Human Pattern Nesting Strategies in a Genetic Algorithms Framework

by

Rahul Dighe

M.S., Mechanical Engineering,
Massachusetts Institute of Technology, 1992

B. Tech., Mechanical Engineering,
Indian Institute of Technology, Bombay, 1988

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

Doctor of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1995

Signature of Author: _____
Department of Mechanical Engineering
May 10, 1995

Certified by: _____
Mark J. Jakiela
Associate Professor
Department of Mechanical Engineering
Thesis Committee Chair

Accepted by: _____
Ain Sonin
Chairman, Departmental Graduate Committee

Human Pattern Nesting Strategies in a Genetic Algorithms Framework

by

Rahul Dighe

Submitted to the Department of Mechanical Engineering
on May 10, 1995 in partial fulfillment of the
requirements for the degree of
Doctor of Science in Mechanical Engineering

Abstract

Pattern nesting refers to the process of arranging a set of predefined shapes to occupy the least amount of space. This thesis discusses an automated pattern nesting system developed for the layout of complex, polygonal, two-dimensional shapes under translational constraints on a roll of fixed width such that the length of material used is minimized. A hierarchical representation that separates numerical and symbolic search is developed. This representation avoids the creation of invalid layouts and facilitates genetic search. Human pattern nesting strategies are incorporated in the probabilistic framework of genetic algorithms. Models of human intuition are developed to identify and evaluate key part locations for the numerical search. Intuitive justification for the identification of key locations for the placement of parts in a layout and a model of human intuition to quantify area waste created between part boundaries is developed. These models are used to effectively restrict the search to a finite set of solutions. The probabilistic framework of genetic algorithms is used for the symbolic search. The genetic search is augmented by the incorporation of human search reduction strategies.

Thesis Supervisor:

Prof. Mark J. Jakiela
Title: Associate Professor, Department of Mechanical Engineering

Thesis Committee:

Chairman:
Prof. Mark J. Jakiela, Department of Mechanical Engineering.

Members
Prof. David Gossard, Department of Mechanical Engineering
Prof. Kevin Otto, Department of Mechanical Engineering.
Dr. Joe Marks, Mitsubishi Electric Research Laboratories.

Dedication

To my parents: Rajani and Prafulla Dighe

Acknowledgments

I wish to thank my advisor, Prof. Mark Jakiela, for his support, encouragement and guidance throughout my stay here. I hope this work has done justice to his patience and faith.

I also wish to thank other committee members, Prof. David Gossard, Prof. Kevin Otto and Dr. Joe Marks for their support and insightful comments.

I would like to take this opportunity to thank a few of my many friends who have contributed to this work: Rakesh Gupta, Jayaraman Krishnasamy and Kazuhiro Saitou.

I am grateful to our sponsors, Ford Motor Company for the financial support for this project. Also, the discussions with pattern nesting expert Guy D'Ana have contributed a great deal to some of the main concepts in this work, and I am thankful to him for spending time with me.

The GA library used for work has been implemented by my officemate Kazuhiro Saitou. I am grateful to him for his patience in making the library suitable for this work.

Table of Contents

List of figures

Chapter 1

Introduction

1.1 Automated Pattern Nesting

Pattern nesting refers to the process of arranging a set of objects in order to occupy the least amount of space. Most human beings deal with this problem very often during their day-to-day activities: arranging books on shelves, packing suitcases, arranging containers in refrigerators, to name a few instances. Often, there are restrictions on the way the objects can be arranged and the overall space they can occupy. The total space available in a refrigerator is limited. In addition, placement of containers in a refrigerator is restricted to the shelves inside. Access to the containers is restricted to one side and the containers themselves have certain restrictions on the way they can be placed. The arrangement of books and other objects on shelves has similar restrictions. In addition, aesthetics and ergonomics play an important role in restricting the placement of objects. For most of these activities, human beings use trial-and-error driven by their intuition and understanding of the shapes of the objects to be arranged. In most cases, finding the best arrangement possible is not necessary. Often, the definition of best is not very clear due to the nature of requirements; e.g. comparison of aesthetic considerations with space limitations. Also, the arrangements are changed frequently and this limits the amount of effort that can be spent each time. However, commercial applications of this problem justify significant efforts in finding better and better arrangements of objects. For mass manufactured products such as automobiles and stereos, a good arrangement can be worth millions to the manufacturer. For industries such as packaging and shipping, it is important to pack as much cargo as possible in the available space. A good arrangement of cargo can reduce the cost of shipping significantly.

Automation of repetitive tasks has always been of interest to human beings. Since computers came into being, there has been significant interest in automating the process of

pattern nesting. The process of developing a good layout is not yet fully understood, however, and the automated pattern nesting systems developed so far do not match the performance of human experts. While it is not known whether human experts always find the best possible layouts, most pattern nesting methods being developed attempt to match human performance.

1.1.1 Forms in various dimensions and applications

The problem of placing shapes in order to occupy the least amount of space occurs in different forms in different dimensions:

One-dimensional form: A set of one dimensional objects is given along with a set of bins of the same size. The objective is to place the objects in the bins in order to occupy the least number of bins (see figure 1.1.1.1). This is known as one-dimensional bin-packing or simply, bin packing. An application of this problem is for cutting pieces of stock from available bars of a fixed length. The objective is to minimize the number of bars used, while cutting all the pieces of required sizes.

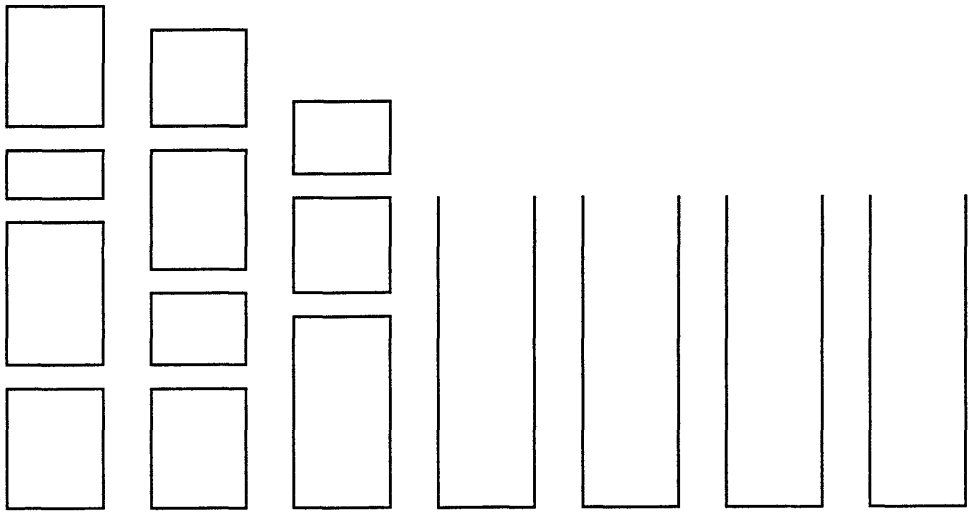
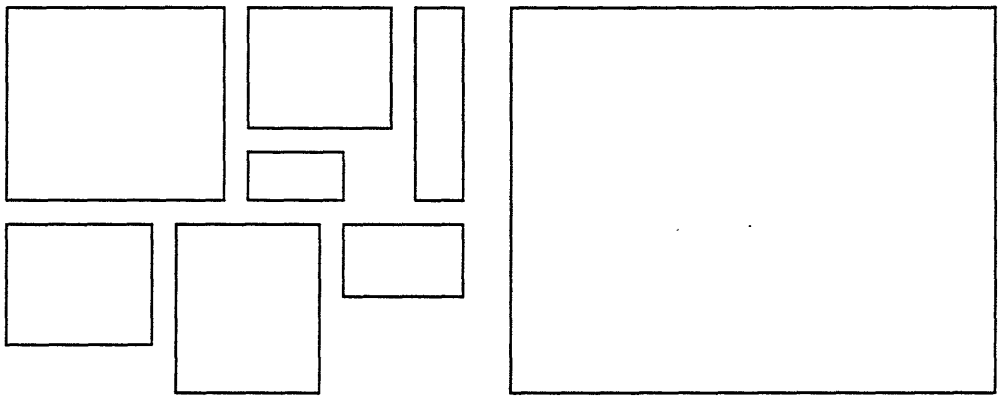


Figure 1.1.1.1 One-dimensional bin-packing

Two-dimensional form: A set of two-dimensional objects is to be placed on a plane with the objective of minimizing the total area occupied. Often, a set of rectangular shapes is to be placed in a given rectangle, as illustrated in figure 1.1.1.2. This problem is known



Rectangles to be placed on blank Blank available for placement

Figure 1.1.1.2 Two-dimensional bin-packing of rectangles

as two-dimensional bin-packing. For VLSI layout (see figure 1.1.1.3), shapes to be placed

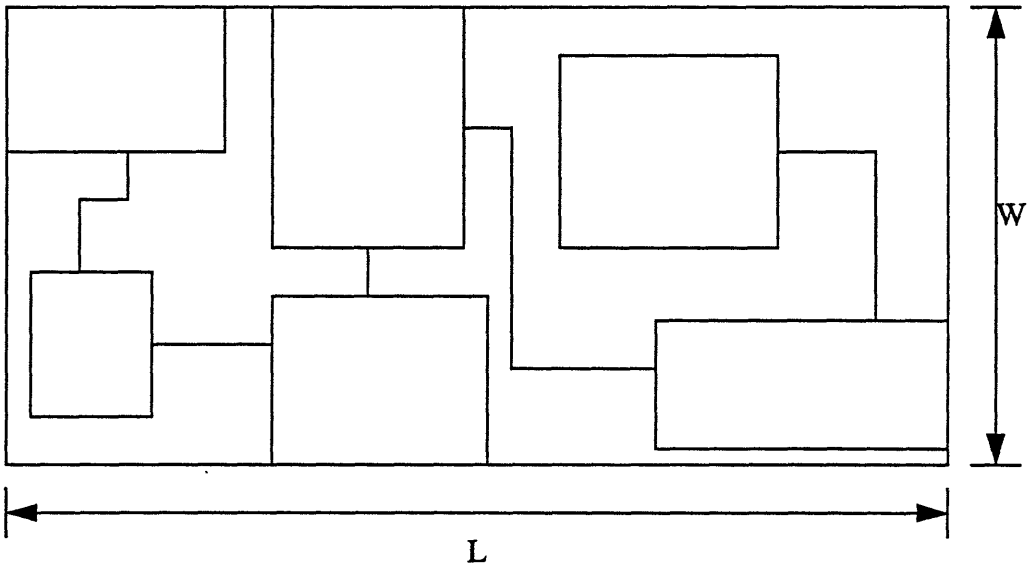
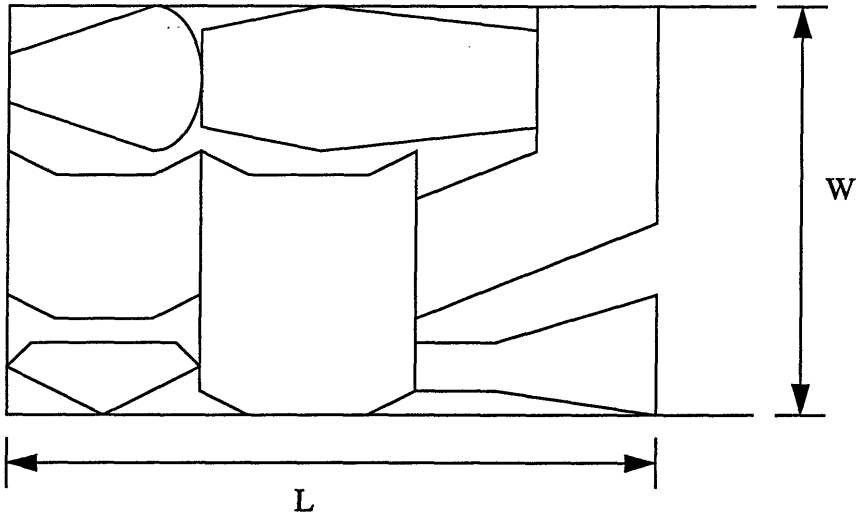


Figure 1.1.1.3 VLSI layout: rectangular shapes with wire routings.
Both length and width can be varied to minimize
the area of the total rectangular enclosure

are rectangular and have constraints on the placement of those shapes that arise from wire

routings between these rectangles. Pattern nesting refers to be problem of placing a set of free-form shapes in a rectangular area. This has applications in the garment, ship-building and sheet-metal industries. Often, these shapes are to be cut from a roll of material of constant width. The objective in that case is to use the shortest length of roll possible (see figure 1.1.1.4). In the ship-building industry, the shapes often occur in mirror images of each other. In the garment industry, the shapes can usually be oriented only along certain direc-



W = Width of blank L = Length of layout

Figure 1.1.1.4 Pattern nesting for free form parts.
Width of material is fixed, length is minimized.

tions, to ensure matching patterns after sewing. The layout of sheet metal parts is usually free of orientation and placement constraints.

Three-dimensional form: A set of three-dimensional shapes is to be placed in the smallest enclosure. Most often, the enclosure is pre-determined; the objective is to fit all the given shapes inside. For shipping of cargo, the shapes are usually simple, e.g. rectangular boxes. Also, there are few restrictions on the placement and orientation of shapes. The placement of the mechanical assembly under the hood of an automobile requires dealing with complex shapes and a variety of functional constraints on the location and orientation of parts that arise from requirements such as easy access to certain parts, prevention

of damage due to the heat generated, kinematic and functional connections, feasibility of assembly, etc. (Jay Kim, 1987). Design of consumer product housings (Wallace and Jakiela, 1993; Dighe and Jakiela, 1994) is often based on aesthetic, functional and manufacturing considerations. The placement of functional parts inside the housings is done after the enclosure has been determined. In that case, the location and orientation of functional parts inside is constrained by the placement of parts externally accessible, e.g. the placement of buttons, speakers, etc. in a stereo restricts the placement and orientation of the electronic parts inside.

1.1.2 Intractability of the problem

The problem of optimally arranging shapes is known to be computationally very expensive. The solution space grows exponentially with the number of shapes. No algorithms for optimal shape nesting are known. In fact, the problem of bin-packing is known to be NP-hard (see Garey and Johnson, 1979), meaning that no algorithm is available that can be guaranteed to solve such problems in exactly polynomial time; unless P = NP, only an exponential-time search can guarantee the optimality of a solution. Such an exponential time search, of course, can be carried out only for a trivially small number of shapes. The other variants of this problem can be converted into the bin-packing problem (see Zhenyu Li, 1994). The search for the optimum is made more difficult by a large number of invalid layouts due to overlap of shapes. In one dimension, the placement of one part relative to another without causing overlap is trivial; simple translation of parts relative to each other is sufficient. This process cannot be generalized easily in higher dimensions, e.g., in two dimensions, the parts each have three degrees of freedom. The choice of locations x, y, θ for each part without causing overlap is not obvious; a variety of combinations of the location and orientation variables can lead to an invalid arrangement. Thus, when a large number of parts are to be placed, the number of valid layouts are few. The search for a global optimum suffers from two conflicting goals: area minimization and overlap minimization. The same problem exists in three-dimensions where each part has six degrees of freedom.

