithm that generates a sequence of Markov chains for a sequence of decreasing temperature values. At each temperature, the generation process is repeated again and again until the probability distribution of the system states approaches the Boltzman distribution (Lin, Kao and Hsu, 1993,p.1753). If the temperature is decreased slowly enough, the Boltzman distribution tends to converge to a uniform distribution on the set of globally minimal states. The analysis of the simulated annealing algorithm can be found in the literature cited by Lin, Kao and Hsu (1993) and elsewhere. The SA algorithm does not guarantee finding a global minimum with probability 1.

Major advantages of using nature-based algorithms are their broad applicability, flexibility, ease of implementation, and the potential of finding near-optimal solutions (Lin, Kao, and Hsu, 1993, p.1753). Lin, Kao and Hsu (1993,p.1753) have noted following observations regarding SA algorithm.

1. There is a trade-off between the quality of the final solution obtained and the execution time required by simulated annealing, and the execution time is sensitive to the decrement ratio of the temperature.
2. It is easily trapped to local minima if the temperature drops too quickly.
3. The initial value of temperature effects the total number of iterations required by the annealing.
4. There is still some chance of departing from good solutions if the number of iterations at low temperature regions is not large enough.
5. It is not a trivial task to detect the equilibrium of the system at each temperature, so that the length of the Markov chain may not be easily controlled.

A general Simulated Annealing algorithm is as follows (Lin, Kao and Hsu, 1993, p.1757).

```
1.0   Initialise the parameters of the annealing schedule;
2.0   Randomly generate an initial state as the current state;
3.0   $k = 1$
4.0   Repeat
5.0       Repeat
6.0       Generate next state;
7.0       $\Delta C = Cost\_of\_next\_state - Cost\_of\_current\_state;$
8.0       $P_r = \min\left\{1, \exp\left(\frac{-\Delta C}{T_k}\right)\right\};$
9.0           if $P_r > rand[0,1)$ then $current\_state = next\_state$
10.0      Until system equilibrium at $T_k$;
11.0      $T_{k+1} = T_k * \alpha;$
12.0  Until system has been frozen
13.0  Accept current state as final state.
```

Simulated Annealing and Genetic Algorithm:

Lin, Kao and Hsu (1993) have explained that the probability of the SA algorithm arriving at a better solution increases as the number of iterations are increased at a specific temperature. This characteristic of SA algorithm results in long computation times (Lin, Kao and Hsu, 1993, p.1754).

Lin, Kao and Hsu (1993) have proposed the Annealing-Genetic (AG) technique that combines Genetic Algorithm with Simulated Annealing to design an efficient annealing schedule that improvises SA technique and computes a near optimal solution within a reasonable time.

The AG technique begins with random initialisation of first population strings and calculates the initial temperature of the population by the formula shown below (Lin, Kao and Hsu, 1993, p.1755).

$$initial\_temp = \frac{the\_highest\_cost - the\_lowest\_cost}{population\_size / 2}$$

For each k^{th} epoch, a temporary population of strings (P'_{k+1}) is generated from current population of strings (P_k). To generate (P'_{k+1}), each string from (P_k) is selected in turn and it is randomly modified. The modified string is then added to (P'_{k+1}) if the following condition is true (Lin, Kao and Hsu, 1993, p.1757).

$$\min\left[1, \exp\left(-\Delta C / T_k\right)\right] > random[1,0)$$

where $\Delta C =$ cost of next point – cost of current point

$T_k =$ temperature of current population

The next generation (P_{k+1}) is generated from (P'_{k+1}) after applying various genetic operators. The temperature of the $k+1^{th}$ generation is calculated by following formula (Lin, Kao and Hsu, 1993, p.1755).

$$T_{k+1} = T_k * \alpha$$

where $\alpha =$ temperature coefficient

The annealing process continues until the number of generations are over or majority of the strings are identical in a population. The latter is known as a 'system frozen' condition (Lin, Kao, and Hsu, 1993, p.1755). The AG technique proposed by Lin, Kao and Hsu (1993) is as follows.

```
1.0   Initialise the parameters, ie., $population\_size, T_o$, and
      $\alpha (0 < \alpha < 1)$
2.0   Randomly generate $P_o'$;
3.0   Apply genetic operators to $P_o'$ to create $P_o$;
4.0   Calculate the fitness and the cost for each point in $P_o$;
5.0   Calculate the average cost of $P_o$;
6.0   Solution = Current point = lowest cost point in $P_o$;
7.0   $k = 0$
8.0   While system is not frozen do
9.0   $No\_of\_point = 0;$
10.0  While $no\_of\_point <= population\_size$ do
11.0      Generate next point from current point by a strategy
12.0      $\Delta C = Cost\_of\_next\_point - Cost\_of\_current\_point;$
13.0      $P_r = \min\left[1, \exp\left(\frac{\Delta C}{T_k}\right)\right];$
14.0      if $P_r > random[0,1)$ then
15.0          put next point into $P_{k+1}'$
16.0          $current\_point = next\_point$
17.0          $no\_of\_point = no\_of\_point + 1$
18.0      else pick another point from $P_k$ as $current\_point$;
19.0  end while
21.0  Apply genetic operators to $P_{k+1}'$ to create $P_{k+1}$;
21.0  Calculate the fitness and the costs for each point in $P_{k+1}$;
22.0  Calculate the average cost of $P_{k+1}$;
23.0  If the average cost point in $P_{k+1}$ < $solution\_vector$ then
      update $solution\_vector$;
24.0  If it is the initial stage then determine the initial temperature
      $T_1$;
25.0  Else $T_{k+1} = T_k * \alpha$;
26.0  Current point = lowest cost point in $P_{k+1}$;
27.0  $K = K + 1;$
```

```
28.0  If frozen condition is true then set system is frozen;
29.0  End while
30.0  Perform the local search procedure;
31.0  Print solution vector as solution.
```

The SPORDAC prototype implements the above algorithm to optimise the usage of display area.

The SPORDAC prototype implements the mechanism for generating an initial layout of windows as follows.

Generating initial layout

The user generates an initial layout by repeatedly creating child windows in the parent window opened by the prototype. Pressing down the right mouse button and dragging the mouse to a different location in the parent window and releasing the mouse button creates a child window. Mouse-down and mouse-up points on the canvas determine two opposite points of the child window requested by the user. A child-window represents an instance of *CChildWnd* class and is registered in a class derived from *CDocument* class of MFC framework.

A child-window can be added and deleted from the parent window at any time.

The SPORDAC prototype has captured 'Left-Double-Click' event of the mouse to update the size and location of each child window present in the display area. The SPORDAC algorithm proposed by this thesis is as explained below.

Shadow Propagation for Overlap Removal and Display Area Compaction (SPORDAC) Algorithm Background

The main assumption of SPORDAC algorithm is that every window extends its shadow in all four directions.

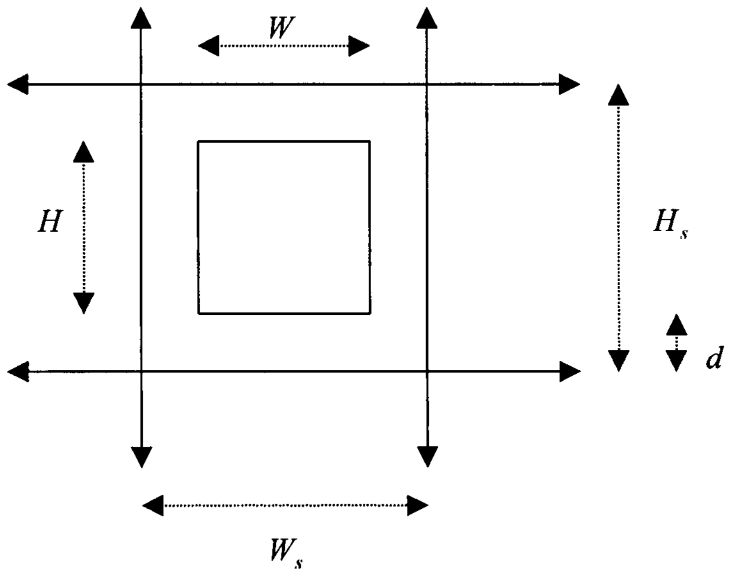


Figure 5.1 An illustration of shadows extended by a window.

The above figure illustrates a rectangular window with width of W units and height of H units. If the inter-window gap is d units then the horizontal and vertical shadows of above window are defined as follows.

Horizontal Shadow

The SPORDAC algorithm defines a horizontal shadow of infinite width and finite height H_s for each window. For the window shown in figure 5.1, the height of the horizontal shadow is calculated as shown below.

$$H_s = H + 2d$$

Where H_s = Height of horizontal shadow
H = Height of window
d = Inter-window gap ($d >= 0$)

The horizontal shadow of a window is assumed to be parallel to the X axis.

Vertical Shadow

The SPORDAC algorithm defines a vertical shadow of infinite height and finite width W_s for each window. For the window shown in figure 5.1, the width of the vertical shadow is calculated as shown below.

$$W_s = W + 2d$$

Where W_s = Height of horizontal shadow
W = Height of window
d = Inter-window gap ($d >= 0$)

The vertical shadow of a window is assumed to be parallel to the Y axis. The SPORDAC is a one-dimensional compaction algorithm. It compacts the layout in one direction at a time. Hence the SPORDAC algorithm is executed twice to achieve compaction in both horizontal and vertical directions. The SPORDAC algorithm considers the shadow in the compaction direction at a time and ignores the shadow in the other direction. Thus only horizontal shadows are considered during horizontal compaction and vertical shadows are considered during vertical compaction.

If the window in figure 5.1 has its center at (X,Y) then the top edge of the horizontal shadow will be at $Y-H/2+d$ and the bottom edge of the horizontal shadow will be at $Y+H/2+d$. Similarly we can say that the right edge of the vertical shadow will be at $X+W/2+d$ and the left edge of the vertical shadow will be at $X-W/2+d$.

A discussion about how the concept of 'shadow' is useful for overlap removal and compaction of display layout follows.

Overlap Removal and Compaction

Suppose two windows with widths W_1 and W_2 have their respective centers at (X_1,Y_1) and (X_2,Y_2). If these windows overlap then it is possible to say that,

$$|X_1 - X_2| < \frac{W_1}{2} + \frac{W_2}{2} + d \text{............(1)}$$

Hence to remove the overlap we must ensure that the distance between X coordinates of two windows is at least equal to $\frac{W_1}{2} + \frac{W_2}{2} + d$. Similar relationships could be derived for the Y direction.

It follows from the earlier discussion about horizontal and vertical shadows that when two windows overlap, they would cross each other's horizontal and vertical shadows. If two windows are not overlapping then they may or may not cross each other's horizontal or vertical shadows.

The following figure illustrates different situations for horizontal and vertical shadow crossings for two windows.

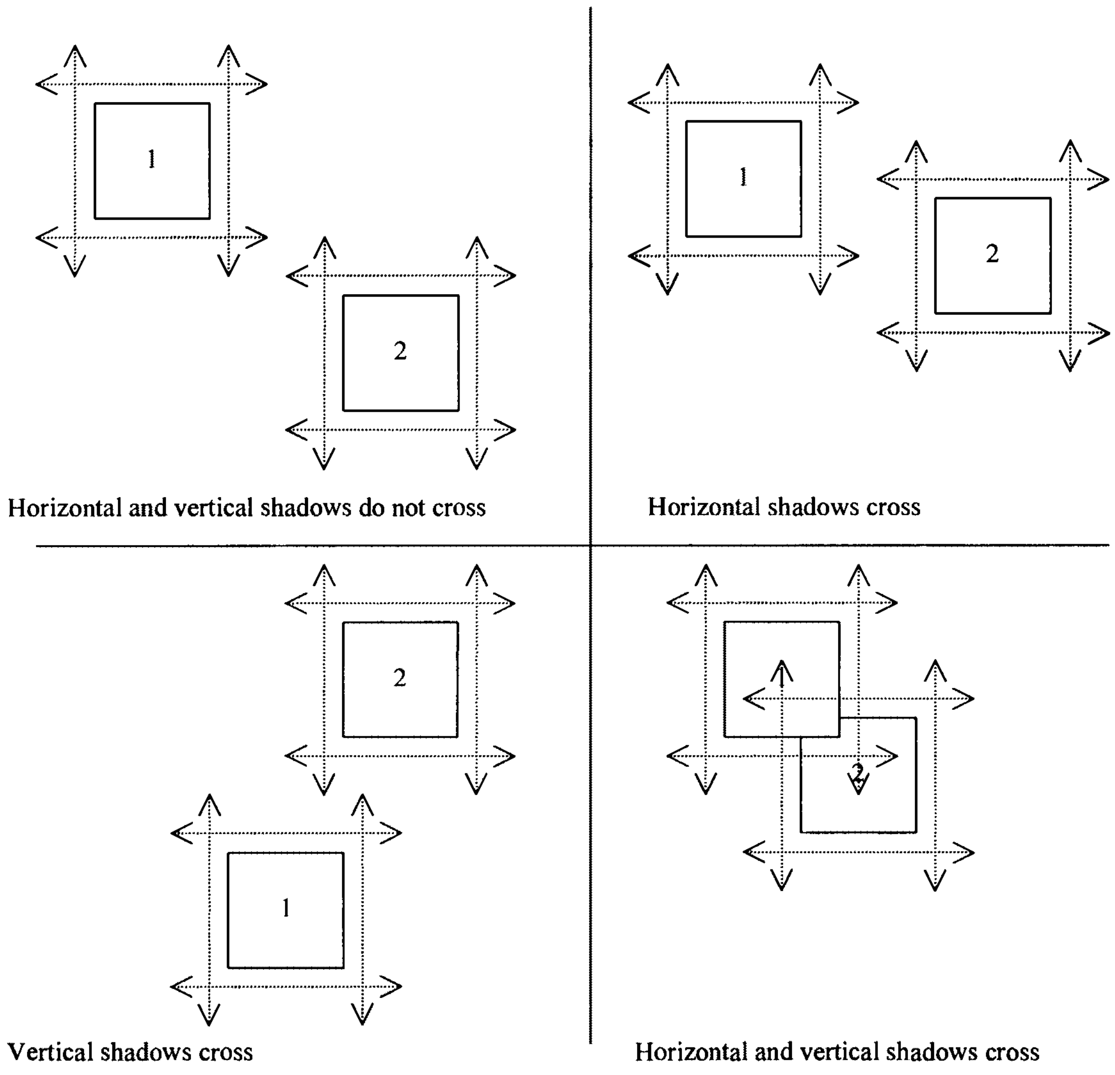

1
2
Horizontal and vertical shadows do not cross
1
2
Horizontal shadows cross
2
1
Vertical shadows cross
2
Horizontal and vertical shadows cross

Figure 5.2 Illustration of two windows crossing each other's shadows

It is clear from the above figure that when the shadows of two windows cross each other, equation (1) is useful in calculating the positions of both windows such that they do not overlap and there is the minimum gap possible in the compaction direction.

The following figure depicts all situations where horizontal shadows of two non-overlapping windows cross each other.

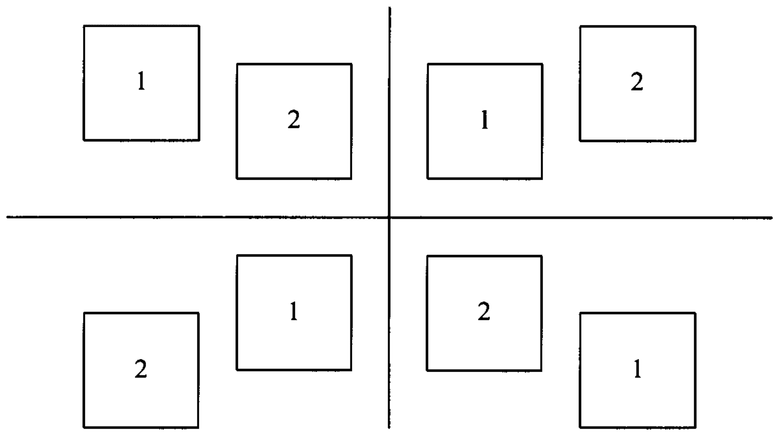


Figure 5.3 All cases of X direction shadow crossings for two windows.

Similar cases for Y direction shadow crossings can be determined as well.

So far we have discussed different situations for two windows. The concept of 'shadow' can be applied to handle any number of windows. The application of 'shadow' in horizontal (X) direction is described below. Similar operation can also be performed in vertical (Y) direction.

The SPORDAC algorithm begins with initialising a list of currently displayed windows in increasing order of X coordinate of their centre. An empty list is initialised to store the information about windows as they are placed on the display area one by one.

The fundamental operation is to read one window at a time from the sorted window list and test whether the current window crosses horizontal shadow(s) of any previously scanned window(s) or not. If the current window crosses the horizontal shadow(s) then X coordinate of its centre is updated to remove any overlaps generated by the current window and compact the layout in horizontal direction. After the current window is placed on the display area, it is added to the list of scanned windows and the process continues until all windows are placed.

The first window read from the sorted list does not cross any horizontal shadow hence the first window is placed on the left edge of the display area without modifying its vertical position.

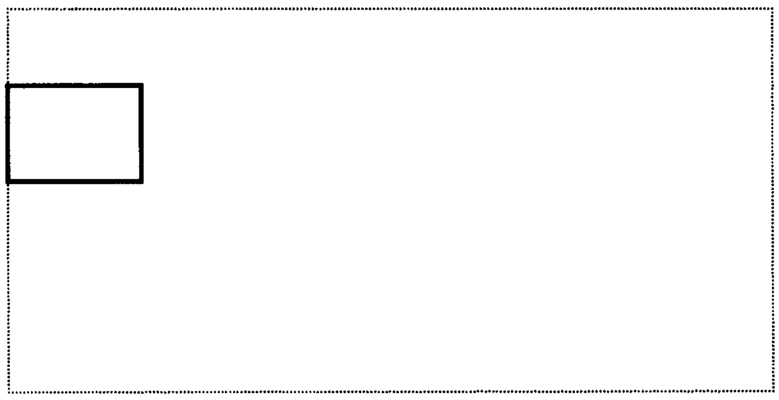
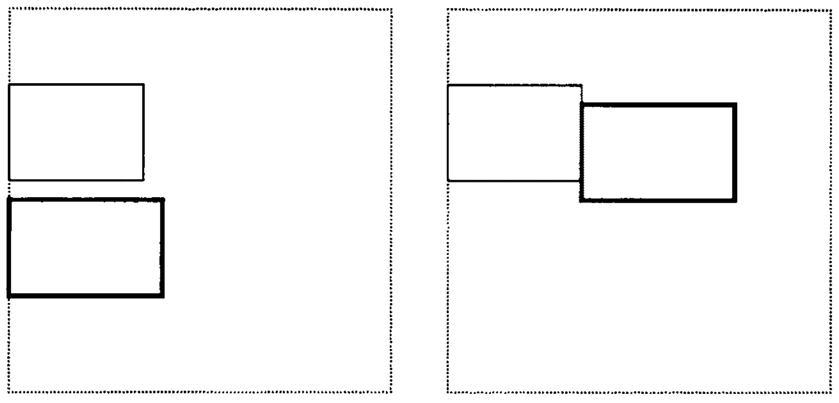


Figure 5.4 Placement of first window in display area.

The second window may or may not cross the horizontal shadow of the first window. If the second window does not cross horizontal shadow of the first window then it is placed on the left edge of the display area.



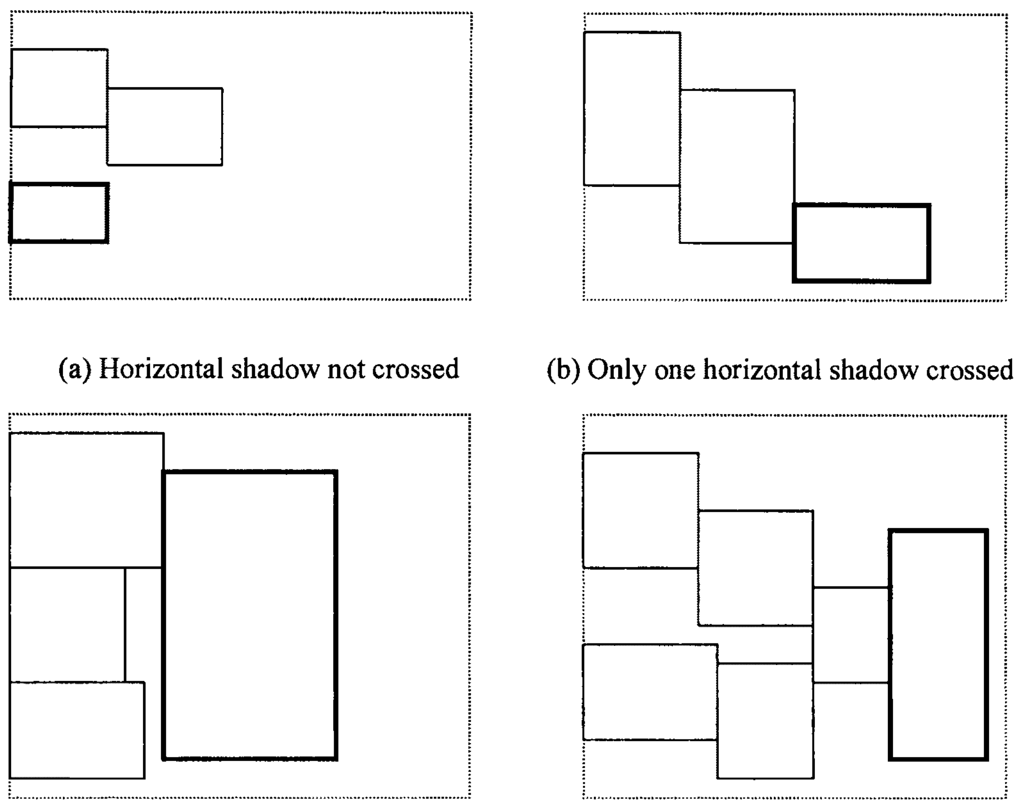
(a) Horizontal shadow not crossed (b) Horizontal shadow crossed

Figure 5.4 Placement of second window in display area.

However if the second window crosses the horizontal shadow of the first window then the second window is placed to the right of the first window such that any existing overlap is removed and only the inter-window gap (d) exists between two windows.