1.2 Thesis goals

This thesis focuses on the problem of automated pattern nesting of two-dimensional shapes. As mentioned earlier, this variant of the problem has applications in the ship-building, garment and sheet-metal industry. The material used in these applications is usually very expensive, and for the garment and sheet-metal industry, the parts to be nested are used in mass-manufactured products. Thus, an extra one percent utilization in a layout can result in savings of hundreds of thousands to millions of dollars. The shapes of parts to be nested are complex, non-convex, and sometimes multiply-connected. The shapes are to be cut from a roll of material, enforcing a constraint on the width of the layout. Often, the shapes can be oriented only along a few directions. Besides these two constraints, there are no restrictions on the placement and orientation of the shapes on the blank of material. This makes the understanding and application of complex geometric aspects of the problem critical to obtaining good layouts. This is different from the one-dimensional form of the problem, where shape geometry does not play a role in the optimization process, and from the three-dimensional form, where functional constraints on the placement and orientation of the parts limit the role of the geometric aspects in the optimization process.

The goal of this research has been to develop an automated pattern nesting system that can be practically used for the layout of two-dimensional, complex shapes in an industrial environment. Human experts routinely perform better than most computer systems reported so far. Keeping this in mind, the aim has been to match human performance most of the time.

1.3 Contributions

An automated pattern nesting system for layout of two-dimensional parts has been developed. Non-convex, two-dimensional, polygonal shapes can be nested on a blank of material using this system. Genetic algorithms (see Goldberg, 1989) are used as the basic framework for optimization. The main contributions of this thesis are the development of a hierarchical representation for the layout of general, two-dimensional, complex shapes and the incorporation of human strategies in a hierarchical genetic search.

1.3.1 Hierarchical representation

Layout creation can be modeled as a sequential or a hierarchical process. Parts can be placed one at a time, at a good location, and a good sequence of placement of parts can be found. The hierarchical representation discussed in chapter 6 is based on this process. The search for good locations for a single part or a group of parts is separated from the order in which parts are placed or grouped. Such a hierarchical representation is different from a totality representation in which placement of all parts is considered simultaneously, and can be considered to be a generalization of the symbolic representation commonly used for the layout of rectangles.

1.3.2 Human strategies

For the placement of a single part on a blank, only a few locations, termed corner points, need be considered out of all valid locations possible. An intuitive explanation for the focus of attention on these corner points is developed. A simple method based on force equilibrium to identify these corner points is proposed. By using this method, the geometric aspect of the search can be made deterministic and trivial. The search for good solutions can be restricted to a purely symbolic search: the best sequence in which shapes are placed on the blank along with the best choice of individual corner points. The problem is still NP-hard, but the role of the geometric aspects of the problem in the search can be eliminated.

Human experts are able to identify what can be termed "good" locations for shape placement. By placing a shape on a blank, voids are created between shapes. Identification of locations where such voids are reduced seems obvious to humans. However, description of such locations via a computer algorithm is a difficult task. A model has been developed to quantify the waste area that arises from the voids created between shape boundaries. A weighted sum of areas trapped between part boundaries is obtained based on the distance of each area element from the boundary of the part being placed. A single quantity capturing the waste between part boundaries can be used effectively to identify good locations for part placement. This model has been developed by observing human experts at work.

Human experts employ several strategies to efficiently create good layouts. Often, large parts are placed first and the smaller ones are placed in the voids created between them. Closely packed subgroups of parts in a layout are frequently retained in the subsequent layouts. Parts are exchanged within a layout to obtain shorter layouts or sometimes, to create space for movement of other parts. Often, layouts are modified based on their length; few parts are moved around if the layout efficiency is very close to the expected efficiency and vice-versa. In the system described here, such strategies are incorporated using the basic framework of genetic algorithms as a probabilistic optimization method. None of the strategies alone is likely to produce very good layouts. For this reason, the strategies are used probabilistically to maintain the possibility of finding the global optima. Also, the calculation of probabilities to invoke any strategy is based on simple quantities such as area, length, etc. No complex geometric reasoning is used, since such reasoning could lead to domain specific search and results. These strategies help identify good points in the solution space efficiently. This is crucial for a practical, automated system useful for industry.

1.4 Organization of the thesis

This thesis is organized as follows.

Chapter 2 provides a brief introduction to genetic algorithms. The motivation and philosophy underlying genetic algorithms are explained based on the theory of natural selection. The basic steps involved in using a simple genetic algorithm are explained along with the reasoning behind these steps. Two main principles important to the effectiveness of genetic algorithms, the schema theorem and the building block hypothesis, are discussed briefly. Chapter 5 and chapter 6 will revisit these principles in developing effective representations for pattern nesting. A comparison of genetic algorithms with other optimization methods is made to provide more insight into the use of genetic algorithms.

Work of others related to this area is surveyed in chapter 3. The area of pattern nesting has been of interest for over thirty years and the literature abounds with related efforts. The work closest to this thesis is reviewed in this chapter and is divided into three

main categories: mathematical programming methods, heuristics-based methods and two probabilistic methods: genetic algorithms and simulated annealing.

Chapter 4 describes different part placement and layout evaluation methods. Three methods of part placement for pattern nesting are discussed along with their relative merits: symbolic placement, naive placement (based on variables x, y, θ) and assembly-based placement. Two layout evaluation methods are also discussed in chapter 4. Overlap area calculation is the most commonly used method. However, it is computationally expensive and some of the researchers use approximate estimates of overlap area. A new, efficient method of layout evaluation is developed, based on the placement of seeds representative of parts and the extent of maximum possible growth of these seeds.

Chapter 5 discusses are termed as totality approaches for pattern nesting. The placement of all the parts is considered together in this approach. The disadvantages of using a naive representation for part placement are illustrated by the examples. The examples also illustrate the computational effectiveness of evaluation of layouts based on growth of shapes from representative seeds as against the calculation of overlap. An assembly based placement of can be used in a totality approach by making the assembly sequence of parts a part of the chromosome. This chapter also illustrates the use of repeated speciation to improve the search potential of a totality approach.

A hierarchical approach developed for pattern nesting is discussed in chapter 6. In this approach, instead of considering all parts at the same time, individual parts or pairs of parts are treated one at a time. Two levels of optimization are used, each using a different genetic algorithm as the search method. Numerical optimization is used at the lower level to locate pairs of parts relative to each other, while the higher level search tries to find the optimal symbolic arrangement of the shapes to be nested. An assembly based placement approach that eliminates the problem of overlap is used at the lower level. Two representations are discussed; a binary tree based representation for the layout of shapes in a rectangular enclosure and a string based representation for the layout of shapes on a blank of material of given width. Modified crossover operators are used for both representations.

Several different evaluation functions are discussed with illustrative examples.

Chapter 7 describes the incorporation of human strategies in the probabilistic framework of genetic algorithms. These strategies are focused on the problem of pattern nesting on a blank of fixed width. Certain key locations on the layout considered by human experts for the placement of a single part can be identified as corners. The process of finding these locations with an intuitive justification is developed. A model of waste area created during the placement of individual parts in a layout is described. This model has developed based on observation of human experts at work, and can be used to quantify the potential detriment of voids between part boundaries in a very general manner. In the remainder of the chapter, strategies used by human experts to effectively search for good sequences in which the parts can be placed in the layout are discussed. These strategies are used probabilistically in order to maintain the possibility of finding the global optima.

Conclusions from this work are presented in chapter 8. Also, possible extensions of this work are discussed.

Chapter 2

Genetic Algorithms

2.1 Origin and Philosophy

Genetic algorithms, developed by John Holland (1975), are based on the theory of natural selection. Over a period of time (and generations), species adapted themselves to better suit their surroundings. The theory of natural selection explains this process of evolution and adaptation based on the "survival of the fittest" phenomenon. Individuals with characteristics that were suitable for survival in their environment had higher life expectancy and thus reproduced more. This led to a proliferation of their suitable characteristics in the subsequent generations. Individuals became "fitter" for survival as generations progressed due to the process of combination of the characteristics of parents during reproduction. On the other hand, those individuals without such characteristics did not survive long and reproduced less. Thus, the instances of their characteristics in subsequent generations decreased in number, resulting in a smaller number of "unfit" individuals over generations. Genetic algorithms mimic this process of evolution and adaptation for optimization.

Characteristics of individuals depend on their genetic material. These characteristics are represented in their chromosome. The genetic material of an offspring results from a combination of the chromosomes of parents. This recombination occurs via what is known as a "crossover" of genetic material. This is accompanied by certain infrequent, random changes in the chromosome called "mutations". Genetic algorithms utilize the two processes of crossover and mutation along with "reproduction" to produce offspring from parents while exploring the solution space.

2.2 Optimization using Genetic Algorithms

A typical genetic algorithm for optimization works very much like the process of evolution described above (see Goldberg, 1989). A population of individuals is created with randomly chosen characteristics. Each individual is represented by a chromosome. A typical chromosome representation is a string of binary numbers as shown in figure 2.2.1.

1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 0 1 0 0 0 1 1 0 1 1

Figure 2.2.1 Chromosome of binary numbers

The optimization variables are mapped to this string of binary numbers. Thus, the string contains information that when decoded in a certain fixed manner, characterizes each individual. This is illustrated in figure 2.2.2, where the two variables that govern the shape and size of the rectangle, length and height, can be decoded from the string of binary numbers.

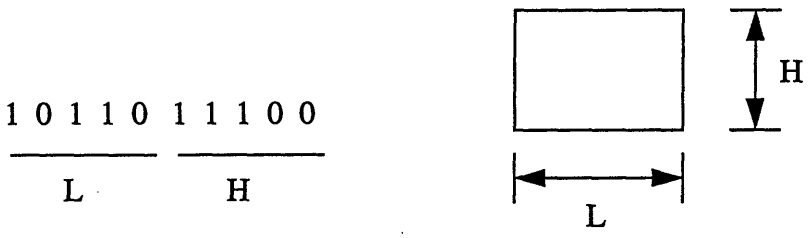


Figure 2.2.2 Dimensions of rectangle mapped to binary chromosome

A set of such individuals constitutes a population. A genetic algorithm begins with the creation of a population of individuals, called the initial population. The bits in the chromosomes of the initial population are set randomly to prevent bias towards any particular characteristics.

After the creation of the initial population, each individual is evaluated according to the optimization criterion and is assigned a "fitness" value. This fitness value quantifies the ability of the individual to survive in the environment, which is the domain of the optimization. The next generation of the population is created by choosing parents from the population. This process is termed "reproduction". The selection of parents is probabilisti-

cally based on their fitness value; individuals with higher fitness have a better chance of being selected for reproduction. The probability of their selection is often calculated as:

$P_{select} = F_i/\Sigma F_i$;

where,

F_i = Fitness of individual i.

ΣF_i = Total fitness of the population.

Such a selection strategy ensures that "fitter" individuals" reproduce more. The selection is made probabilistically since some individuals with lower fitness values do contain some useful characteristics. This selection strategy is known as Roulette-Wheel selection due to its probabilistic nature and the method of calculating the probability of selection of each individual for reproduction.

The selection of parents for reproduction occurs in pairs. The two selected individuals, known as parents, are mated to produce offspring via a crossover of genetic material. Figure 2.2.3 shows a typical crossover operation. A location along the chromosome is ran-

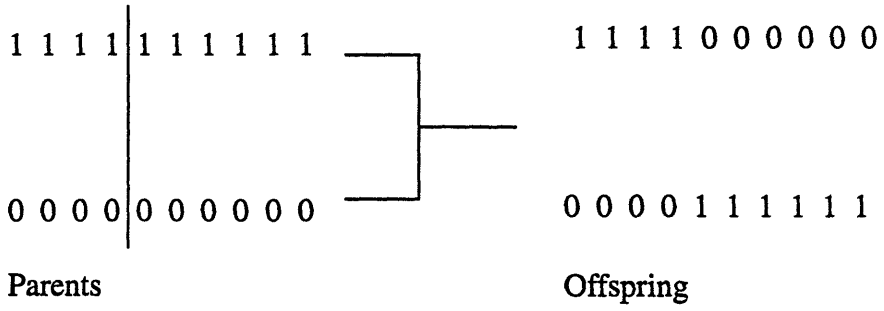


Figure 2.2.3 Single point crossover

domly chosen. The chromosome is divided into two parts at this location and the complementary parts of the chromosome from each parent are combined to produce offspring. This is known as a single point crossover. In this manner, the characteristics of parents are combined in their offspring. Crossover is not carried out on all parents. Sometimes, the parents are carried into the next generation without alteration. This prevents the algorithms from being overly destructive of existing genetic material. A certain probability

associated with the crossover operation, called the probability of crossover (P_{cross}), determines whether two parents will be mated via crossover. This probability is usually fixed at some value throughout the optimization procedure, although it can be varied as the search progresses.

The mutation operation simply flips randomly selected bits in the chromosome with a certain probability, known as the probability of mutation (P_{mut}). This is illustrated in figure 2.2.4. Each bit in the chromosome can be considered to contribute to a character-

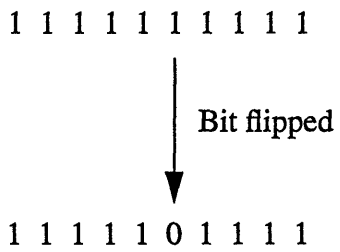


Figure 2.2.4 Mutation

istic of the individual. Since the values of bits in the chromosomes of the offspring are obtained from the values of the corresponding bits in one of the parents via reproduction and crossover, the characteristics of individuals in each generation are limited to the characteristics present in their parents. Since the parents themselves are selected from the randomly created initial population, certain desirable characteristics may not be obtained. Mutation ensures variety in the population and prevents premature convergence to suboptimal individuals. The random nature of mutation makes it destructive and hence, it is used sparingly. Similar to the probability of crossover, the probability of mutation is usually predetermined for the search.

The original population is then replaced by this newly created one. This process is repeated several times (generations) until some termination criterion is met, e.g. the average fitness or the fitness of the best individual does not improve in subsequent generations, or the number of generations exceeds some number. This optimization process can be

summarized in figure 2.2.5.

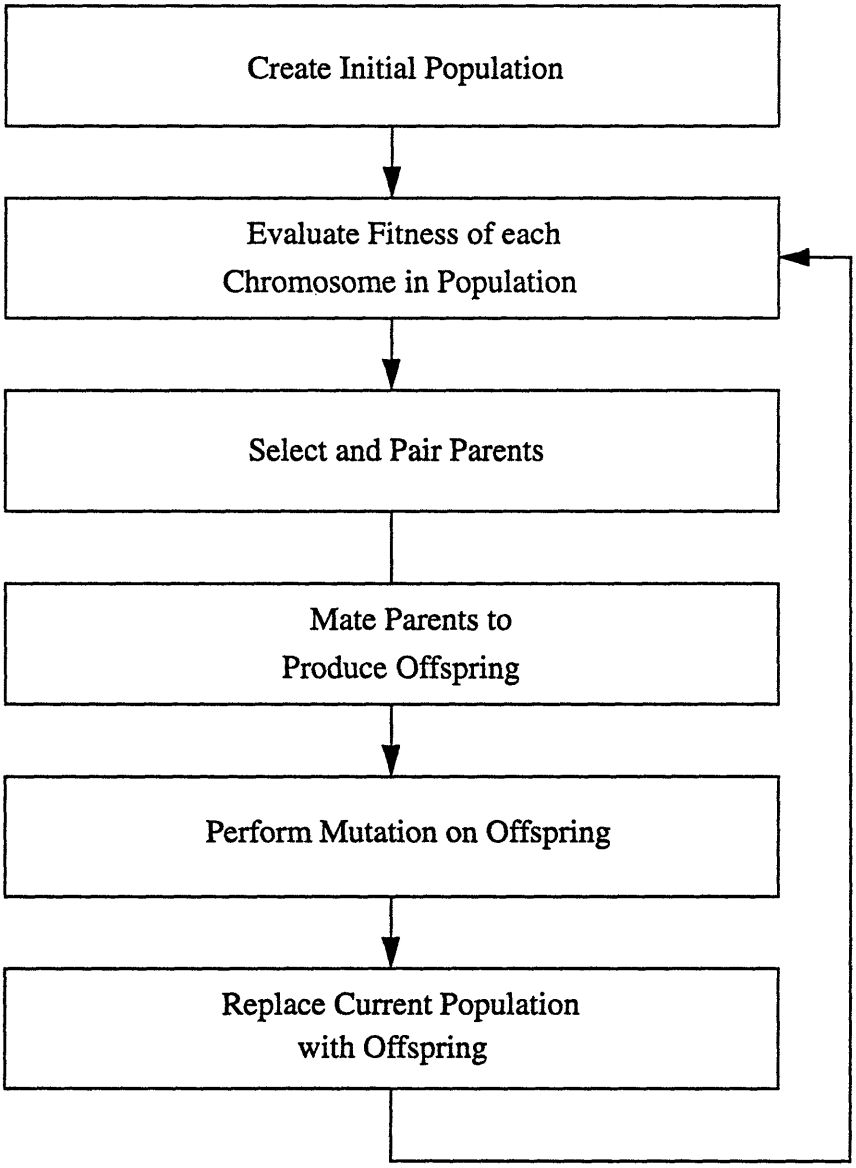


Figure 2.2.5 Flow of a Simple Genetic Algorithm

2.3 Basic Mechanism of Genetic Algorithms

To understand the efficiency of genetic algorithms and to exploit their potential, it is important to understand the schema theorem and the building block hypothesis. The schema theorem, as described by Goldberg (1989), explains the proliferation of the char-

acteristics of fitter individuals in a population. A schema in a chromosome can be thought of as a pattern or arrangement of bits. Consider, for example, the substring 1111 in the chromosome shown in figure 2.3.1. This substring forms a schema in the chromosome. It can be shown that 3^n such schemata exist in a binary coded chromosome of length n

1 1 1 1 0 0 1 0 1 0

Figure 2.3.1 Schema 1111 of length 4

(Goldberg, 1989). Due to the use of the selection strategy explained above, the number of instances of schemata associated with fitter individuals increases roughly exponentially with the number of generations (Goldberg, 1989). Each schema represents a particular characteristic of an individual, depending on the nature of the representation used. Thus, a growth in the schemata associated fitter individuals implicitly leads to a proliferation of their useful characteristics.

The building block hypothesis states that a combination of good building blocks can lead to better ones. Genetic algorithms operate on the basis of this hypothesis, assuming that good schemata can combine to produce better schema through the process of crossover of genetic material. Although studies done by Bethke (1981) support this hypothesis, it does not hold true for all application domains. Also, it may hold true for a particular representation of a given problem and not for some other representation of the same problem. The results of optimization of a set of variables can be different from those obtained by transforming those variables into a different set. Thus, while using genetic algorithms for optimization, it is important to use a chromosome representation for which combining building blocks from two good individuals via crossover produces better individuals.

2.4 Comparison with other optimization methods

Genetic algorithms differ from other methods of optimization in four ways (Goldberg, 1989).

1. GAs use a coding of the variables of optimization, not the variables themselves. As shown in figure 2.2.2, the variables that govern the shape of the rectangle are mapped to the string of binary numbers. Genetic algorithms operate on this string, searching for the optimal values of the binary numbers.

2. GAs search with a population of points, as against a single point, in the search space. The point to point methods of search can converge to a local optimum if the solution space is multi-modal (many-peaked). However, genetic algorithms operate on a set of points simultaneously (a population of strings), searching for many peaks in parallel. Thus, the probability of finding a false peak is reduced over methods that go from point to point.

3. GAs use only the objective function value, not any auxiliary information such as derivatives of the objective function. Many other optimization techniques require other information besides the objective function value to find an optimum. For example, gradient search techniques rely on derivatives, calculated either analytically or numerically, to reach the local minimum point corresponding to the current solution. Simulated annealing requires the definition of a neighboring operator to go from one solution to another. Such auxiliary information is not needed by genetic algorithms. They make use of a single crossover operator for all optimization problems for which the variables are encoded as chromosomes.

4. GAs use probabilistic transition rules. Deterministic rules work very well if they are based on facts about the domain of optimization. Otherwise, since they limit the search to certain regions of the solution space, using such rules could imply the potential loss of the global optimum. As an example, a possible deterministic selection strategy could be to select only a few of the best solutions from the population for reproduction. The roulette-wheel selection strategy used by genetic algorithms takes into consideration the possibility that a solution of lower fitness could have certain desirable characteristics, which when combined with certain other characteristics in another solution, could produce a good solution.

Chapter 3

Related Work

The nesting problem is of importance to many industries and automating the process of shape layout has been an active area of research since the advent of computers. Most efforts can be broadly classified into three categories: mathematical and dynamic programming, heuristics-based methods, and recently, probabilistic methods such as genetic algorithms and simulated annealing.

3.1 Mathematical and dynamic programming methods

The dynamic programming methods typically use moves of shapes to be nested along the axes and hence, are more appropriate for rectangular shapes than complex shapes. A review of such methods is presented by Sarin (1983).

Recently, a research group at Harvard University, under the direction of Victor Milenkovic, has reported work in pattern nesting using mathematical programming. They use two-dimensional configuration spaces to prevent overlaps between pairs of polygons. Li (1994) and Milenkovic et al. (1994) have developed translational compaction algorithms which, given a pattern layout, increase the nesting efficiency through continuous translation of the polygons. This is done by solving a sequence of linear programming models of the layout. After each model is solved, polygon positions are updated and then used to update the model. These compaction algorithms are very useful to gain an extra 2-3 percent efficiency from an existing layout. They can compact hundreds of polygons; in under one minute 150 polygons can be compacted on a 28 MHz SPARCstation (TM). They have also implemented powerful algorithms for rotational compaction.

Work done by Daniels and Milenkovic (1994) is focused on developing translational containment algorithms which are used to create a new layout. Given a container

and a set of polygons, the algorithms fit the polygons into the container, if a placement is feasible. The algorithms are also able to detect infeasible situations. Their general approach is to prune the two-dimensional configuration spaces, quickly check for a solution, and, if no solution is found yet, subdivide a configuration space and recurse. Some of their pruning and checking techniques use linear programming. One checking method uses linear programming to attempt to produce a nonoverlapping solution from an overlapping candidate solution. Their algorithms are practical for up to ten nonconvex polygons. The number of polygons to be placed in a completely new layout is large, and so they use a two-phase approach in which pieces are classified as "large" and "small". First the large pieces are placed. Then they use containment to build groups of small items which fit into the gaps determined by large neighboring pieces. Finally, they assign groups to gaps.

3.2 Heuristics methods

The basic strategy in these methods is to find a set of rules that work well for the specific application chosen. Examples of such applications are VLSI layout, ship-building, and clothing layout. Not surprisingly, these methods work well for the specific application and tend to break down if an unusual situation is encountered.

Albano and Sapuppo (1990) describe a branch-and-bound search method for cutting irregular shapes by placing each successive part at the leftmost and lowest location available on the blank. This search is further reduced by retaining only a certain number of alternatives at each stage in the allocation. Although they deal with irregular shapes, they limit their search to 180-degree rotation of the shapes. This restriction, along with the placement policy is quite limiting as large voids can be produced in the layouts. Also, a branch-and-bound strategy does not make use of the evaluation of one solution in order to choose other solutions to explore, as genetic algorithms do. Thus, it is not a directed search as genetic algorithms can be. Adamowicz and Albano (1976) describe a two stage approach: (1) clustering of shapes into well packed rectangles; and (2) layout of rectangles. Once the shapes are clustered into a rectangle, their relative positions are fixed and

this can obviate potentially good solutions. This clustering seems to work well for cases where shapes occur as mirror images of each other. Qu and Sanders (1987, 1989) have attempted to solve a variation of the problem: rectangular blanks of various sizes to be filled in an optimal manner by the shapes to be cut. They approach the problem in two stages. In the first stage, a given blank is filled such that scrap is minimized. In the second stage, the best sequence of blanks is chosen from the various blanks available. A set of heuristics is used for the first stage and the second stage reduces the vast search space by several methods such as backtracking. The first stage is limited by rectangular approximations of irregular shapes. In the ship-building industry, Cheok and Nee (1991) describe a system that uses a shape processor to classify shapes as floors, brackets, etc., and as "big" and "small". This is followed by a local optimization that finds the smallest rectangular enclosure of similar parts and places small shapes into the void areas of big shapes. Finally, the rectangles are arranged on the blank according to certain heuristics. The use of shape similarity can be rather limiting for general applications. Also, according to the authors, classification of shapes into few categories can lead to poor solutions if highly irregular shapes are present. The system developed by Cai et al. (1987) for the clothing industry is based on two sets of rules: layout rules and modification operators. Also, some interesting criteria for evaluation of layouts, such as flatness of the profile formed by the outermost shapes on the material stock, are used. Although modification operators provide a way of backtracking, they operate locally and the notion of global optimality is not addressed.

3.3 Probabilistic methods

Two probabilistic methods, genetic algorithms and simulated annealing, have recently generated significant interest. Earlier applications of these methods to pattern nesting have been limited to rectangular shapes. Fourman (1985) developed a symbolic layout system using genetic algorithms for compaction of VLSI layouts that takes into account connectivity constraints between rectangles. The chromosome uses labels such as "above" and "right-of" to describe the relative locations of shapes. Such a symbolic representation is not easily generalizable to n-sided polygons; the symbolic representation

relies on the fact that parts are rectangular and the edges of rectangles are oriented parallel to the principal axes. Smith (1985) describes a system that combines heuristic rules such as "Skyline pack" and "Slide pack" to pack rectangles into a given bin with a genetic search for the best order of the objects to be packed into the bin. His "order crossover" is, in our opinion, the best alternative for the usual single point crossover which can produce invalid configurations. Our efforts utilize this crossover with a more general, geometric interpretation that will be described in section 6.3. Kroger et al. (1990) have developed yet another symbolic representation for the layout of rectangles that locates rectangles above and to the right of other rectangles. They have also developed a recombination operator that produces valid offspring as an alternative to single point crossover. Again, generalization of this representation to n-sided polygons is not obvious. In a more recent effort, Reeves (1994) has reported attempts to combine heuristics with genetic search for one-dimensional bin-packing with encouraging results. Also, his idea of reducing the search space by removing objects that occupy a bin exactly is interesting although difficult to generalize to higher dimensions for n-sided shapes. A hybrid approach combining local optimization with genetic search for nesting general shapes has been developed recently by Fujita et al. (1993). The local minimization ensures good local packing whereas the genetic search based on "order" crossover tries to find the optimal order of the shapes. The local optimization requires significant computation of overlap of polygons. In addition, the violation of constraints must be penalized in combination with the search for the shortest length of material stock. The choice of weighting factors used for different terms in the evaluation function can be difficult and sensitive to the shapes. Although the local minimization tries to maintain the neighboring relationships between shapes derived from the string chromosome by using an additional term in the objective function, it appears that a single chromosome could map to various arrangement of shapes. This can prevent the genetic algorithm from searching the solution space efficiently.

The efforts using simulated annealing seem to have been fewer than those using genetic algorithms. The system described by Jain et al. (1990) is aimed at dealing with sheet metal layouts with general shapes. The examples shown, however, are restricted to a small number of shapes. Although this may not be a fundamental limitation, their

approach is based on computation of overlap and becomes impractical if a large number of shapes are present. Oliviera and Ferreira (1993) have recognized the problems associated with overlap computation and report two different approaches based on simulated annealing to deal with this issue. The first one uses discrete raster approximations of the blank and shapes to be laid out. The implementation described does not allow rotations of shapes and the layout quality seems to be restricted by this fact. The raster approximation of a shape is invariant under translations but not under rotations. This is a limitation if rotations of shapes are to be allowed in the implementation. The other implementation approximates the overlap areas by their rectangular enclosure. Such an approximation can possibly mislead the search depending on the nature of the overlap polygon. We will revisit the issue of approximation of overlap area in section 4.2.1

3.4 Summary of related work

The heuristics-based methods seem to suffer from a lack of generality in comparison to the probabilistic methods. The heuristic rules developed work well within certain problem domains and break down easily if an unusual problem is encountered. Augmentation of the heuristic search with a branch-and-bound search is a useful step towards generality, although it is not directed as a genetic algorithm can be.

The probabilistic methods do not suffer from these limitations. So far, the efforts using these methods are based mainly on naive representations of the problem that lead to many invalid (overlapping) configurations. This leads to extensive computation and forces the search to make a trade-off between minimizing area and minimizing overlap. The lack of a meaningful mapping to the physical problem domain tends to make modification operators such as crossover ineffective. Since the solution space grows exponentially with the number of shapes, the search becomes inefficient when the number of shapes to be nested becomes large. A representation and approach that completely eliminate the problem of overlap computation and that utilize the crossover operator in a meaningful manner are described in the chapter 6.

Chapter 4

Part Placement and Layout Evaluation

4.1 Part Placement methods

Part placement refers to the method and representation used for locating a part or a set of parts in a layout. Several different methods and representations have been developed by researchers in the area of pattern nesting. In this section, the two most commonly used methods, symbolic placement and naive placement, are discussed. Also, a new assembly-based placement method is described.

4.1.1 Symbolic placement

Consider the problem of placing a given set of objects in a given set of bins of equal size, with the objective of minimizing the total number of bins used. This is shown earlier in figure 1.1.1.1. In this one-dimensional case, the objects can simply be stacked above each other in the given bins. As shown in figure 4.1.1.1, object B is placed on top of object A in bin 1. The other bins are filled similarly, requiring 4 bins to place all of the objects. This method of arranging the objects can be captured in a symbolic representa-

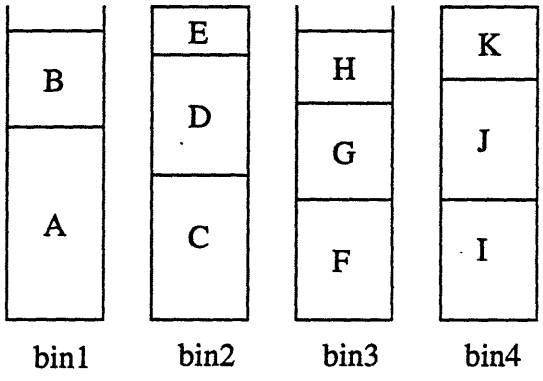


Figure 4.1.1.1 One dimensional object placement in bins

tion: B above A, D above C, etc. The bin used can be included in a modified representation: B above A in bin1, D above C in bin2, etc. The optimal arrangement can be found by searching over the possible different arrangements using such a symbolic representation. Such a representation is very intuitive and natural; there are no gaps created between objects in the bins.