Any window that does not cross the horizontal shadow of previously scanned windows is placed to the left of the display area.

From the third window onwards, it may happen that a window does not cross horizontal shadow of previously scanned windows or it may cross horizontal shadow of one or more previously scanned windows. If the current window is crossing the horizontal shadow of only one previously scanned window then the current window is placed to the right side of that window.



(a) Horizontal shadow not crossed (b) Only one horizontal shadow crossed

(c) More than one horizontal shadow crossed

Figure 5.5 Placement of a window in display area.

If the current window is crossing the horizontal shadow of more than one window then a short list of these windows is made. The current window is crossing the horizontal shadow of every window in the short listed window. The short list is further analysed to mark those windows in the short list that are not on the left hand side of any other window in the short list. Then the window with the highest value of the right hand edge is found from the 'marked' windows. This window is the closest neighbour of the current window. The current window is then placed to the right side of this window. This way the concept of 'shadow' is useful in overlap removal and compaction in one direction.

It is understandable that the process of overlap removal and compaction may generate a display layout that falls short of optimum display area usage or does not fit into the available display area.

The SPORDAC prototype integrates the SPORDAC algorithm and the AG approach to improve display area utilisation. The SPORDAC algorithm places all the windows in a virtual display area of infinite width and height. The SPORDAC prototype scales the solution display layout to the available display area size to generate the final layout.

The formal description of SPORDAC followed by the description of integration of SPORDAC algorithm and AG approach follows.

SPORDAC Algorithm

The following discussion explains horizontal layout compaction from left to right direction using SPORDAC algorithm. A similar operation can be applied in vertical direction to achieve Y compaction.

The SPORDAC algorithm assumes a virtual display area of indefinite size and calculates new position of each window. Then the layout is mapped from virtual display area to physical display area using a simple scaling operation. The scaling operation ensures that the solution layout is encapsulated in the available display area.

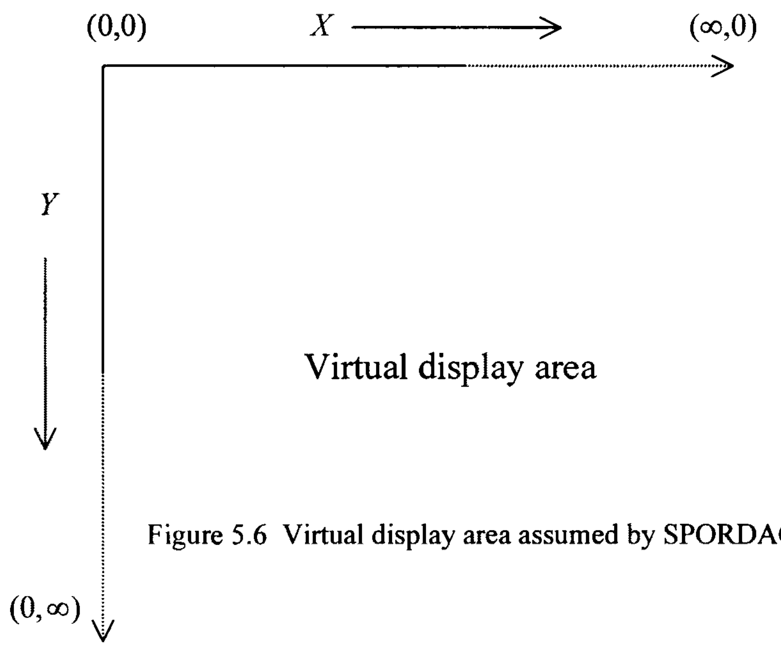


Figure 5.6 Virtual display area assumed by SPORDAC

The block diagram of the whole process is as shown as below.

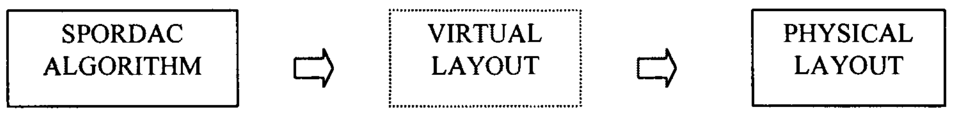


Figure 5.7 Display layout process

General overview of the SPORDAC algorithm is as follows.

```
1.0   Sort all windows in ascending order of X coordinate of their
      centre.
2.0   Initialise an empty scan list.
3.0   While (total scanned windows <= total windows)
4.0       Read next window from the sorted window list.
5.0       Prepare a short list of windows whose X direction
          shadows are crossed by X direction shadow of current
          window from the list of already scanned windows.
6.0       Mark only those windows from the short list, which
          are not at the left side of any other window in the
          short list.
7.0       Find the marked window with highest value of X
          coordinate for it's right edge.
8.0       Update the left coordinate of the current window with
          the sum of inter window gap and the highest value of
          the right edge coordinate found in the previous step.
9.0       If the short list found in step 5.0 is empty then
          initialise the left coordinate of the current window
          with the value of inter window gap.
10.0      Add current window to scan list.
11.0      Update necessary counters.
11.0  End.
```

Let us consider the simple example given below to see how this algorithm works.

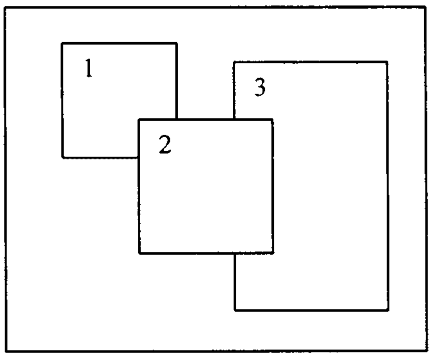


Figure 5.8 Example layout of windows after window sizes are recalculated.

Step 1.0 Sorted list of windows would be {1,2,3}.

Step 2.0 Scan list is initialised to empty list.

Step 3.0 1 is selected as the current window.

Step 4.0 1 does not cross X direction shadows of any previously scanned window. Hence 1 is placed at the left edge of the display area.

Step 5.0 1 is placed in the scanned window list. Display layout at this stage is as shown below.

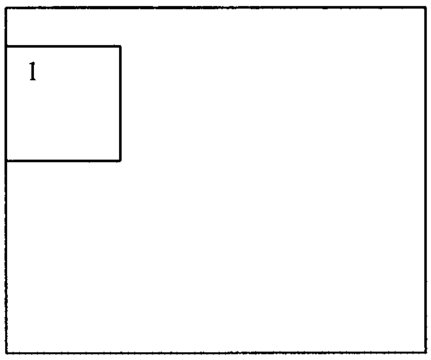


Figure 5.9 Layout after first window is re-positioned

Step 6.0 2 is selected as the current window.

Step 7.0 2 crosses X direction shadow of 1. Hence 2 is placed next to 1. Now the display layout is arranged as follows.

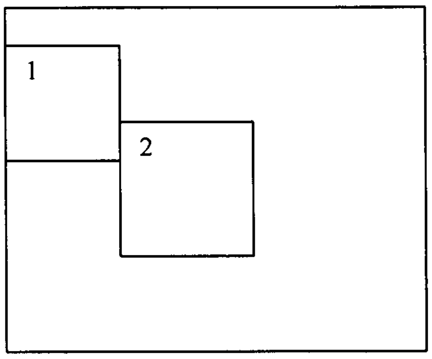


Figure 5.10 Layout after second window is re-positioned

Step 8.0 2 is added to scan list.

Step 9.0 3 is selected as current window.

Step 10.0 3 crosses X direction shadows of 1 and 2. Step 6.0 of the algorithm selects 2 as a direct neighbour of 3 because 2 is at the right side of 1.

Step 11.0 3 is placed to the right side of 2. Display layout at this stage becomes as shown below and the process stops.

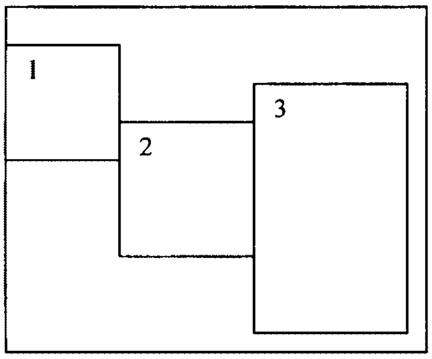


Figure 5.11 Layout generated by X compaction process

Y compaction applied on above layout will produce final layout as follows because Y directional shadows are not crossed.

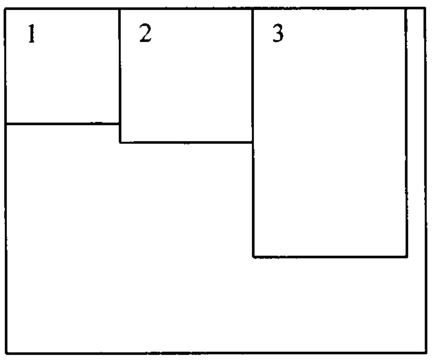


Figure 5.12 Layout generated after Y compaction applied to figure 5.11

The above layout is generated in the virtual display area explained earlier. Scaling all the windows to the physical display area generates the final layout.

It is noted earlier in this Chapter that two independent one dimensional compaction operations may not result in optimum usage of display area. The SPORDAC prototype integrates the SPORDAC algorithm and the Annealing Genetic method proposed by Lin, Kao and Hsu (1993) to promote display area optimisation.

A general overview of the AG approach is as outlined earlier in the chapter. The remainder of the chapter describes how SPORDAC algorithm is integrated with AG approach to calculate final display layout.

Integration of SPORDAC with AG Approach

The SPORDAC prototype consists of the SPORDAC algorithm integrated with the AG approach along with suitable Graphical User Interface (GUI). The main focus of this section is to explain integration of the SPORDAC algorithm with the AG approach and describe functioning of different genetic algorithm operators in the SPORDAC prototype.

It is explained earlier in this chapter that AG approach continually operates on a set of likely candidates for the final solution until a certain terminating condition is met. Each candidate solution is known as a 'string' and the set of strings is known as a 'population'. Each string in a population represents a likely solution to the problem being solved. Therefore each string in the population initialised by the SPORDAC prototype should represent a compact non-overlapping display layout. Every string in the SPORDAC prototype is represented by an instance of *CGenString* class.

The *CGenString* class encapsulates the data structure to represent the display layout and implements methods to access and manipulate the encapsulated display layout information. An array of integers represents the display layout information as shown below.

$$Id_1, left_1, top_1, width_1, height_1, \ldots, Id_i, left_i, top_i, width_i, height_i, \ldots, Id_n, left_n, top_n, width_n, height_n$$

Where,

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

or

$$top_1 + \frac{height_1}{2} \le \ldots\ldots \le top_i + \frac{height_i}{2} \le \ldots\ldots \le left_n + \frac{left_n}{2} \; .$$

and

$$Id_1 \ne Id_2 \ne \ldots\ldots Id_{i-1} \ne Id_i \ne Id_{i+1} \ldots\ldots \ne Id_n$$

The display layout is represented as a dynamic array of integers to store information of each window's ID, left edge coordinate, top edge coordinate, width and height.

The SPORDAC algorithm and the scaling operation are implemented as private methods of *CGenString* class. The constructor of CGenString class calls these methods. A constructor is the default method that is automatically called when an instance of a class is generated. The SPORDAC prototype passes the display layout information as an array of integers to the constructor of the *CGenString* class to create an instance of a string. The constructor of the *CGenString* class calls the method implementing the SPORDAC algorithm to remove any existing overlaps and compact the layout in both directions. Then the constructor of the *CGenString* class calls the method to scale the computed layout to the available display area and calculate the cost of the computed layout. The cost of the computed layout represents the goodness of the string. The cost of the string is calculated as shown below.

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

where $Void\_area =$ total unoccupied area in display layout

In the best case scenario, the solution will fully utilise the available display area and would be free of void area. Hence the minimum value possible for the cost of a string is 0. In the worst case scenario the solution could have its cost very close to 1 but less than 1. The value is always less than 1 because the SPORDAC prototype does not allow a window size to be reduced to 0.

The process of initialising a string utilises the SPORDAC algorithm to ensure that it represents a likely candidate solution. The population of the strings is then acted upon by the AG approach to compute the final solution. This way the SPORDAC algorithm and the AG approach are integrated by the SPORDAC prototype.

The AG approach implemented by the SPORDAC prototype utilises mutation and crossover operators to manipulate the population of the strings. Some of the important considerations for the implementation of mutation and crossover operators are as follows.

Firstly, the user interaction with the open windows decides their width and height. Hence the mutation and crossover operators need not modify these properties of a window. However, they can modify the left and top edge values of a window.

Secondly, the resultant layout generated by mutation and crossover operators may have overlaps and may not fit in the available display area. The resultant child strings are used to generate instances of the *CGenString* class to remove overlaps.

The operation of the mutation and crossover operators is as described below.

Mutation

Suppose that the mutation operator is to operate on a string that represents following display layout.

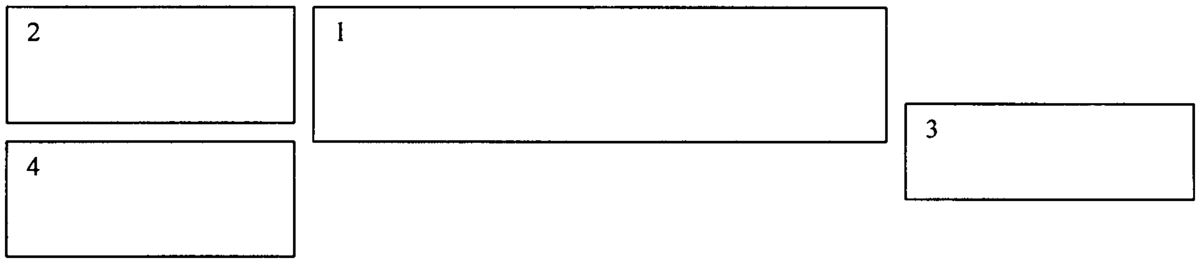


Figure 5.13 Initial windows layout

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

$Id_1 = 2, left_1 = 0, top_1 = 0, width_1 = 100, height_1 = 50$,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 it's top edge and left edge values are modified as shown below.

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

This string represents the display layout as shown below.

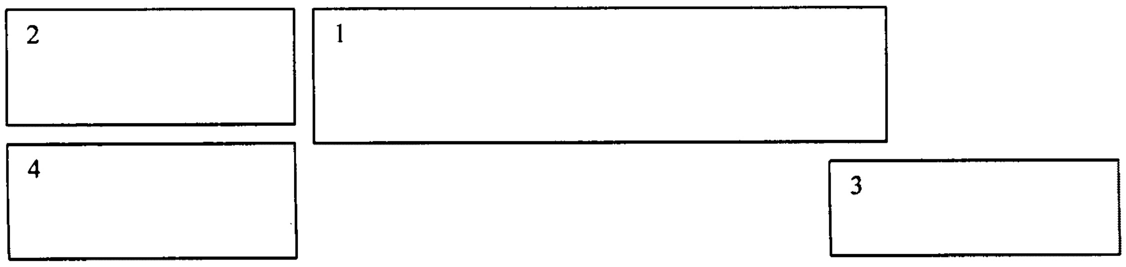


Figure 5.14 Windows layout after manipulation by mutation operator

If the inter window gap is set to 0 then the final compacted layout will be as shown below.

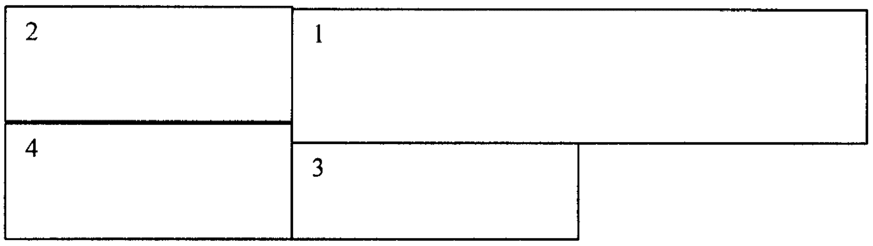


Figure 5.15 Final layout after compaction

For above display layout, the genetic string would be initialised with following values.

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

Crossover

The crossover operator has to consider the situation where discrepancies may arise after the crossover operation. The initial population is randomly generated and each string in a population always represents a non-overlapping compact display layout.

It is easy to understand that each string will represent same number of windows. However, random initialisation of genetic strings may not arrange windows at the same location in every string. Hence it is possible that two strings will represent same number of windows but the order of windows may not be identical.

Therefore it may not be possible to perform crossover operation depending upon physical location on string. Also, the width and the height of any window should not be modified during crossover operation.

The crossover operator randomly selects window and swaps left and top edges coordinate values of windows whose Ids are more than selected window's Id. For example, if a display layout has ten windows and crossover operator selects fifth window then left and top edge coordinate values of all windows from sixth window to tenth window are swapped. This ensures that both children have same number of windows and does not generate any discrepancy.

An example of crossover operation is explained in the figure shown on the following page. The crossover operator has selected second window.

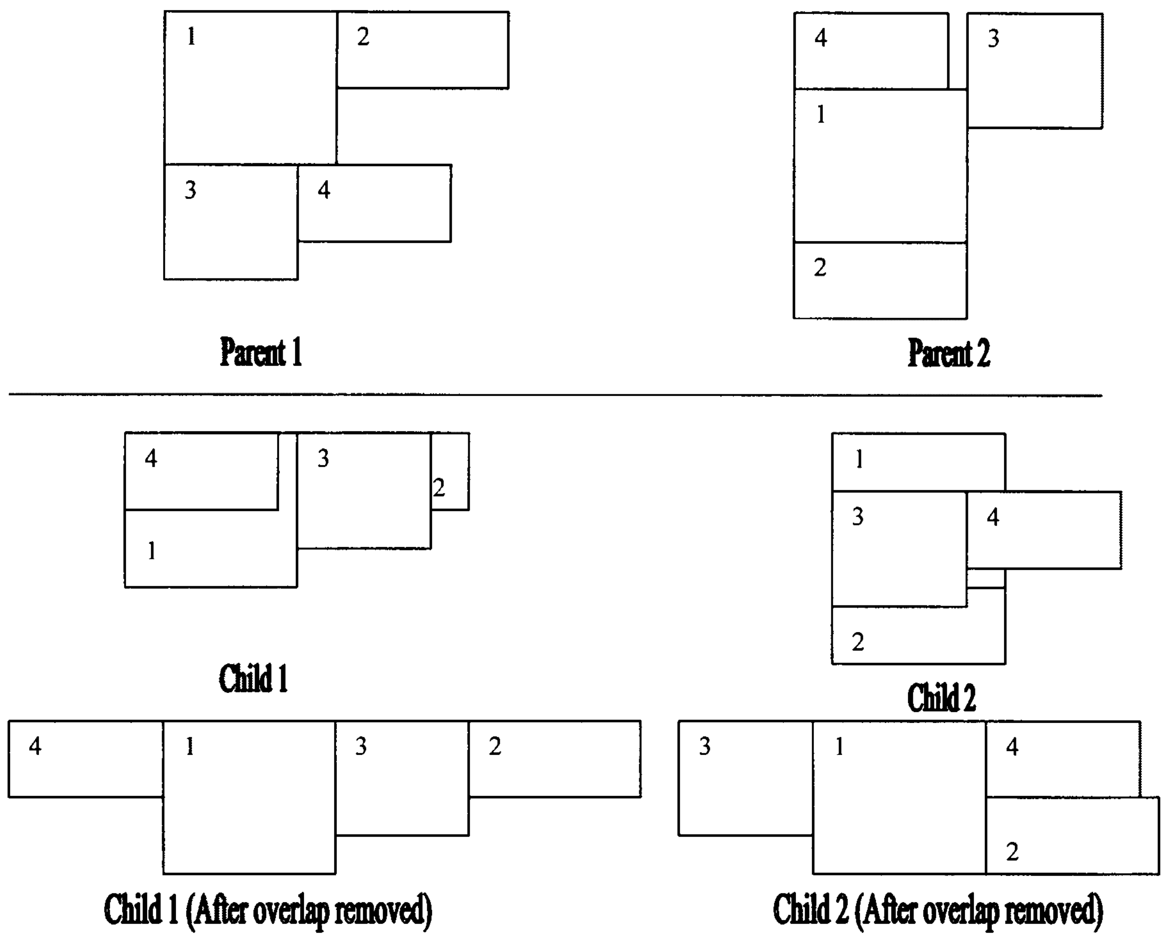

1
2
3
4
Parent 1
4
3
1
2
Parent 2
4
3
2
1
Child 1
1
3
4
2
Child 2
4
1
3
2
Child 1 (After overlap removed)
3
1
4
2
Child 2 (After overlap removed)

Figure 5.16 Example of crossover operation

The above figure illustrates how crossover operation is useful in generating alternative solutions. The crossover operation generates two children strings with overlaps. The array of integers representing each child is used to construct a corresponding instance of *CGenString* class. As explained earlier in the chapter, this would create two solution display layouts as shown in the figure 5.16.

Next follows the description of how the final solution is computed by the SPORDAC prototype.

The user generates initial layout by creating one or many child windows on the display area and interacts with any one of the child windows by double clicking in its client area. This interaction triggers the SPORDAC prototype to recalculate new sizes of each window. An instance of *CGenString* class is created using the new sizes of the windows. This string object holds the temporary solution.

Next, the AG approach is executed to calculate the final solution. The AG approach begins with random initialisation of initial population. The AG approach stops execution if a string with the cost of 0 is found or user-specified number of generations has been evolved.

The solution computed by the AG approach is compared with the temporary solution initialised at the beginning of the process. The better solution is then mapped to the display area.

A general overview about how the final solution is computed by the SPORDAC prototype follows.

1.0 *The user generates initial layout of windows that may contain overlaps.*
2.0 *The user double clicks on a child window. The new size of each window is calculated.*
3.0 *An instance of a class (CGenetic) that encapsulates AG approach is created.*
4.0 *A string of integers representing the layout of windows with their new sizes is passed to the instance of CGenetic created in step 3.0.*
5.0 *The AG implementation of CGenetic initialises the population of necessary strings by creating instances of CGenString class as and when required.*
6.0 *The AG algorithm executes and returns with the solution.*
7.0 *The solution is mapped on available display area.*

This concludes the discussion about how the SPORDAC algorithm and the AG approach are integrated in the SPORDAC prototype.