For the two-dimensional problem of placing a given set of rectangles on a given set of rectangular blanks, a similar symbolic representation can be used. In addition, let us suppose that the rectangles cannot be rotated. Consider the placement of rectangle B on the blank shown in figure 4.1.1.2, when rectangle A has already been placed. The four sides of the blank are labeled boundaryL, boundaryR, boundaryB, boundaryT. The rectangle B can be placed to the right of rectangle A, or above it. To the right of A, the rectangle can be placed such that it borders the edge of the blank, i.e., it is above the lower edge (boundaryB). Thus, location B1 can be described symbolically as: right-of A, above boundaryB. Similarly, the location B2 can be described as: above A, right-of boundaryL. This representation and method of placement is simple as long as the rectangles do not overlap at the chosen locations. Consider placing an additional rectangle C after choosing the lower of the two locations B1 for rectangle B. The candidate locations for the placement can be found by considering pairs of available edges in the existing layout: above A, right-of boundaryL (C1); above A, left-of B (C2); right-of B, above boundaryB (C3). Notice that the locations C1 and C2 cause overlap of rectangles; only location C3 is valid. Thus, a simple extension of the symbolic representation used in the one dimensional case needs to be modified to account for the problem of overlap; we need to select one location from all the valid locations. If more than one valid location were available for the placement of rectangle C, we would apply the same criterion as that used for the placement of rectangle B. This strategy for placement of objects can be described in the following steps:

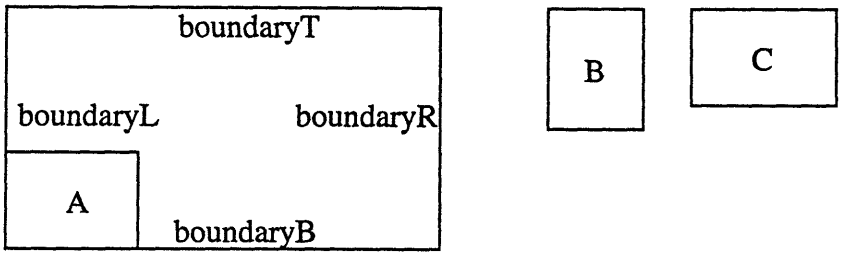
i. Find all candidate locations by forming pairs of edges existing on the layout.

ii. Identify all valid locations from the possible locations.

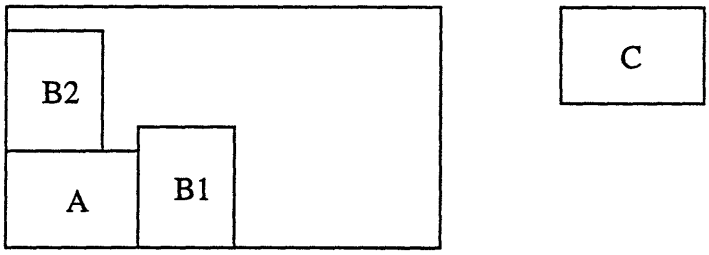
iii. Place the rectangle at a valid location based on a predetermined criterion.

Thus, the symbolic representation and method used for part placement in one

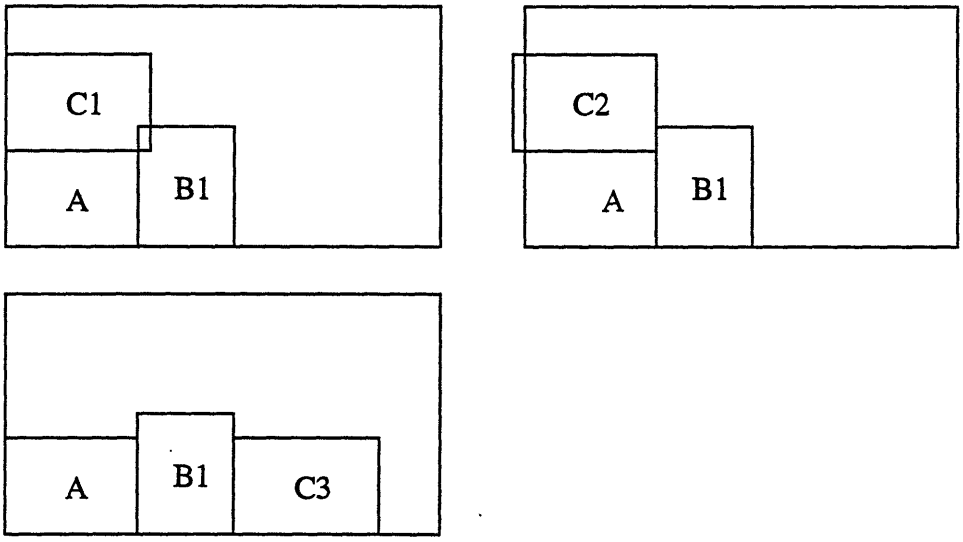
dimension can be modified for the placement of rectangular shapes in two dimensions.



Rectangle A placed on blank



Two possible locations for rectangle B



Three possible locations for rectangle C; only one valid location

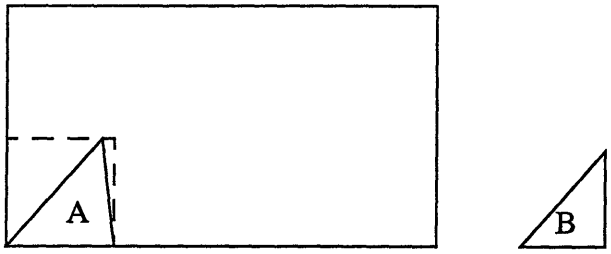
Figure 4.1.1.2 Rectangle placement on a rectangular blank

Notice again that the representation naturally reduces gaps between parts by aligning the

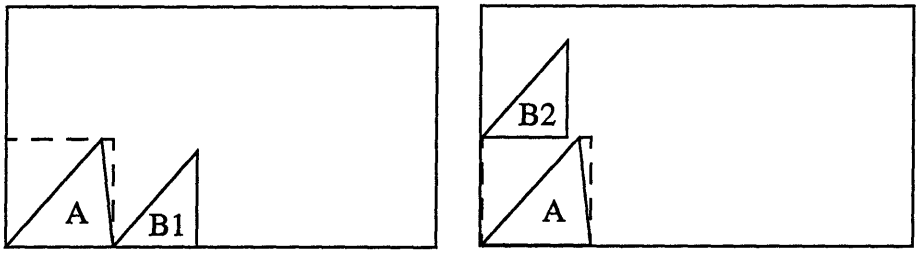
edges together. This is made possible since all the edges of the parts are parallel to the coordinate axes. A similar representation can be used for arrangement of parts in three-dimensions.

4.1.2 Naive representation

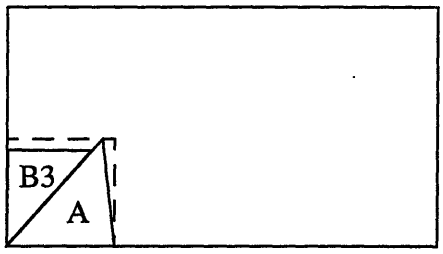
The symbolic representation described above cannot be easily extended to the case where the parts to be placed are not rectangular. As an example, consider the placement of polygonal part B shown in figure 4.1.2.1 when part A has been placed on the blank. Loca-



Polygon A placed on blank



Locations reached for polygon B using symbolic representation



Desirable location for polygon B

Figure 4.1.2.1 Symbolic representation for polygon placement

tions described symbolically are ambiguous, e.g., right-of A, above boundaryB. Part A

does not have an edge parallel to the coordinate axes and the use of the description "right-of A", is not specific enough. One could interpret the term "right-of A" to imply "to the right of the bounding box of A", and this will yield the location B1. Similarly, the description "above A, right-of boundaryL", could be interpreted to be the location B2. Considering locations B1 and B2 for placement rules out many good candidate locations, e.g. B3, which can be reached by rotating part B. Note that we have not defined what a good location is. Intuitively, location B3 appears to be a good candidate[1]. Therefore, we could restrict ourselves to a simple criterion like choosing the lowest valid location.

Thus, a symbolic representation for placement of parts of complex shapes is not obvious. For this reason, a naive representation based on two coordinates of the part, x and y, is often used. If the part shape has a rotational degree of freedom, an additional variable, θ, is included. This is illustrated in figure 4.1.2.2. If we return to the problem considered in figure 4.1.2.1, the desired configurations like B3 can be reached. However, the number of

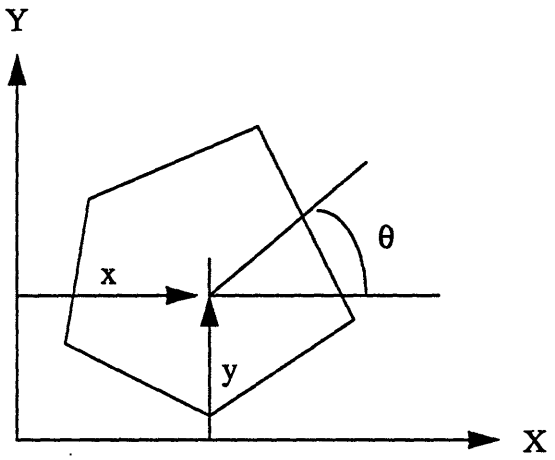


Figure 4.1.2.2 Variables used to locate one shape

possible locations is infinite. In addition, the choice of values for variables x, y and θ that leads to the lowest valid configuration is not obvious. A search over the set of variables x, y and θ is required to locate the lowest possible configuration[2]. This search is made diffi-

1. We will return to the problem of quantifying good locations in chapter 7. The locations B1 and B2 create large gaps between part boundaries and hence, can be considered to be less desirable.

cult by the fact that a large number of configurations are invalid due to the overlap of parts. In addition, the calculation of overlap is computationally expensive. A typical objective function used for such as search is:

$F = \text{Minimize } (Y_i + P*(A_o/A_i))$;

where,

Y_i = Height of the highest vertex of the polygon to be placed;

A_o = Area of overlap with other polygons;

A_i = Area of the polygon to be placed;

P = Penalty factor for overlap.

4.1.3 Assembly based part placement

The problem of locating polygons can be viewed as a problem of actually assembling them into position. Consider the trivial problem of finding the smallest area rectangular enclosure for the two polygons shown in figure 4.1.3.1. The location of one polygon is fixed. This polygons is allowed to rotate about its center. Along line L, the second polygon is located and oriented in a random manner. This polygon is moved in the vertical direction until it comes in contact with the fixed polygon. The location and orientation of the moving polygon can be varied along with the orientation of the fixed polygon until an optimal set of values is found. Notice that the expensive overlap computation operation is eliminated since all resulting configurations are valid. The objective function has a single term:

$F = \text{Minimize } (\Sigma A_i/A_t)$;

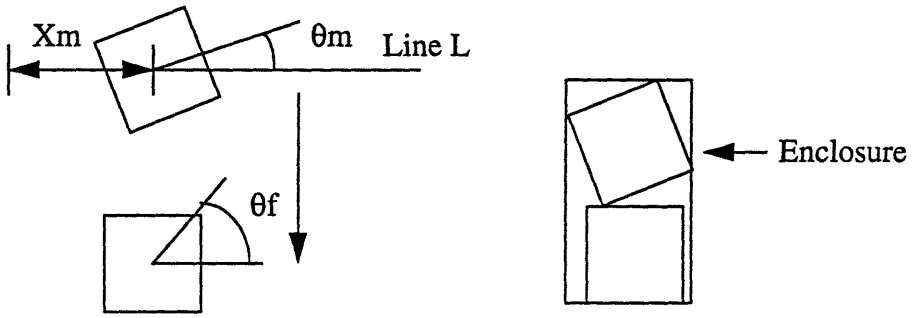
where,

A_i = Area of polygon i;

A_t = Area of the enclosing (smallest area) rectangle.

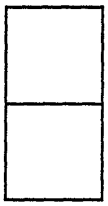
---

2. If several locations correspond to the lowest height, the lefmost of those locations can be chosen.



a. Assemble moving polygon

b. Result of assembling polygon



c. Optimal nesting

Figure 4.1.3.1 Assembly based part placement
to find the smallest rectangular enclosure

The three variables used in this approach are different from those used in the naive approach. This transformation of variables is made possible by viewing the problem of part placement as one of part assembly. This approach can be particularly useful when feasibility of assembly of given shapes is an important consideration, e.g. the assembly of components under the hood of an automobile. The naive representation can generate layouts that cannot be assembled whereas this approach automatically generates a feasible assembly procedure along with the optimal layout. Consider again the problem shown in figure 4.1.2.1. As shown in figure 4.1.3.2, the assembly based approach can be modified to find the lowest valid configuration on the blank. The polygonal part B can be dropped from the line L along the vertical direction until it comes in contact with the edges of the polygon A or the edges of the blank boundaryL, boundaryR and boundaryB. Thus, the rectangular blank can be considered to be a bin made up of three edges: boundaryL,

boundaryR and boundaryB. The parts are then dropped from the opening at the top of the bin until they interfere with the edges of the bin or any of the parts placed before. Only two variables Xm and θm are used to locate the parts. The objective function does not require any penalty term for overlap of areas:

F = Minimize Y_i;

where, Y_i = Height of the highest vertex of the polygon to be placed.

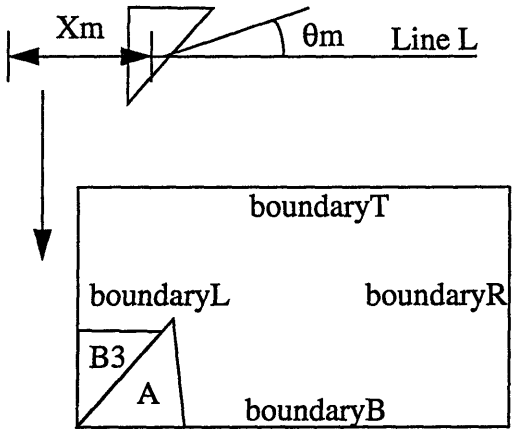


Figure 4.1.3.2 Assembly-based approach to find lowest location

Notice that the operation of moving a polygon in a given direction is governed purely by geometry; no simulation of dynamic behavior of objects is used. The interference check required is done very simply by a set of line intersection checks. This is illustrated in figure 4.1.3.3. The intersection of vertical lines drawn through the vertices of the fixed polygon with the edges of the moving polygon generates a set of distances (dm_i) that the moving polygon can travel without interference. Another set of distances (df_j) is similarly generated by interchanging the roles of the two polygons in the computation. The minimum of these distances is the amount the moving polygon can travel in the vertical direction.