A discussion about implementation of the SPORDAC prototype follows.

Object Model

The object model of the SPORDAC prototype is as shown below.

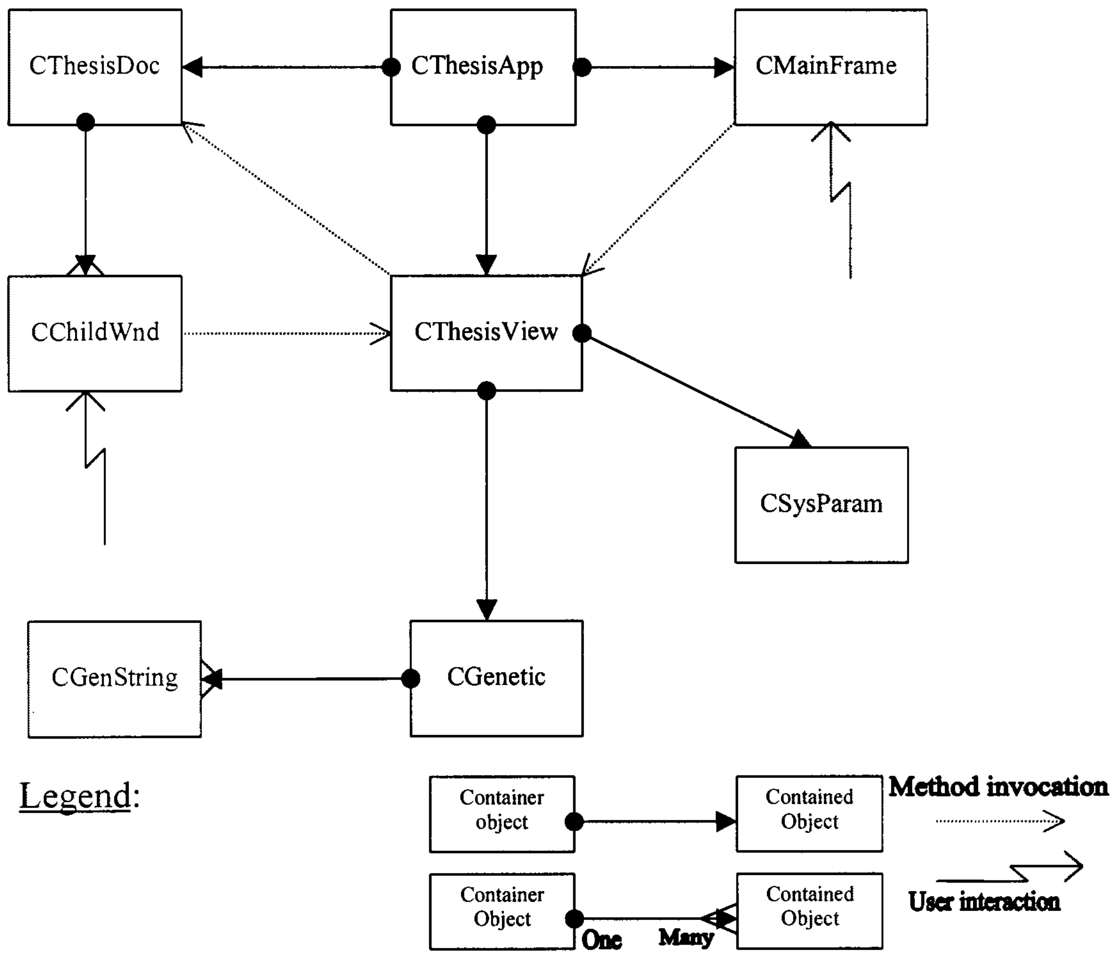


Figure 5.17 The object model of SPORDAC prototype

The above figure represents the relationship between various C++ classes implemented for the SPORDAC prototype. The Visual C++ application framework has generated *CThesisApp*, *CThesisDoc*, *CThesisView* and *CMainFrame* classes. The *CMainFrame* class represents the underlying parent window for the SPORDAC prototype. The *CThesisDoc* and

CThesisView classes represent the *Document/View* architecture of the SPORDAC prototype. It is sufficient to note here that all user interactions performed on parent window are diverted to the running instance of the *CThesisView* class. We request the reader to refer to appendix for more information on the *Document/View* architecture. The relevance of *CGenetic* and *CGenString* classes is already explained earlier in the chapter. The *CChildWnd* class represents a child window created by the user on the underlying parent window. The *CThesisDoc* class maintains a list of all valid *CChildWnd* instances. The *CSysParam* class represents the dialog box that enables a user to modify certain system parameters such as population size, number of generations, probability of crossover, probability of mutation, inter-window gap, etc.

The display layout process begins when a user double clicks on a child window. The appropriate event handler in the child window invokes a method in the running instance of the *CThesisView* class to recalculate the size of each window and compute display area. The user interactions to create a new child window are also handled by the instance of *CThesisView* class.

This concludes the discussion of the SPRODAC prototype to solve the display layout problem. The following chapter presents the results obtained by the research and analyses them.

This chapter discusses the results of the SPORDAC prototype and analyses them. This chapter begins with the description of various sample display layouts generated by the SPORDAC prototype followed by the analysis of the results generated by the prototype.

Some of the sample layouts generated by SPORDAC prototype are as shown below. Every window clicked by the user is marked with ●.

SAMPLES

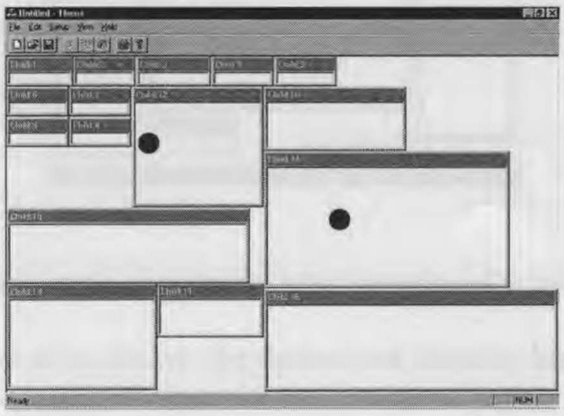


Figure 6.1 Sample layout generated by SPORDAC prototype

The above layout was generated by the SPORDAC prototype after removing overlaps from the windows and compacting them.

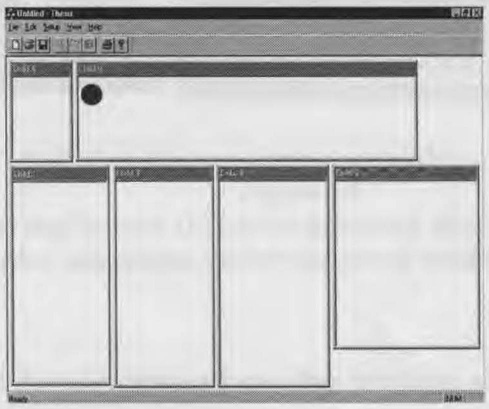


Figure 6.2 Compact and optimum layout generated by SPORDAC prototype

The above figure shows the optimised display layout generated by the SPORDAC prototype after user interaction with the central child window. It is apparent from the layout that the prototype has generated the best solution possible. Also, the interacted window has relatively large area.

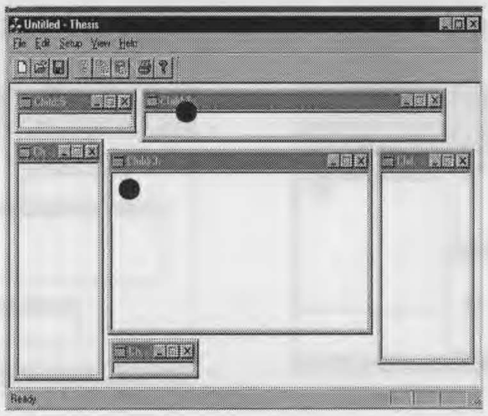
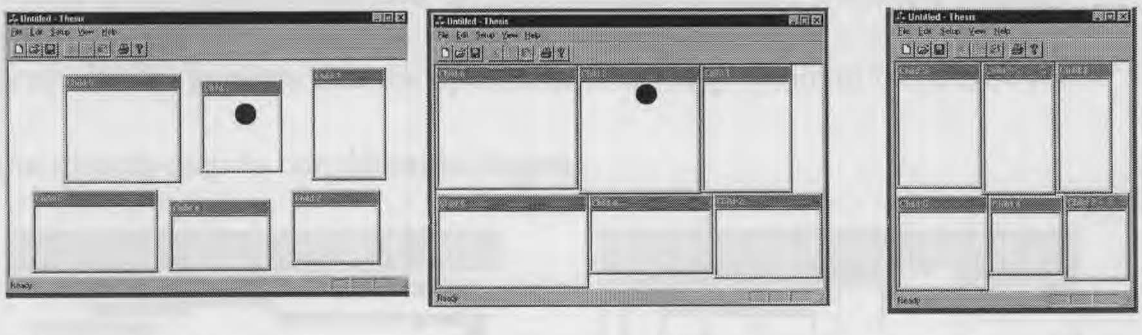


Figure 6.3 Optimised layout generated after user interaction with the marked window

The above figure also shows the optimised display layout calculated by the SPORDAC prototype.

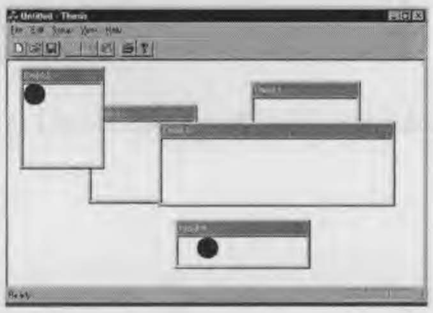


(a) (b) (c)

Figure 6.4

(a) Initial layout generated by user; (b) Layout generated after user clicks marked window; (c) Layout generated after user resizes underlying parent window.

The above figure demonstrates how the relative size of the clicked window increases in the final layout. One can also observe that the relative positions of the windows and their aspect ratios are maintained in the resultant layout computed by the SPORDAC algorithm. The figure 6.4(c) demonstrates that the final layout is encapsulated in the available display area after its size is modified. One can observe that the SPORDAC algorithm has preserved the mental-map of the diagram.


(a) Initial layout generated by the user

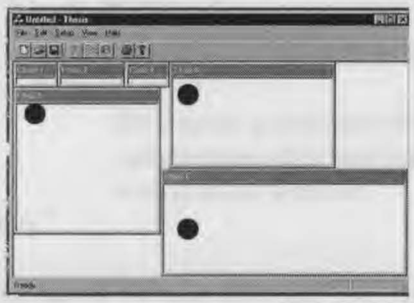
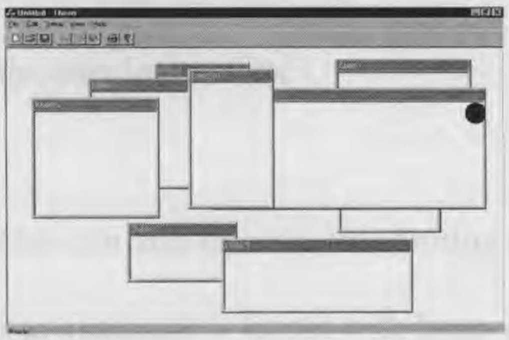

(b) Final layout

Figure 6.5

Figure 6.5(a) shows the initial layout generated by the user. The child window-6 is completely overlapped in the initial layout. After several user interactions with the windows 4,5, and 6, one can observe that the windows do not overlap, and they are contained in the available display area. It also appears that the operation of overlap removal has destroyed the mental-map to considerable degree.