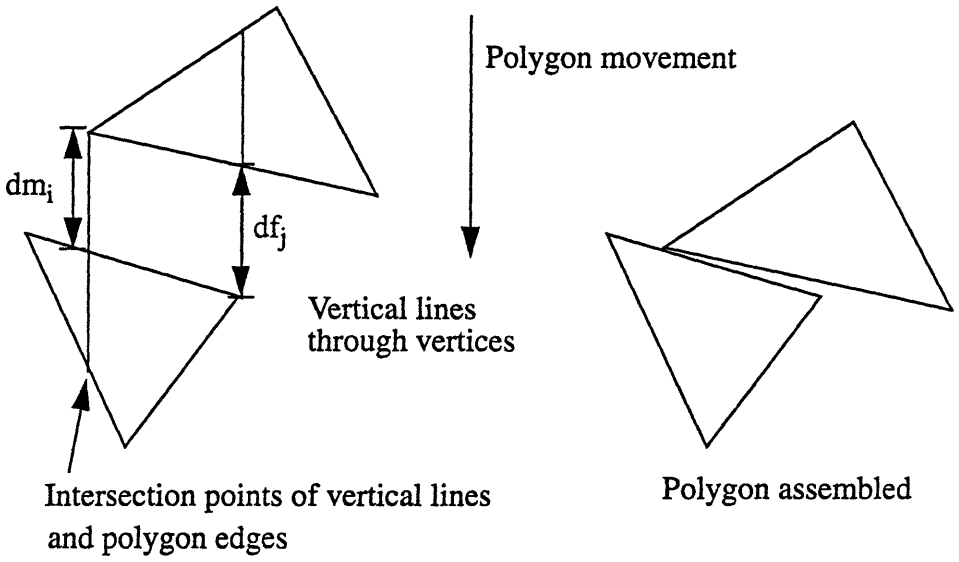


Figure 4.1.3.3 Intersection checks for assembly of polygons

Since an interference check is required between every segment of a polygon and the vertical line passing through every vertex on the other polygon (and vice-versa), the order of computation for this interference check is O(m*n), if m and n are the number of edges of two polygons to be placed optimally relative to each other. This is the same as the order of computation (for the computation) of overlap, where an interference check between every segment of a polygon and every segment of the other polygon is required. However, besides the detection of intersection, overlap computation requires the computation of the actual polygon(s) of overlap and subsequently, the area of overlap. This can be summarized as:

$(T_o = C_o*(m*n)) > (T_a = C_a*(m*n))$, since $C_o > C_a$.

where,

T_o = Time for overlap computation.

C_o = Constant of overlap computation.

T_a = Time for assembly-based computation.

C_a = Constant of assembly-based computation.

The number of operations required for overlap computation is higher than those

required for a simple interference check.

Note that certain configurations (see figure 4.1.3.4) cannot be reached using this assembly-based part placement. Also, holes inside (multiply connected) polygons cannot be filled by other polygons using this approach. This is a limitation of such an assembly-based approach to part placement. For part placement in two dimensions, an additional (third) dimension is available for part placement. Thus, when parts are cut from the blank using the configuration shown in figure 4.1.3.4, they cannot be separated from each other in the plane of the layout; they need to be disassembled along the vertical direction. The same holds true for parts placed in holes inside other parts. However, for three-dimensional parts and their arrangements, this advantage vanishes since no additional dimension is available for assembly or disassembly of parts. In that case, an assembly based representation and placement method is very useful for generating feasible configurations.

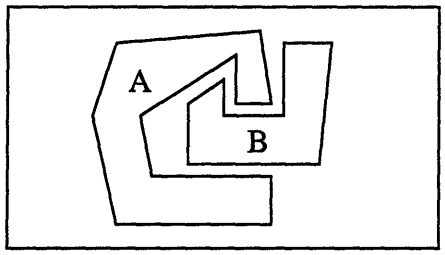


Figure 4.1.3.4 Configuration unreachable using assembly-based approach

4.1.4 Summary of part placement methods

Of the three methods described above, the symbolic representation is the simplest. It has been widely used in application domains where the parts are rectangular, e.g. VLSI layout. The naive representation is the most general, and it has been used in several of the probabilistic approaches described in chapter 3. However, it suffers from the problem of overlap computation and overlap penalization. This is discussed with examples in chapter 5. The assembly based representation avoids the problem of overlap of parts and generates feasible layouts. Chapter 6 illustrates the use of this representation in a hierarchical approach.

4.2 Layout evaluation methods

After placing the given parts, the arrangement or layout must be evaluated according to some criterion. Usually, the evaluation criteria are simple, e.g., length of the layout, percentage efficiency of packing, etc. However, depending on the placement method used, parts could overlap. In that case, the objective function must include a term to penalize the extent of overlap. This section briefly discusses methods that have been used to evaluate the overlap of parts and presents a new method as an alternative to overlap calculation.

4.2.1 Overlap evaluation

As mentioned in section 4.1.3, a direct computation of overlap area requires an interference check of all polygons with each other, the formation of overlap polygons and then the calculation of the areas of the overlap polygons. Since this direct evaluation is computationally expensive, alternative methods are needed.

Ferreira and Oliviera (1993), have reported two such methods. The first one uses discrete raster approximations of the parts and the blank. This is illustrated in figure 4.2.1.1. From such approximations, the elements of the blank covered by the parts can be easily identified. The number of such pixels covered simultaneously by more than one

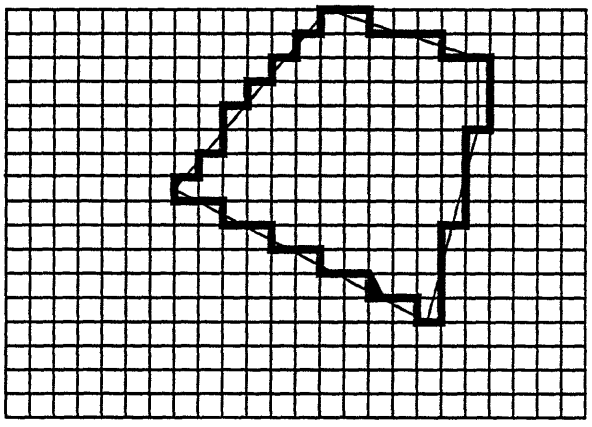


Figure 4.2.1.1 Discrete representation of blank and polygonal part

polygon can be used as a good approximation of the actual overlap area. This approach works very well when the parts are not rotated. The discrete raster approximations of the parts remain the same under translation. The pixels covered by the approximation of a part at any location can be obtained very simply by translating the original approximation. However, the approximation needs to be recomputed if the part is rotated and, this is a limitation of this approach. The second method uses the area of the bounding box of the overlap polygons as an approximation of the area of overlap. This is illustrated in figure 4.2.1.2. This requires interference checking between all the possible polygon pairs. How-

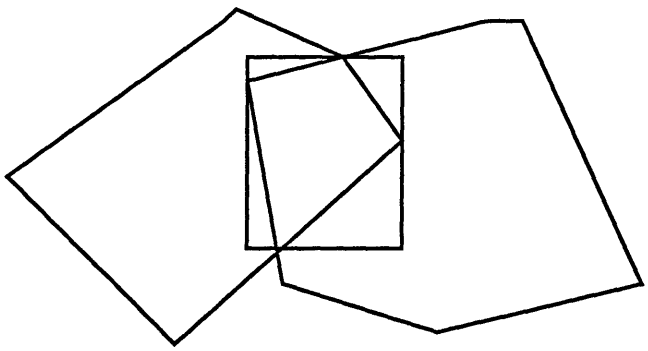


Figure 4.2.1.2 Bounding box of overlap polygon

ever, it eliminates the expensive operation of forming the overlap polygons for each polygon pair. Thus, it provides a relatively inexpensive alternative. The accuracy of the approximation depends on each case. As shown in figure 4.2.1.3, the bounding box of the overlap polygon of polygons P1 and P2 approximates the actual overlap polygon very well. The overlap polygon of polygons P3 and P4 is much smaller than its bounding box. Notice that such cases occur in a large number of layouts, especially towards the end of the search, when the edges of two parts tend to be aligned with a slight amount of overlap.

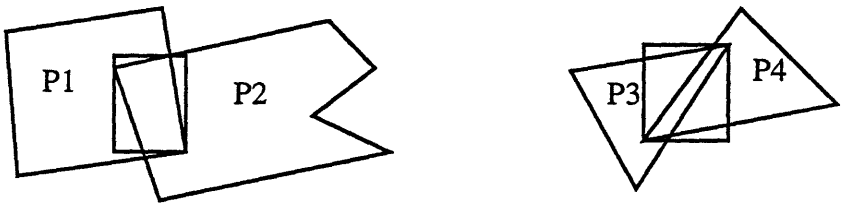


Figure 4.2.1.3 Approximation of overlap polygon by bounding box

4.2.2 Part growth based evaluation

The basic purpose of overlap computation is to quantify the extent to which the layout is invalid. The same purpose can be served by computing the extent to which the layout is valid. Consider the process shown in figure 4.2.2.1. Parts are grown into their original shape in predetermined steps starting at their area centers. Area centers of parts A and B are located and oriented by the variables of placement x, y and θ. Starting from only

the area centers, in each step, the parts expand by $1/10^{th}$ of the actual size. The total num-

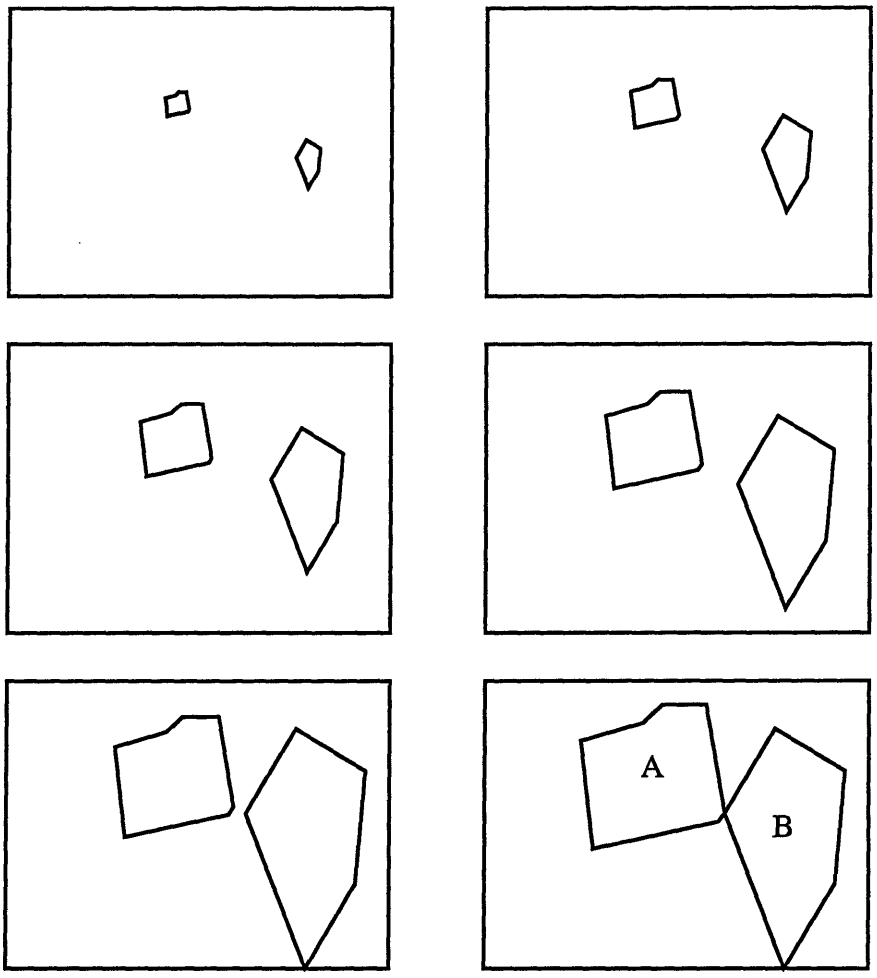


Figure 4.2.2.1 Part growth based layout evaluation

ber of steps in which the original size is reached can be varied. The growth process for any part stops when the part interferes with another part or with the boundary of the rectangular enclosure. The number of steps any part can grow before interference occurs contributes to the measure of the validity of the layout. In figure 4.2.2.1, part A grows 5 steps and part B grows 6 steps. Thus, the total validity of the layout can be considered to be 11/20 (= (5 + 6)/(10 + 10)). This validity of the layout can be combined with the main optimization criterion such as efficiency. As an example (see figure 4.2.2.2), if a single polygon C is to be placed in a configuration that minimizes the height of its highest vertex when polygons A and B have already been placed at locations where they grow to their full size, the fol-

lowing objective function can be used:

$F = \text{Minimize } (Y_i - P*(G_i/G_m))$;

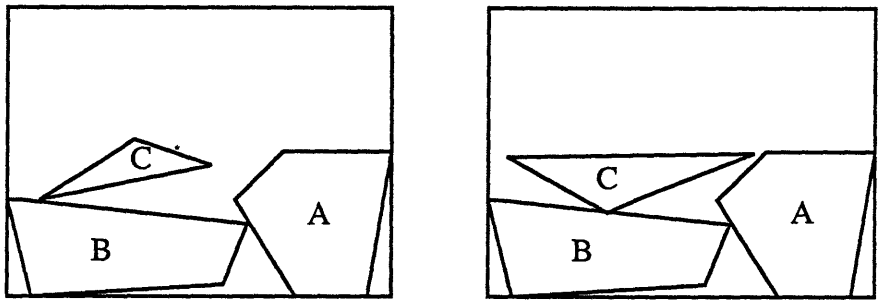
where,

Y_i = Height of the highest vertex of the polygon i to be placed;

G_i = The growth of polygon i (number of steps);

G_m = Growth (number of steps) required to reach the original size;

P = Penalty factor for the invalidity of the layout.



Randomly chosen configuration Lowest possible configuration

Figure 4.2.2.2 Placement of part using growth based evaluation; parts A and B are already placed; part C is to be placed in the lowest configuration.

Each step of the growth process can be thought of as increasing the radius of the polygons by a fraction of the total radius. Since the area of enclosing circle increases as the square of the radius, if steps of equal size are used, the increase in area of the polygons in every successive step is higher. Therefore, the radius of the enclosing circle (and hence, the expansion) of the polygon can be calculated using the square root of the step number, instead of the actual step number. This ensures similar increase in area of the polygons in each growth step.

When n polygons are placed simultaneously, this method requires an interference check of all the polygons with each other ($O(n^2)$) for each growth step and the computa-

tion time varies depending on the number of growth steps used. However, the computationally expensive operation of forming the overlap polygons is eliminated. Thus, the order of computation is the same as in the case of direct overlap evaluation, but the number of steps are different. Therefore, it provides an efficient measure of the validity of a layout. When a total of 10 growth steps is used, the method is about 3 times faster than the computation of actual overlap.

Chapter 5

Totality Approach

Part placement can be approached in two basically different ways. Placement of all the parts can be considered simultaneously in what we term a *totality approach.* All the variables required to describe the locations and orientation of the parts are considered simultaneously for optimization. This approach is very different from a *hierarchical approach,* in which a single part or a set of parts, is placed at a time in the space available. In this chapter, the totality approach is discussed. Three variables are required to locate one shape on a plane. Using genetic algorithms for the minimization of the total area occupied, the total (3n) variables required to locate n shapes are mapped to a chromosome. The naive representation discussed earlier is used with a totality approach.

5.1 Representation

Figure 5.1.1 shows a set of parts to be arranged in the smallest possible rectangular enclosure. The three variables (x, y, θ) required to locate each shape on a plane can be

Figure 5.1.1 Polygons to be arranged in the smallest area rectangular enclosure

mapped to a binary chromosome as shown in figure 5.1.2. With the chromosome representation chosen, for a randomly generated set of values, overlap of parts is produced very easily as shown in figure 5.1.3. The usual single point crossover can be used with such a

1111 0110 0011 0010 1101 1111 0110 1011 0001 0001 1100 0110

x y θ x y θ x y θ x y θ

Polygon 1 Polygon 2 Polygon 3 Polygon 4

Figure 5.1.2 Location and orientation variables mapped to binary chromosome

representation. Notice that in the chromosome shown in figure 5.1.2, the bits corresponding to some polygons are closer to each other than others. As an example, the bits corre-

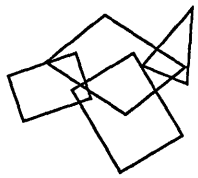


Figure 5.1.3 Overlap of polygons produced with randomly generated values for the bits in the chromosome

sponding to polygons 1 and 2 are closer to each other than the bits corresponding to polygons 1 and 4. This means that the probability of the bits corresponding to polygons 1 and 2 being carried into the offspring after a single point crossover is higher than the similar probability for polygons 1 and 4. To overcome this bias caused by single point crossover, a modified crossover as shown in figure 5.1.4 can be used. This crossover is similar in spirit to the single point crossover. The bits corresponding to a randomly chosen number of polygons are carried into one of the offspring directly. The bits corresponding to all but one of the remaining polygons are carried directly into the other offspring, and a single point crossover occurs on the substring of bits corresponding to the remaining polygon. In figure 5.1.4, the bits corresponding to polygon 1 and 4 (selected randomly) are carried into child1 from parent 1 and into child2 from parent 2. The bits corresponding to polygon 2 (selected randomly out of the remaining polygons 2 and 3) are carried directly into child2 from parent 1 and into child1 from parent2. A single point crossover occurs within the bits corresponding to (the remaining) polygon 3. Such a crossover eliminates the positional bias created by a single point crossover. We refer to this crossover as a *modified single point crossover.*

The two methods of layout evaluation discussed in chapter 4, overlap computation and growth based evaluation, can be used to quantify the validity of a layout. The fitness function used for the evaluation of the layouts generated by a naive representation penalizes the invalidity of the layout.

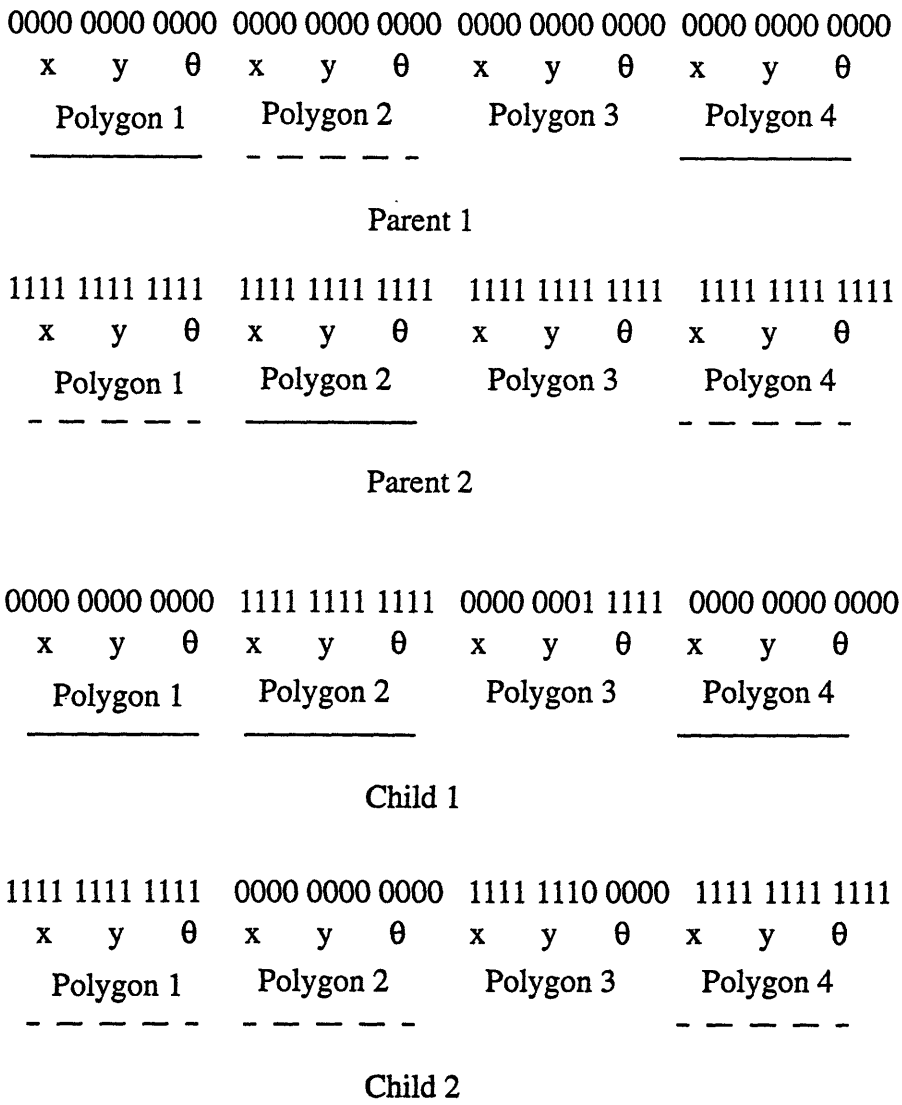


Figure 5.1.4 Modified single point crossover

5.2 Overlap computation

The following fitness function is used when the overlap area of parts is evaluated directly:

$F = \text{Minimize } (A_t/(\Sigma A_i;\ i = 1,..,n) + P^*(\Sigma\ (\Sigma A_{ij}/A_j;\ j=i,..,n);\ i=1,..,n)/((n^2));$

where,

A_i = Area of polygon i;

A_t = Area of the rectangular enclosure;

A_{ij} = Area of overlap of polygon i and j;

P = Penalty factor for overlap.

Overlap area for each polygon pair is normalized by the area of one of the polygons and penalized. Also, the entire summation is normalized by the total number of polygon pairs possible. Thus, the maximum value reached by the penalty term in the fitness function is 1.

Figure 5.2.1 shows layouts obtained for different values of the penalty factor P. As can be expected, large values tend to produce layouts with little overlap but low nesting efficiency and vice versa. Each of the results shown were obtained after 125 minutes of computation on a 50 mips workstation. A simple genetic algorithm was used. The relevant details of the genetic algorithm are listed below:

Crossover: Modified single point;

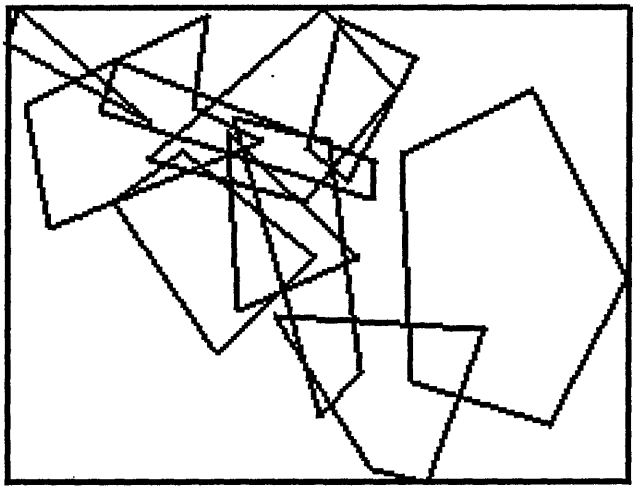
Population size: 100;

Number of generations: 300;

Probability of crossover: 0.65;

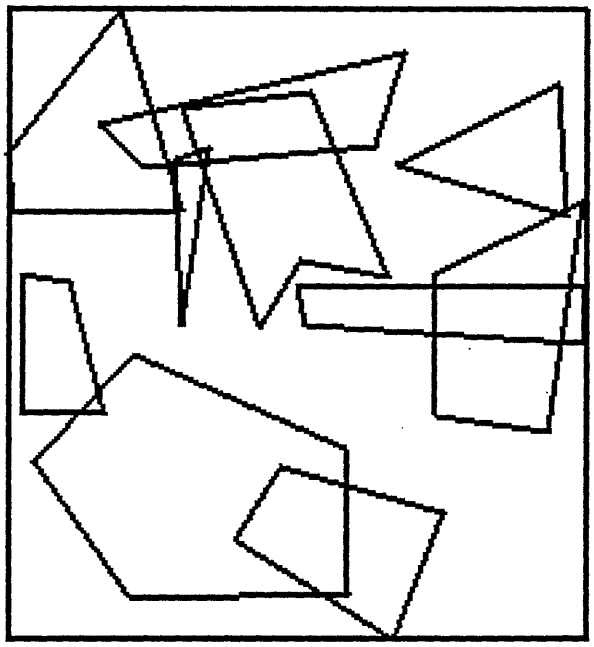
Probability of mutation: 0.003;

Selection method: Roulette-wheel based, with elitist selection.

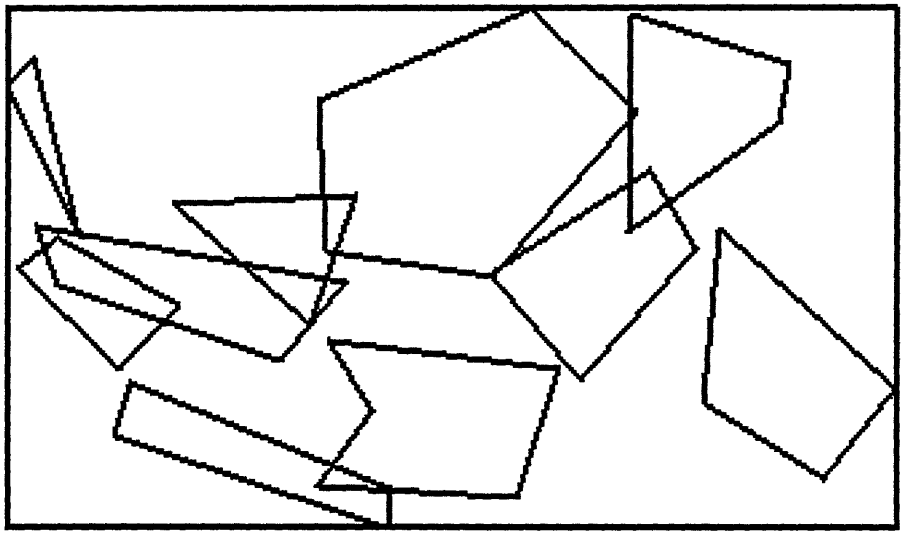


Penalty factor = 1.0

Figure 5.2.1 Layouts with different overlap penalty functions

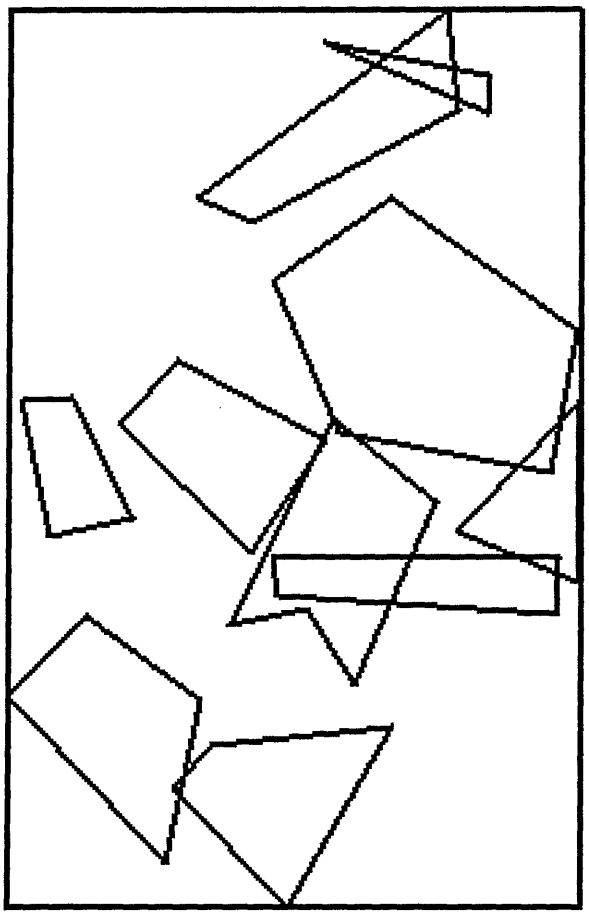


Penalty factor = 10.0



Penalty factor = 25.0

Figure 5.2.1 (continued) Layouts with different overlap penalty functions



Penalty factor = 50.0

Figure 5.2.1 (continued) Layouts with different overlap penalty functions

5.3 Growth based evaluation

The alternative to overlap evaluation, growth evaluation, described in section 4.2.2 can be used to quantify the total validity of a layout. The locations and orientations of the polygonal shapes are altered by a genetic algorithms search. After growing each shape in predetermined steps until it interferes with the other shapes, the total growth is used as a measure of the validity of the layout. The following fitness function can be used:

$F = \text{Maximize } ((\Sigma A_i; i = 1,..,n)/A_t + G^*(\Sigma(G_i/G_m); i=1,..,n)/n);$

where,

A_i = Area of polygon i;

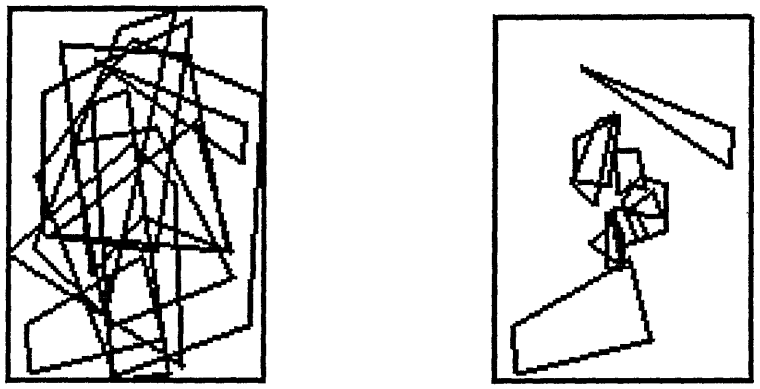
A_t = Area of the rectangular enclosure;

G_i = The growth of polygon i (number of steps).

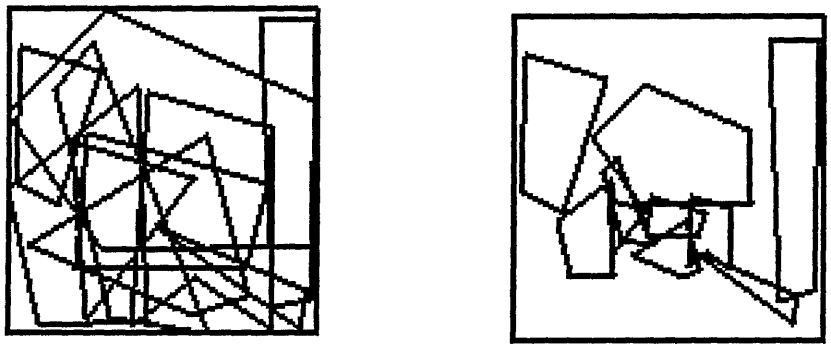
G_m = Growth required (number of steps) required to reach the original size.

G = Growth factor for validity of layout.