(a) Initial layout generated by the user

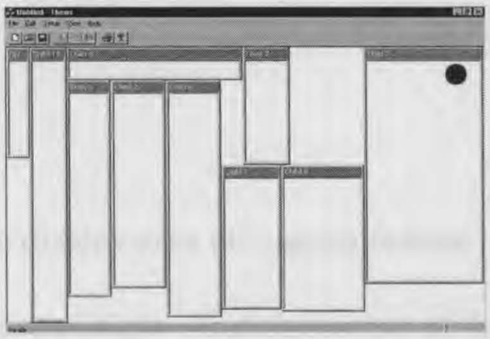
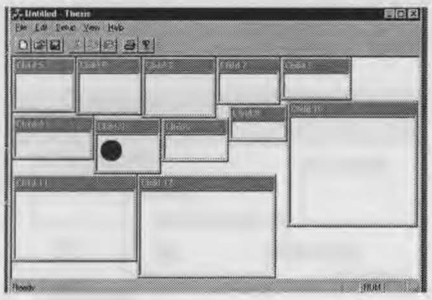

(b) Optimised final layout

Figure 6.6

The figure 6.6 is another example of optimised display layout calculated by the prototype.


(a) Layout calculated by the algorithm

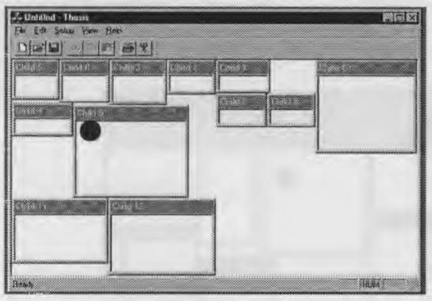
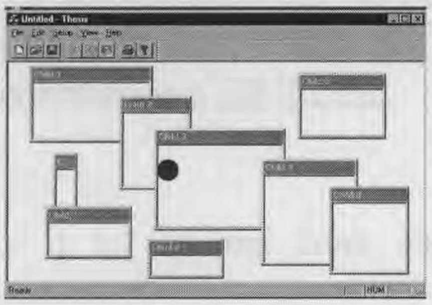

(b) Layout generated without area optimisation after user interaction with marked window.

Figure 6.7

The figure 6.7(a) shows the layout generated by the SPORDAC prototype with display area utilisation turned on. While the figure 6.7(b) shows the layout generated by the SPORDAC prototype with display area utilisation feature turned off.

This experiment and comparison of figure 6.4 with figure 6.7 suggests that the prototype is able to maintain the mental map of the display area to fair degree if the original or the initial layout was overlap free before the user interaction.

One can also observe that turning off the display area utilisation feature has generated a layout with large void around right – bottom corner of the parent window. This is understandable as the SPORDAC implementation first generates the layout from left to right followed by a similar operation in top to bottom direction.


(a) Initial layout

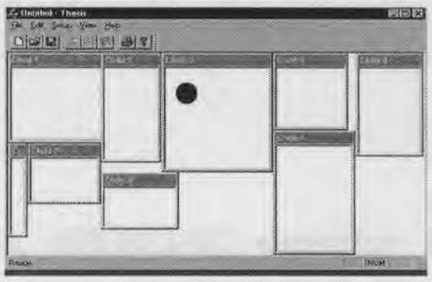

(b) Final layout

Figure 6.8

The figure 6.8(a) shows the initial layout generated by the user. The figure 6.8(b) shows the layout generated by applying the SPORDAC algorithm in Y direction followed by X direction. We can observe from previous examples that the order in which the SPORDAC algorithm is applied makes a difference to the quality of layout generated.

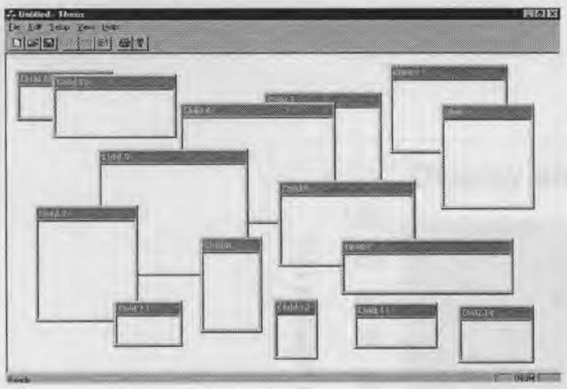

Initial layout

Figure 6.9

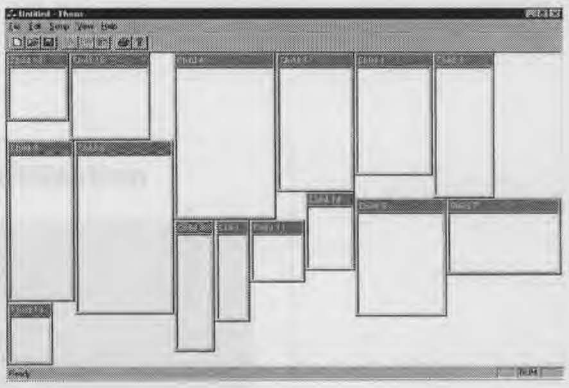

Optimised layout generated by the prototype

Figure 6.10

The figure 6.9 shows the initial layout generated by the user and figure 6.10 shows the optimised layout generated by the prototype with SPORDAC algorithm applied in Y direction followed by X direction.

Performance of SPORDAC

As it is apparent from above examples, SPORDAC algorithm is successful in removing overlap from windows layout and compacts the layout to reasonable degree. The prototype was tested on 133MHz Pentium machine with display area of about 600 X 300 pixel. In the best case, the prototype has been successful in achieving compaction with around 93% utilisation of display area with 15 windows open.
In the worst case we have observed 65% utilisation of display area with 15 windows open. On average we were able to achieve 75% utilisation of display area with 15 windows open.

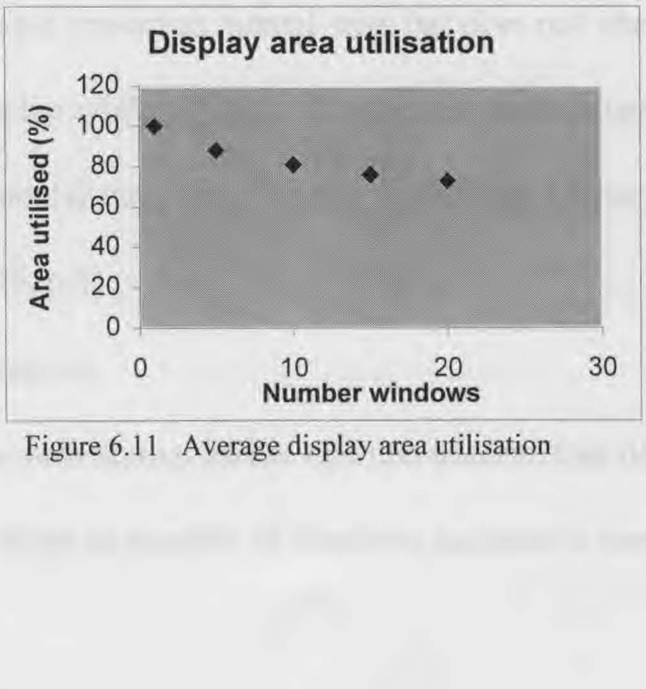


Figure 6.11 Average display area utilisation

CHAPTER : 6

The Results

Above graph depicts average display area utilisation performance for different number of windows. For each case, the prototype was run ten times.

One can observe that the final display area optimisation and mental-map preservation achieved by the SPORDAC prototype depends upon following aspects.

- Initial layout

  Figures 6.8 and 6.11 suggest that if the initial layout is overlap free then the resultant layout tends to maintain the mental-map to a reasonable degree. If the initial layout has overlapping windows then the overlap removal operation performed by SPORDAC is working against the preservation of mental-map.

  This characteristic seems to be like a counter example of SHriMP view technique. Chapter-2 of the thesis observes that SHriMP view technique preserves mental-map but does not attempt to remove overlap while SPORDAC removes the overlap and maintains mental-map under certain conditions (Storey and Muller, 1995, p.3).

- Number of windows

  From the layouts shown earlier one can observe that display area utilisation drops as number of windows increase in number.

- Nature of interaction

  Figure 6.9 is an example of a situation where the user has diverted most of interaction to three windows in particular. Hence these windows occupy maximum display area and other smaller windows are placed around bigger windows. It is apparent that this situation results in a better utilisation of display area.

- Size modification

  With every interaction the SPORDAC prototype increases the width and the height of the clicked window by 15% and decreases the width and the height of other windows by 15%. Thus, window sizes are modified linearly.

  Experimenting with exponential functions for window size modification resulted in drastic changes in sizes of windows. Hence such functions were not utilised for SPORDAC and their usage was subsequently stopped for this research.

- Order of compaction

  The reader can observe differences in the windows layout arrangement while comparing figures 6.12, 6.13 and 6.14 with rest of the figures. SPORDAC is a one dimensional compaction method. Hence to compact the layout in both directions, the prototype applies the same algorithm in two different directions.

The order selected makes an impact on the amount of compaction achieved by SPORDAC.

The AG approach implemented for SPORDAC produced the following characteristic graph.

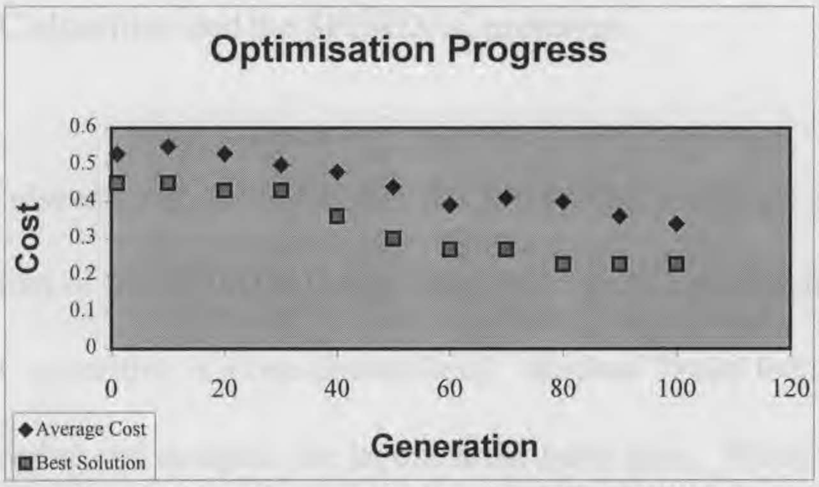


Figure 6.12 Performance of AG approach for SPORDAC

The above figure plots the performance of AG approach for SPORDAC prototype. It is evident from the graph that the annealing process starts with a relatively high value of cost. When the process reaches around 40th generation, quality of the solution improves by a fair degree. From 60th generation onwards we see relatively smaller improvement in the solution. The plot resembles in nature to the curve produced by Lin,Kao and Hsu (1993) for their implementation of AG method for set partitioning problem.