The second term in the fitness function measures the total validity of the layout. The total growth possible is divided into Gm number of steps. The growth factor can be varied just as the overlap penalty factor. Figure 5.3.1 shows layouts obtained for different values of G.The polygons grow to their full size in 10 (Gm) steps. In the figures on the right, the polygons have been grown until they come in contact with other polygons. If they are allowed to grow further into their full size, they overlap with each other as shown in the figures on the left. Again, for high values of G, the polygons in the layouts are spread far away from each other. The polygons have grown closer to their original size and thus have less overlap. The computation time required in this case for the same number of evaluations as in section 5.2 is much less: 30 minutes. This illustrates the computational efficiency of the growth based evaluation method. The number of steps in which the parts are grown can be increased towards the end of the run for better accuracy. As mentioned in section 4.1.3, the polygons are expanded in each step by the square root of the step number to ensure similar growth in each step.

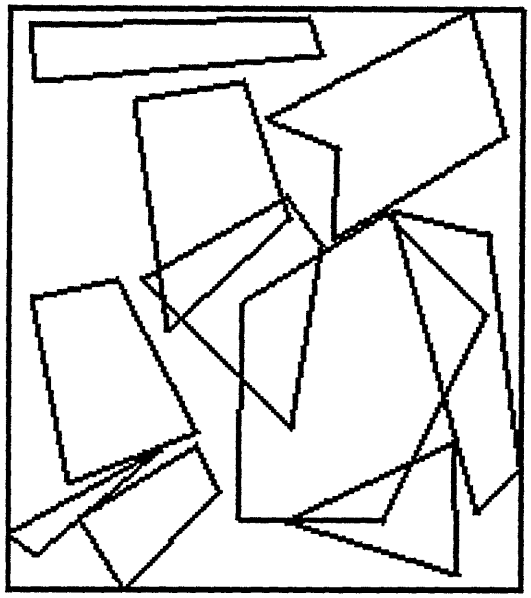


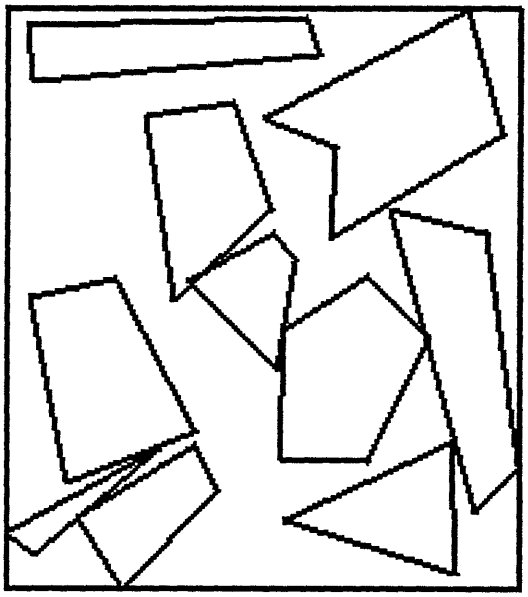
Growth factor = 0.2



Growth factor = 1.0

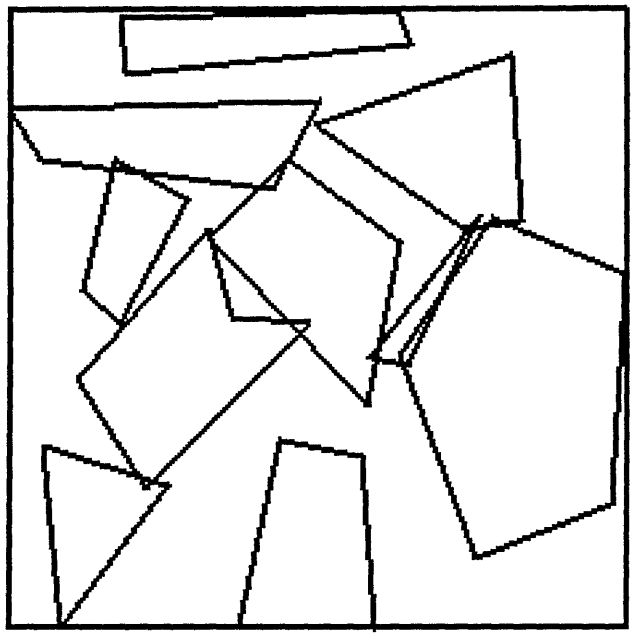
Figure 5.3.1 Layouts with different growth factors

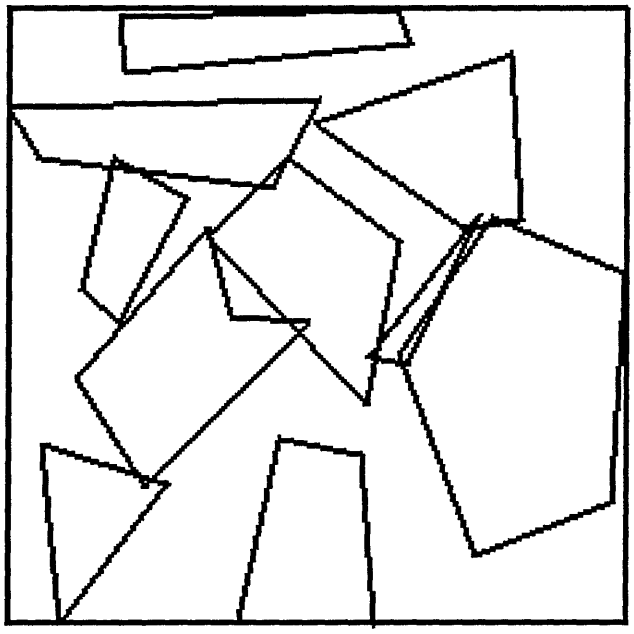




Growth factor = 5.0

Figure 5.3.1 (continued) Layouts with different growth factors





Growth factor = 10.0

Figure 5.3.1 (continued) Layouts with different growth factors

5.4 Limitations of a totality approach

The results shown in the previous section expose the limitations of a totality approach. Parts overlap even at the end of the run. The packing densities obtained are not satisfactory (The same polygonal shapes can be packed to obtain better packing efficiencies using a hierarchical approach, as illustrated in chapter 6). The poor results could be attributed to the representation and the crossover used. As mentioned earlier, a randomly generated set of values for the binary chromosome often leads to invalid layouts. The crossover operation can also yield invalid layouts after mating two valid parents. This is illustrated in figure 5.4.1. The two parents chosen for crossover are valid layouts, yet after performing a crossover, the two offspring layouts produced are invalid. The building

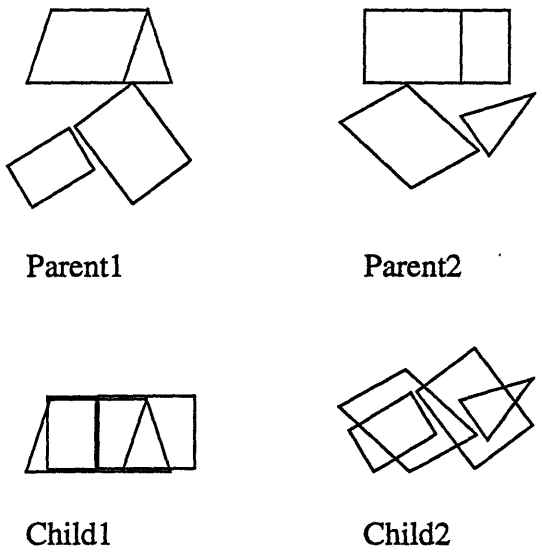


Figure 5.4.1 Destructive nature of crossover for a totality approach

block hypothesis on which genetic algorithms operate, does not hold true in this case. Combining two good blocks from parent1 and parent2 does not lead to a better individual, very often it leads to an invalid individual. A schema carried over into the offspring during the crossover operation does not represent the same characteristic in the offspring as in the parent. At first glance, since any schema represents the location and orientation variables of some of the shapes being arranged, replicating these values in the offspring would appear to carry over certain characteristics of the parent into the offspring. However, the

characteristics and quality of a layout include the location and orientation of each part *relative* to the remaining parts. The two subgroups that are combined to produce offspring are, by themselves, good building blocks, since the remaining parts are not placed over them. These characteristics of the layouts are only implicitly represented in the chromosome; certainly, the crossover operation does not take into account the importance of the relative positions of parts over absolute positions. This importance of positioning rectangular parts relative to each other is captured in the symbolic representations described earlier in section 4.1.1. In summary, a better representation than simply mapping the variables x, y, θ into a binary chromosome is needed.

Chapter 6

Hierarchical Approach

This approach is termed "hierarchical" since it consists of two levels of optimization. Given a set of polygons, a high level GA operates on symbolic arrangements of the polygons and tries to find the optimal arrangement. A low level GA decodes these arrangements in a certain manner and actually lays out the polygons by finding optimal locations for each polygon in the arrangement. In this section, we describe two implementations of this hierarchical approach.

In the first implementation, a binary tree representation is used at the higher level. The lower level GA is used to find the smallest area rectangular enclosure for a pair of polygons or polygon "clusters". The representation is used to build a binary tree of polygons by using the lower level GA at each node of the tree. This representation is used when there is no constraint on the width or length of the desired layout. The higher level GA searches for the arrangement of polygons in the binary tree that minimizes the area of the total rectangular enclosure.

In the second implementation, a string based representation is used at the higher level. The string represents the order in which polygons are packed on a blank of given width. The lower level GA is used to optimally pack a single polygon on the blank. Polygons are packed on the blank in the order represented by the string and the higher level GA attempts to find the order that minimizes the length of the layout.

6.1 Binary Tree representation

As described in section 4.1.3, an assembly-based approach can be used to find the smallest rectangular enclosure for two given polygons. This approach can be extended to the case where the optimal enclosure for two clusters of polygons is to be found. This is

illustrated in figure 6.1.1, where the smallest area rectangle enclosing the two clusters 1 & 2 is found. Notice that the two polygons in each polygon cluster (cluster 1 and cluster 2) maintain their relative positions. It is thus possible to build up a binary tree of polygons based on many such pairwise nesting operations as shown in figure 6.1.2. Each node in the

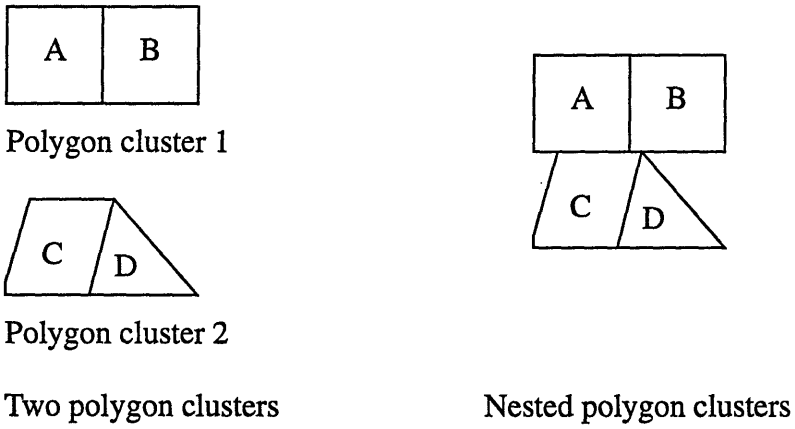


Figure 6.1.1 Optimal nesting of polygon clusters

tree represents a search for the smallest area rectangle for two clusters of polygons; the low level GA performs the evaluation of each node in the tree. After nesting two polygons or two clusters of polygons, the resulting cluster of polygons maintains relative locations and orientations among constituent polygons in all subsequent operations. The tree is

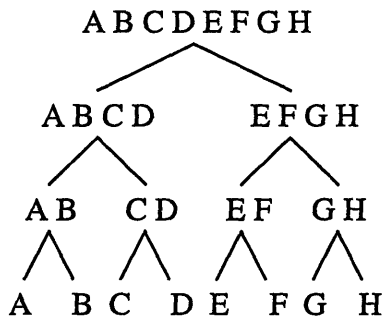


Figure 6.1.2 Binary tree representation

evaluated from the bottom up, resulting in a nesting process as illustrated in figure 6.1.3. Since each node evaluation is restricted to two polygons or polygon clusters, the number

of variables is always three, as explained in section 4.1.3. This ensures fast and robust search at each stage.

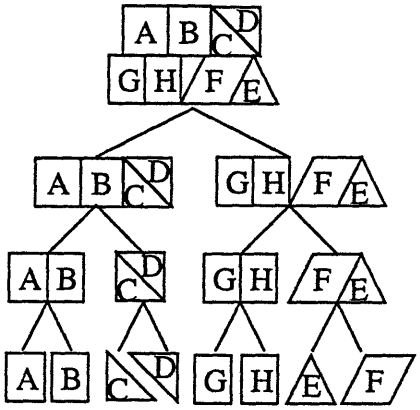


Figure 6.1.3 Evaluation of binary tree

The choice of polygons to be paired with each other obviously governs the quality of packing attained in each packing operation. The higher level GA finds the best tree. The chromosome for this search is purely symbolic as shown in figure 6.1.4. The chromosome appears at the bottom level of the tree, defining the tree uniquely. Such a binary tree representation allows only polygon sets of size 2^n to be packed. This limitation can be overcome by inserting dummy polygons of zero area into the chromosome. The low level search is not required if a polygon is paired with such a dummy polygon. The crossover

A B C D E F G H

Figure 6.1.4 Chromosome for binary tree representation

and mutation operators used for this search will be defined in section 6.3. No restrictions are imposed on the length or width of the enclosing rectangles found at each stage of nesting. The fitness function for the high level GA is:

$F = (\Sigma A_i;\ i=1,..,n)/A_t;$

where,

A_i = Area of polygon i;

A_t = Area of the enclosing (smallest area) rectangle.

Notice that both (i.e., the length and width) dimensions of the smallest area rectangle can be varied as shown in figure 1.1.1.3. Thus, the binary tree representation is suitable for the case where there is no restriction on the dimensions of the resulting enclosure.

Since the relative locations of polygons after each node in the tree is evaluated do not change in the evaluation of the higher nodes, it is possible that the global optimum may not be found. But for a problem of combinatorial optimization such as nesting, it is perhaps more important to find satisfactory solutions in a reasonable amount of time. In most cases, good local nesting tends to produce good nesting overall. This observation about the nesting problem forms the basis of nesting polygons in a hierarchical manner. The typical packing densities achieved even in the very beginning of the higher level search are comparable to those obtained by a naive representation (chapter 5) towards the end of the optimization run. This fact can be attributed to the local optimization carried out at each node in the tree. Thus, average densities obtained by using a hierarchical representation are higher than the average densities obtained by using a naive representation.

Such a hierarchical representation follows the building block hypothesis closely. Each node finds the smallest area rectangular enclosure for two sets of polygons. The combination forms a building block which in turn is combined with another such building block that is created on the adjacent node at the same level in the binary tree. Genetic algorithms rely on the hypothesis that combining two good building blocks leads to another good block. This representation ensures that this hypothesis holds true by finding optimum enclosures at each node in the tree. The hypothesis fails often in the case of a naive representation, since well packed complementary parts of two parents can produce an invalid layout upon combination via crossover.