The above plot was obtained for running optimisation process for 100 generations with value of $P_c = 0.05$, $P_m = 0.005$ and population size of 75. The value of temperature coefficient (α) was set to the value of 0.75 by trial and error.

Next follows the discussion about the execution time requirements for the SPORDAC algorithm and the SPORDAC prototype.

It is noted elsewhere in the thesis that the SPORDAC prototype consists of integration of the SPORDAC algorithm with the AG approach. The SPORDAC algorithm is a one-dimensional, 'shadow' based technique to remove overlap and compact the layout at the same time. While AG approach is a combination of annealing technique and genetic algorithm. The SPORDAC prototype accommodates the SPORDAC algorithm with the genetic algorithm proposed by the AG approach.

It follows from the description of the SPORDAC algorithm that it mainly involves sorting and searching of the windows or display layout objects. Unlike the AG approach, the SPORDAC algorithm does not involve any non-deterministic calculations based on a random number generator. Therefore the execution time required by the SPORDAC algorithm depends upon the searching and sorting algorithms implemented by the user of the SPORDAC algorithm. This thesis has implemented binary searching and sorting techniques for implementation of the SPORDAC.

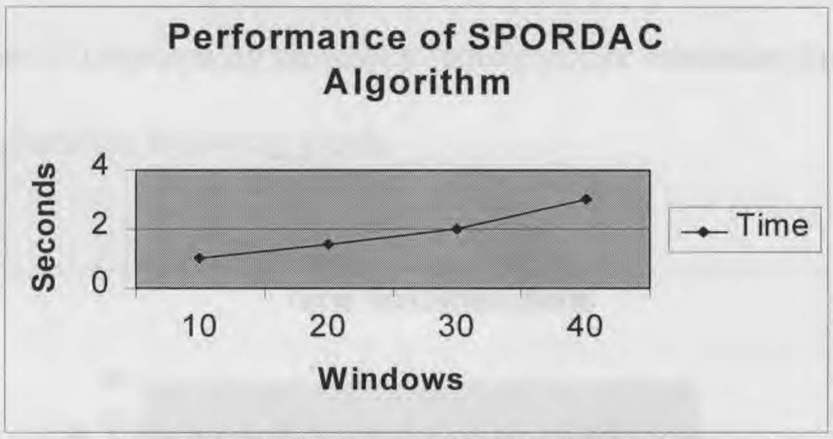


Figure 6.13 Performance of SPORDAC algorithm

The figure 6.13 shows the execution time required by the SPORDAC algorithm to generate a layout. The above timings were observed for application of the SPORDAC algorithm in both directions with AG approach switched off. The SPORDAC algorithm was applied in the X direction followed by the Y direction. The resultant layouts generated were similar in nature to figure 6.7(b), with void areas concentrated on the bottom-right corner of the display area, depending upon the window clicked by the user. Hence one can observe that, the SPORDAC algorithm is quick to generate an overlap free layout but the layout may not utilise the display area very well.

The SPORDAC prototype has integrated the SPORDAC algorithm with the AG approach to optimise the display area usage,. The AG algorithm operates on the display layout generated by the SPORDAC algorithm and attempts to improvise the layout.

The process of improvising the layout requires more execution time as it is evident from the following graph.

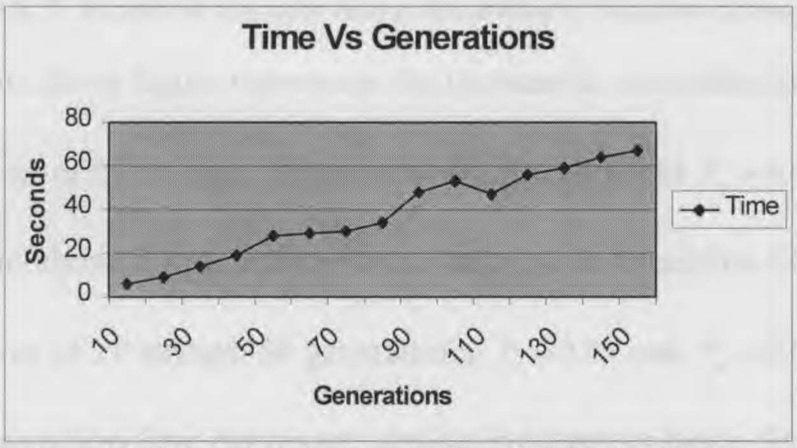


Figure 6.14 Execution time requirements for AG algorithm

The figure 6.14 shows the execution time requirements for the SPORDAC prototype's implementation of the AG approach. The above timing characteristics were observed for five windows with population size of 30 strings, $P_c = 0.05$ and $P_m = 0.005$.

The following figure demonstrates the increase in execution time with respect to increase in the number of windows.

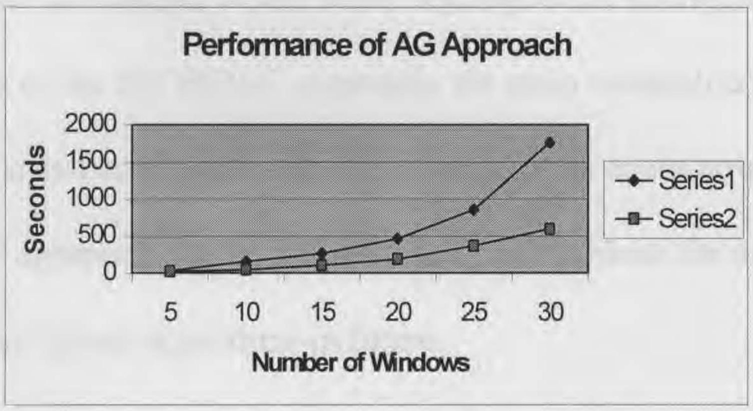


Figure 6.15 Increase in execution time as the number of windows increase

The series1 in above figure represents the increase in execution time for population size of 50 strings, 75 generations, $P_c = 0.05$ and $P_m = 0.005$. The series2 in above figure represents the increase in execution time for population size of 20 strings, 30 generations, $P_c = 0.05$ and $P_m = 0.005$. The above execution time curves are similar in nature to those obtained by Lin, Kao and Hsu (1995) for set partitioning problem.

It follows from the above graph that the execution time required by the AG approach is dependent on the genetic algorithm parameters selected by the user. The execution time increases as the population size and the number of generations increases.

Next and the last chapter of this thesis highlights the strengths and the weaknesses of the SPORDAC approach, the main research contribution of this thesis and concludes with a few suggestions about how the SPORDAC approach can be utilised as a building block for designing other display layout algorithms in future.

CHAPTER : 7

The Conclusion

This concluding chapter of the thesis will highlight the main research contribution, strengths and weaknesses of the SPORDAC approach and mention how the SPORDAC approach could be useful in developing different layout algorithms.

Research findings

The hypothesis stated in Chapter-1 and Chapter-4 had envisaged an automatic interactive window layout generator that can handle any number of windows and arrange them in such a way that they do not overlap, optimise the display area usage and encapsulate the layout in available display area.

This research has been successful in developing a unique SPORDAC algorithm for display layout. This algorithm has succeeded in removing overlap from the display layout. The SPORDAC method is fast and easy to implement as it mainly involves searching and sorting. Implementing efficient searching and sorting algorithms for shadow propagation results in faster operation. The algorithm is able to remove overlap from a given display layout within few seconds.

The algorithm removes overlap and arranges windows in only one dimension at a time. Hence display area utilisation is not always optimum. However, the algorithm arranges windows as close as possible in one dimension and removes overlap at the same time.

Using Annealing Genetic (AG) approach with SPORDAC algorithm results in better utilisation of the display area. The uniform scaling maps the final layout to the available display area. In this way the SPORDAC prototype has been able to achieve non-overlapping, optimised, compact layout which fits in available display area.

It was stated in Chapter-4 that the SPORDAC algorithm should preserve the mental map of the layout to a reasonable degree.

As it is observed earlier in Chapter-1, preservation of the orthogonal ordering is an important condition in preserving the mental map of the user. Sometimes it is necessary to modify orthogonal ordering of the objects to produce a non-overlapping, compact layout.

Hence referring to display layouts generated by the SPORDAC algorithm and the subsequent description, it is safe to state that the SPORDAC algorithm is able to maintain the mental-map under certain conditions.

The chapter-2 of this thesis has surveyed some of the prominent display layout algorithms. Most of the literature surveyed has highlighted final display layout generated by their respective approaches in their respective literary work. Unfortunately very little numerical data was found that would directly relate to numerical data gathered by this research, ie, display area utilisation factors, measure of mental-map preservation, etc. The SPORDAC prototype has utilised AG approach for display area utilisation and Chapter-6 has appropriately acknowledged the performance of this approach.

Successful development and implementation of a new approach to generate layout of dynamic windows has been the main research achievement.

Future research directions

It is observed in Chapter-6 that performance of the SPORDAC algorithm depends upon several factors. One of the important factors is the function used to modify the size of a child window. This research has implemented a linear function to modify the size. It should be an interesting research topic to find a window size manipulation function that also takes mental-map into consideration.

The SPORDAC prototype integrates the SPORDAC algorithm with the AG approach to optimise the display area usage. The figure 6.13 suggests that the SPORDAC algorithm is able to handle significant number of windows in reasonable time. However figure 6.14 and figure 6.15 suggests that the time required for calculating the final solution increases considerably as the number of window increases. Improving the speed of the annealing process is another area to be looked at.

Bac and Perov (1993) have suggested several interesting variations on genetic algorithms. Some of the main variations include application of mutation operator to every string in the population before generating next population, implementing multi-point crossover operators, dynamic calculation of population size, crossover and mutation probabilities, etc (Bac and Perov, 1993, p.230-233).

It should be an interesting research exercise to study the effects of above stated enhancements to the speed of annealing process implemented by the SPORDAC prototype.

It is noted several times that the SPORDAC algorithm is a one-dimensional compaction method. It should be possible to modify the SPORDAC algorithm proposed here to design a two dimensional approach which aims to optimise and compact the display area usage at the same time. The SPORDAC algorithm is a one-dimensional compaction approach. Hence the layout generated needs to be optimised using AG approach. An improved two-dimensional SPORDAC method may not require implementation of annealing operation. This should further improve the speed of the SPORDAC prototype.

The concept of 'Shadow' has the potential to flourish into more sophisticated interactive display layout algorithms.

One such algorithm could consider shadows in both dimensions and directions to calculate the minimum distance required for removing overlap introduced by a window.

Another approach could be to start the arrangement of windows from centre of display area and place all the windows along a hypothetical clockwise or anti-clockwise spiral. This algorithm could use the concept of shadow to move a new window in such a way that the resultant layout has minimum void.

In conclusion, our research has been successful in developing a new approach for display layout.

<u>APPENDIX</u>

Object Orientation

Object Orientation (OO) is a technique for system modelling. OO is used to model a system as a set of interacting objects that in one way or another are related. Object model of a system depends upon what a system designer wishes to represent in a system. Interest in the object-oriented method has grown rapidly over the last few years. This is mainly due to the fact that it has shown many good qualities. Amongst the most prominent qualities of a system designed with an object oriented method are as described below (Jacobson, Christerson, Jonson and Wergaard, 1992,p.43).

- Understanding of the system is easier as the semantic gap between the system and reality is small.
- Modifications to the model tend to be local as they often result from an individual item, which is represented by a single object.

Some of the main concepts associated with object orientation are as described below.

Object:

Microsoft computer dictionary defines object as a variable comprising both routines and data that are treated as a discrete entity. Data encapsulated in a object represents state of an object at a given time. Routines associated with an object are invoked to manipulate state of an object.

An object is created from a template known as 'Class". A class defines internal structure of an object. A class is sometimes called as object's type. A class is a compile time concept while an object is a run-time concept.

An object is an abstract representation of an entity defined in a system. Member data variables and methods or routines could be accessible to the owner of the object or remain private within an instance of an object. This is known as information hiding.

A class may be defined as a descendant of one or more classes. This is known as inheritance. The class that inherits another class is sometimes known as derived class and inherited class is known as base class. A class may consist of aggregation of various other classes instead of inheriting them. Derived class has access to members of base class as determined by a programming language.

A class may inherit from more than one class. This is known as multiple inheritance. There are several advantages and disadvantages associated with single and multiple inheritance and they are highly debatable subjects. There is good amount of literature present which discusses these issues at length. However we are not addressing these issues in our research.

Polymorphism:

A method or a routine in a class may support polymorphism in two ways. They are 'Parametric Polymorphism' and 'Dynamic Binding'.

Parametric polymorphism means that a class implements a method that operates on a general data-type. A parameter of any type can be passed as an argument to this method and the method is able to handle the situation. This mechanism frees the designer from writing several similar routines that perform same operation on different data-types, ie, sorting integers, structures, strings, etc (Jacobson, Christerson, Jonson and Wergaard, 1992,p.49).

Dynamic binding means that a class may implement several routines of same name that may or may not require same number or type of arguments. At run-time, depending upon the types parameters passed to the routine, correct routine is selected and executed (Jacobson, Christerson, Jonson and Wergaard, 1992,p.49).

Document / View architecture

Microsoft has developed Microsoft Foundation Class (MFC) library for development under MS Windows. Using MFC for development frees a developer to spend more time developing structural components of a program and less time worrying about minute details of Windows API. MFC attempts to simplify Windows development process (Prosise, 1995,p.18).

MFC is a hierarchy of about 130 classes. MFC is also an application framework. MFC helps to define the structure of an application and handles much of routine functionality on the application's behalf. *CWinApp* is the class that represents the application itself. The framework supplies most of standard code to support a windows based application. MFC also supports *Document / View* paradigm which allows a program's data to be separated from graphical representations of that data.

Most MFC classes fall into one of these six categories.

- *CObject*
- *Application architecture*
- *Visual Objects*
- *OLE2*
- *Database*
- *General purpose* (Prosise, 1995,p.18).

Document / View classes fall under the category of application architecture classes. The application architecture classes help define the form and structure of an MFC application. *Document / View* paradigm allows abstract representation of program's data in a class derived from *CDocument* class. Document draws a clear boundary between how data is stored and how it is represented on screen. Role of view class is to render document class on the screen and translate user actions into commands that manipulate document object (Prosise, 1995,p.21). The SPORDAC prototype utilises *Document/View* architecture in its implementation.

BIBLIOGRAPHY

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.

Bac,F., Perov,V. (1993). New evolutionary genetic algorithms for NP complete combinatorial optimisation problems. *Biological cybernetics, 69(3).* 229-234.

Battista Di, G., Eades, P., Tamassia, R., and Tollis, I. (June 1994). Algorithms for drawing graphs: An annotated bibliography. *Available: ftp.wilma.cs.brown.edu/pub/papers/compgeo/gdbiblio.tex.Z.*

Bland, J., and Dawson, G. (Sept. 1994). Large-scale layout of facilities using a heuristic hybrid algorithm. *Applied mathematical modelling, 18(9).* 500-503.

Bloesch, A. (August, 1993). Aesthetic layout of generalised trees. *Software-practice and experience, 23(8).* 817-827.

Booch, G. (Feb., 1994). Designing an application framework. *Dr. Dobb's Journal, 19(2)*. 24-31.

Bozer, Y., Meller, R., Erelbacher, S. (July 1994). An improvement type layout algorithm for single and multiple floor facilities. *Management Science 40(7).* 918-932.

Dengler, E., Friedell, M., and Marks,J. (1993). Constraint-driven diagram layout. *Proceedings of 1993 IEEE Symposium on visual languages*. 330-335.

Dodson,D. (1995, September). COMAIDE: Information visualisation using cooperative 3D diagram layout. Paper presented at the proceedings of Graph Drawing '95, Passau, Germany [on-line]. Available http://web.cs.city.ac.uk/research/dig/digpapers.html.

Eades,P. (1984). A heuristic for graph drawing. *Congressus numerantium, 42*. 149-160.

Eades, P. (Aug. 1991). Drawing free trees. *Technical report IIAS-RR-91-17E, International institute for advance study of social informaion science, Fujitsu Limited*. 1-29.

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

Genetic Algorithms (Lectures 1-3). (no date). [Handout]. (Available from Edith Cowan University, Mount Lawley, 6050, Western Australia)

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.

Izumoto, H., Wakabayashi, S., Miyao, J., and Yoshida, N. (1990). A Placement method for size restricted blocks in VLSI layout design. *Electronics and communications in Japan, Part 3, 73(9).* 86-96.

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

Jacobson, I., Christerson, M., Jonson, P., Wergaard, G. (1992). Object oriented software engineering: A Use case driven approach. ACM Press. USA(4th edition). pp.43-60.

Joy, M., and Smith,V. (1995). NP Completeness of a combinatory optimisation problem. Notre Dame journal of formal logic, 36(2). 319-335.

Lai,W. (1993). Building interactive diagram applications. PhD thesis, Department of Computer Science, University of Newcastle. pp. 91-120.

Lai, W., Liu, Y., and Millar, J. (1995). Designing Tree Structure Diagrams for Management. Proceedings of 1995 IEEE annual International Engineering Management Conference (pp. 358-363). Singapore.

Liao, Y., Wong, C. (1983). An algorithm to compact a VLSI symbolic layout with mixed constraints. IEEE Transactions on computer aided design of integrated circuits and systems, CAD-2(2). 62-69.

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.

Luders,P., and Ernst,R. (1993). Automatic display layout in window oriented user interfaces. Interfaces in industrial systems for production and engineering, 8(10). 27-42.

Luders,P., and Ernst,R. (1994). The dynamic screen – Beyond the limits of traditional graphic user interfaces. *13th IFIP World computer congress, 8(10)*. 27-41.

Luders,P., and Ernst,R. (1995). Improving browsing in information by the automatic display layout. *IEEE Symposium on information visualisaion* (pp.26-33). Atlanta:USA.

Luders,P., and Ernst,R. (1995). Improvement of the user interface of multimedia applications by automatic display layout. *Multimedia and networking conference* (pp.54-64). San Jose:USA.

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.

Mangano, S. (April 1994). Algorithms for directed graphs. *Dr. Dobb's journal, 19(4)*. 92-147.

Matsui, T. (1993). A method for two-dimensional layout of semantic structure of graphs. *Electronics and communication in Japan, 76(3)*. 1-12.

Messinger, E., Rowe, L., and Henry, R. (Feb. 1991). A Divide and conquer algorithm for the automatic layout of large directed graphs. *IEEE Transactions on systems, man and cybernetics, 21(1)*. 1-12.

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.

Nakamura, T. (1990). A multiple-window display. *Systems and computers in Japan, 22(7)*. 1-11.

Noik, E. (1993). Layout-independent fisheye views of nested graphs. *Proceeedings of 1993 IEEE Symposium on visual languages*. 336-341.

Okada, K., Onodera, H., and Tamaru, K. (1995). Compaction with shape optimisation and its application to layout recycling. *IEICE Transaction on fundamentals of electonic*, 78(2). 169-176.

Pleeger, C.P. (1989). Security in Computing. Englewood Cliffs, New Jersey: Prentice-Hall.

Prosise, J., (June 1995). Wake Up and Smell the MFC: Using the Visual C++ Classes and Application Framework. *Microsoft Systems Journal*., 17-34.

Prosise, J., (July 1995). Programming Windows 95 with MFC, Part II: Working with Display Contexts, Pens, and Brushes. *Microsoft Systems Journal*., 39-63.

Prosise, J., (Aug. 1995). Programming Windows 95 with MFC, Part III: Processing Mouse Input. *Microsoft Systems Journal*., 57-74.

Prosise, J., (Sept. 1995). Programming Windows 95 with MFC, Part IV: Contending with the Keyboard. *Microsoft Systems Journal*., 35-50.

Prosise, J., (Nov. 1995). Programming Windows 95 with MFC, Part V: Menus, Toolbars, and Status Bars. *Microsoft Systems Journal*., 41-58.

Prosise, J., (Dec. 1995). Programming Windows 95 with MFC, Part VI: Dialog Boxes, Property Sheets, and Controls. *Microsoft Systems Journal*., 53-72.

Prosise, J., (Feb. 1996). Programming Windows 95 with MFC, Part VII: The Document/View Architecture. *Microsoft Systems Journal*., 19-40.

Proth, J., and Souilah, A. (1992). Near optimal layout algorithm based on simulated annealing. *International journal of systems automation: Research and applications, 2*. 227-243.

Rebaudengo, M., and Reorda, M. (August 1996). GALLO: A genetic algorithm for floorplan area optimisation. *IEEE Transaction on computer aided design of integrated circuits and systems, 15(8)*. 943-951.

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.

Shih, L., Enkawa, T., and Itoh, K. (1992). An AI search technique-based layout planning method. *International journal of production research, 30(12).* 2839-2855.

Shin,H., Sangiovanni-Vincentelli, A. (1990). Zone-Refining techniques for IC layout compaction. *IEEE Transactions on computer-aided design, 9(2).* 167-179.

Storey, M., Muller, H. (1995). Graph layout adjustment strategies. Unpublished manuscript, Simon Fraser University, Canada.

Storey, M., Muller, H. (1995). Manipulating and documenting software structures using SHriMP views. Unpublished manuscript, Simon Fraser University, Canada.

Sugai, Y., and Hirata, H. (1991). Hierarchical algorithm for a partition problem using simulated annealing: application to placement in VLSI layout. *International journal of systems science, 22(2).* 2471-2487.

Supowit, K., and Reingold, E. (1983). The complexity of drawing trees nicely. *Acta informatica, 18*. 377-392.

Suzuki, A., Fuchino, T., Muraki, M. and Hayakawa, T., (April 01, 1991). An evolutionary method of arranging the plot plan for process plant layout. *Journal of chemical engineering of Japan, 24(2)*. 226-231.

Tamassia, R., Battista, G., and Batini, C. (1988). Automatic graph drawing and readability of diagrams. *IEEE Transactions on systems, man and cybernetics, 18(1).* 61-79.

Tsuchida, K. (1995). The complexity of drawing tree structured diagrams. *IECE Trans. Inf. & Syst., 78(7).* 901-908.

Wang, M., Liu, C., and Pan,Y. (1991). Computer aided panel layout using a multi-criteria heuristic algorithm. *International journal of production research, 29(6).* 1215-1233.

Yaman, R., Gethin, D., and Clarke, M. (1993). An effective sorting method for facility layout construction. *International journal of production research, 31(2).* 413-427.