6.2 String representation

Often, the given shapes are to be cut from a roll or blank of material of constant

width, as shown earlier in figure 1.4, e.g., the layout of sheet metal parts or pieces of garments. Usually, this constraint of fixed width is accounted for using a penalty term for its violation (see Fujita et al., 1993; Jain et al., 1990; Oliviera and Ferreira, 1993). This problem can be avoided by using a variant of the approach based on assembly explained earlier in section 4.1.3. A "bin" of width W, equal to the blank width, is set up as shown in figure 6.2.1. The polygons to be packed are then assembled into the bin by "dropping" them from line L sequentially. For each polygon i, an optimal configuration in the bin is found by varying the location from which it is dropped and its orientation.The low level GA performs this local optimization. Several criteria for this local optimization can be used. One such criteria we use is the lowest location in the bin:

$F = (Y_i; i=1,..,n);$

where,

Y_i = Height of the highest vertex of the ith polygon (being currently packed) in the bin.

It can be seen that the order in which the polygons are dropped into the bin controls the quality of the nesting along with each local optimization. The higher level GA in this case finds the best possible order. The chromosome used is a list of the polygons (figure 6.2.2) similar to the case of the binary tree described earlier, but with a very different geometric interpretation. Evaluation of this chromosome is done by packing each successive polygon in the list from left to right as illustrated in figure 6.2.3. Once the optimal location for a particular polygon in the bin is found, it is held fixed at that location for the packing of the remaining polygons in the chromosome. The fixed width constraint is automatically accounted for by restricting the range of locations of the polygon along the line L to the width of the bin W. Only two variables govern the packing of each polygon, making each local search very efficient. For the high level GA, the fitness function used is:

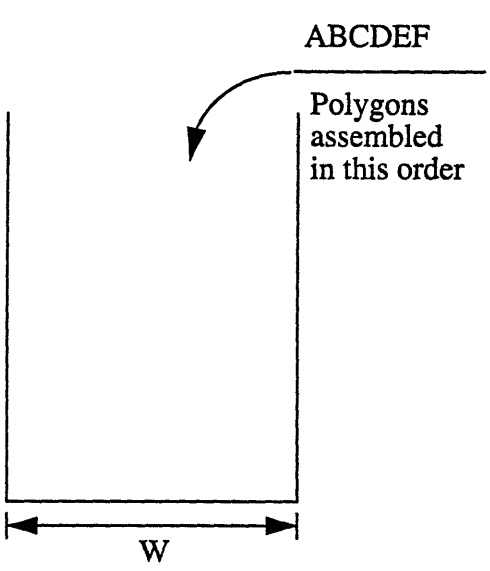
$F = \text{Minimize } (\text{Max } \{Y_i \mid i = 1,...,n\});$

where,

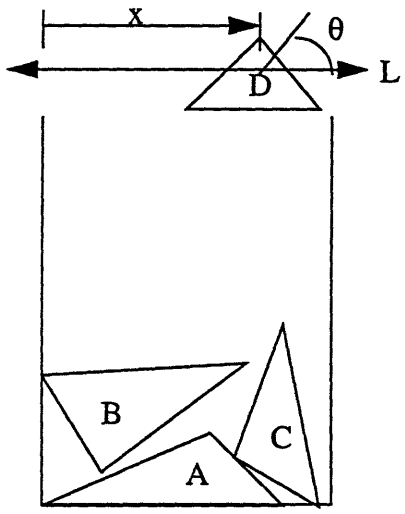
Y_i = Height of the highest vertex of the i^{th} polygon in the bin;

n = Number of polygons to be packed.

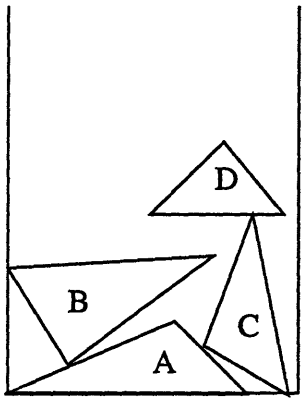
Again, local minimization at each stage ensures a high packing density for each layout as in the case of a binary tree representation. Similarly, the possibility of missing the global optimum exists. This approach is a generalization of the "skyline-pack" algorithm described by Smith (1985). The building block hypothesis works even in this case, although in a different sense. Each stage of local optimization ensures attachment of a good building block to the one constructed in the previous stage. The blocks are built in a sequential manner, as against a hierarchical manner in the tree-based representation.



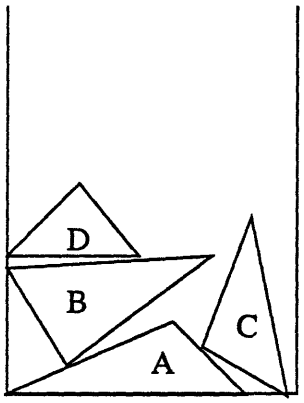
a. Blank available



b. Locating polygon
with low level GA



c. Polygon D assembled
with chosen x & θ



d. Optimal location
for polygon D

x	θ
11101	10101

e. Chromosome for
low level GA

Figure 6.2.1 Low level GA for locating one polygon optimally

A B C D E F

Figure 6.2.2 String chromosome

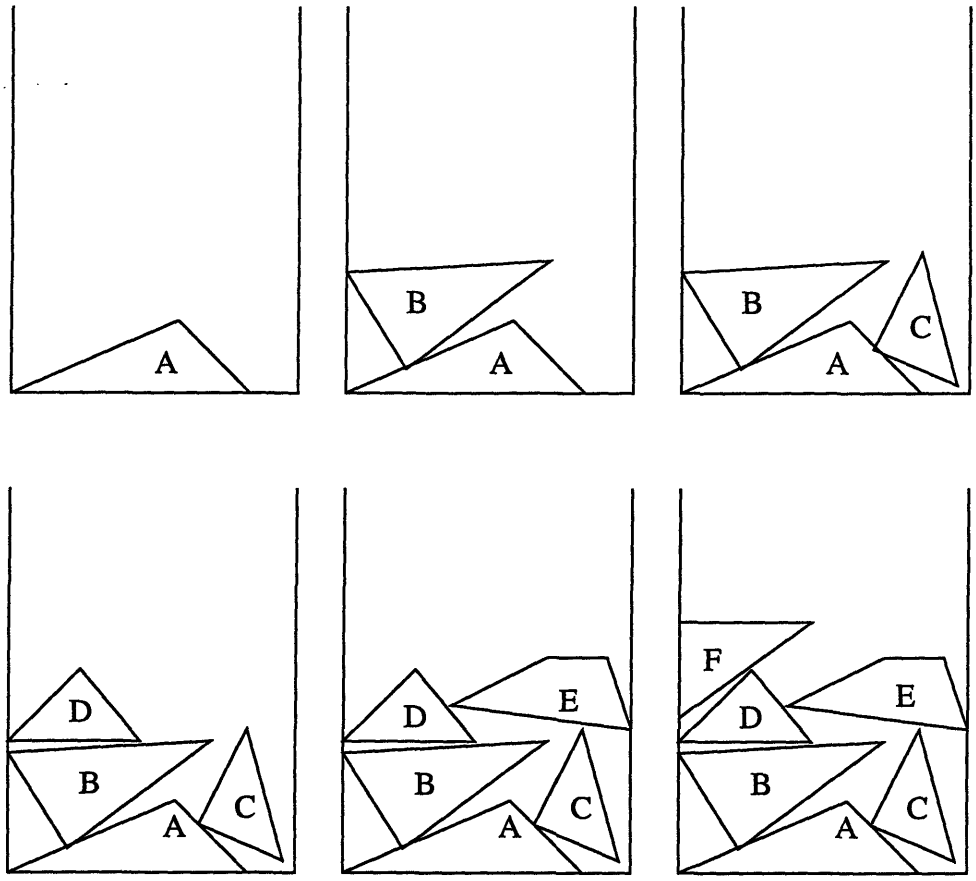


Figure 6.2.3 Stages of evaluation of a string chromosome

6.3 Geometric Interpretation of the Chromosome

The chromosomes used in both representations are the same, although the geometric interpretations are rather different. Figure 6.3.1 shows the mapping of the chromosome in the case of the binary tree representation to the actual geometric domain. Each line represents a node in the tree. The line L1 divides the layout into two distinct regions. Each of these regions is again divided into two distinct regions by lines L11 and L12. The geomet-

rically meaningful schemata for this chromosome are represented by subtrees under every node in the tree: AB, CD, ABCD. The number of such schemata is one less than the number of polygons. Here, by the term "geometrically meaningful schemata" we mean those substrings of the chromosome which maintain the same meaning independent of the remaining parts of the chromosome. Thus, for a chromosome such as ABCDEFGH, the schema ABCD results in the same relative arrangement of the polygons (A, B, C, D) as in the chromosome ABCDEHFG, whereas the schema FG present in both the chromosomes does not. Notice also that the two child nodes of any node in the tree can be exchanged without altering the result, e.g., the schemata AB and BA evaluate to the same result.

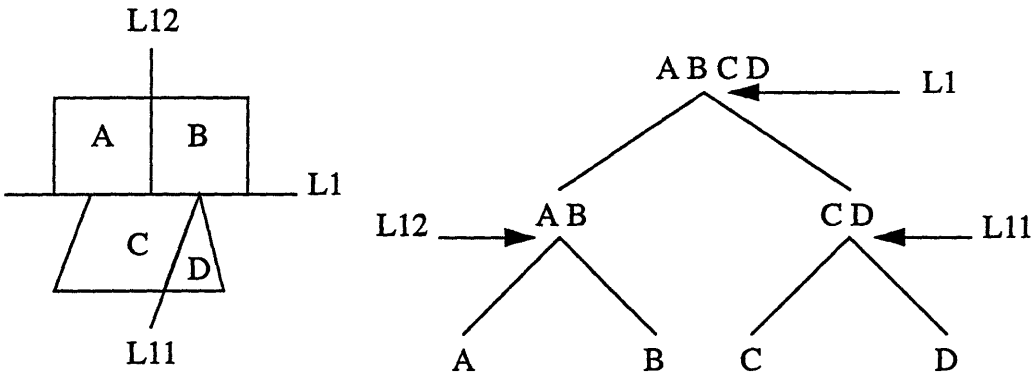


Figure 6.3.1 Geometric interpretation of Binary Tree

For the case of the string-based representation, mapping of the chromosome to the geometric domain is illustrated in figure 6.3.2. Each line represents a point along the length of the chromosome. Thus, any point in the chromosome divides the layout into two geometrically distinct regions: the substring to the left represents the regions below the corresponding line in the layout and the substring to the right represents the region above. During the process of packing polygons, the polygons packed earlier (those on the left side in the chromosome) affect the packing of those packed later (polygons on the right), but not vice-versa. Hence, the geometrically meaningful schemata in this case are defined only by the substrings starting from the leftmost end of the chromosome string, e.g., A,

AB, ABC, ABCD. Some other schemata such as CD, also map to a geometrically distinct region, but the relative arrangement of the two polygons (C and D) will be different from that shown in figure 6.3.2 if the schema in a different chromosome is, for example, BACD. The number of such meaningful schemata equals the number of polygons.

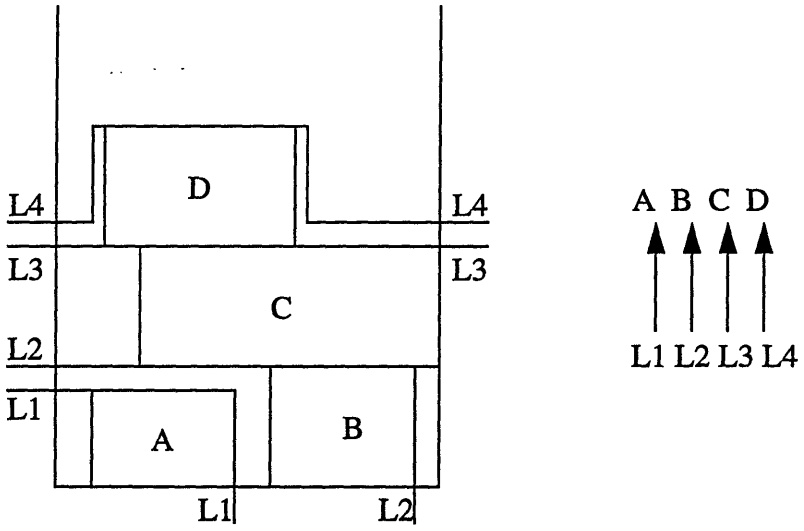


Figure 6.3.2 Geometric interpretation of string chromosome

6.4 Crossover and Mutation

Since the polygons to be nested are predefined, the traditional single-point crossover used for the higher level symbolic GA can lead to invalid chromosomes. This is illustrated in figure 6.4.1. The two parent chromosomes represent valid layouts, although the offspring do not. Two instances of polygon B are created in child 1, while the polygon F is missing. The order crossover suggested by Smith (1985) attempts to maintain one part of each parent instead of recombining complementary parts from both parents. A crossover point is chosen at random along the length of the chromosome. The left part of each parent is carried into the offspring; the order of the remaining polygons is obtained by scanning the other parent. Thus, in figure 6.4.2, the left part of parent 1 is copied into child 1 directly. It is important to preserve the left part of the chromosome since this represents a meaningful schema as explained earlier in section 6.3. If the right part of the chromosome

is preserved, the high level search would be equivalent to a random walk since the right part does not represent any characteristics of the parent in the offspring. The order of the remaining polygons D, E & F in child1 is obtained from their order in parent 2. In our opinion, obtaining the order from the other parent really has little geometric interpretation: it is more of a random shuffling of the right half of the chromosome rather than a combination. This is the crossover operation used in the string-based representation. Thus, the crossover operator preserves only geometrically meaningful schemata and allows the characteristics of fitter parents to increase in number. This operator is similar in spirit to the simulated annealing operator which finds a neighbor of a given solution.

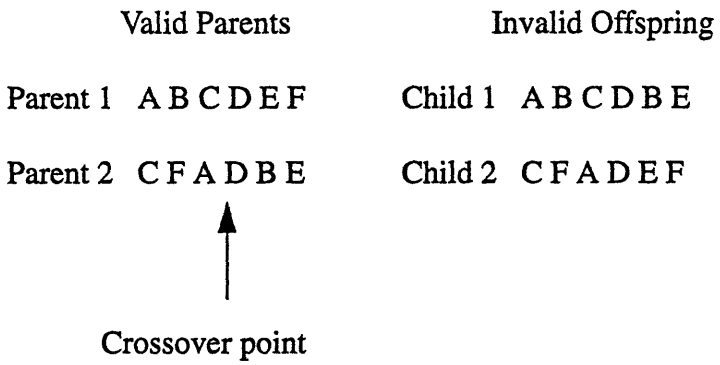


Figure 6.4.1 Single point crossover

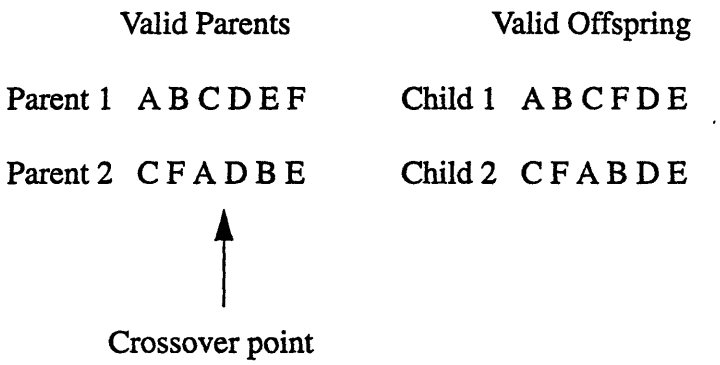


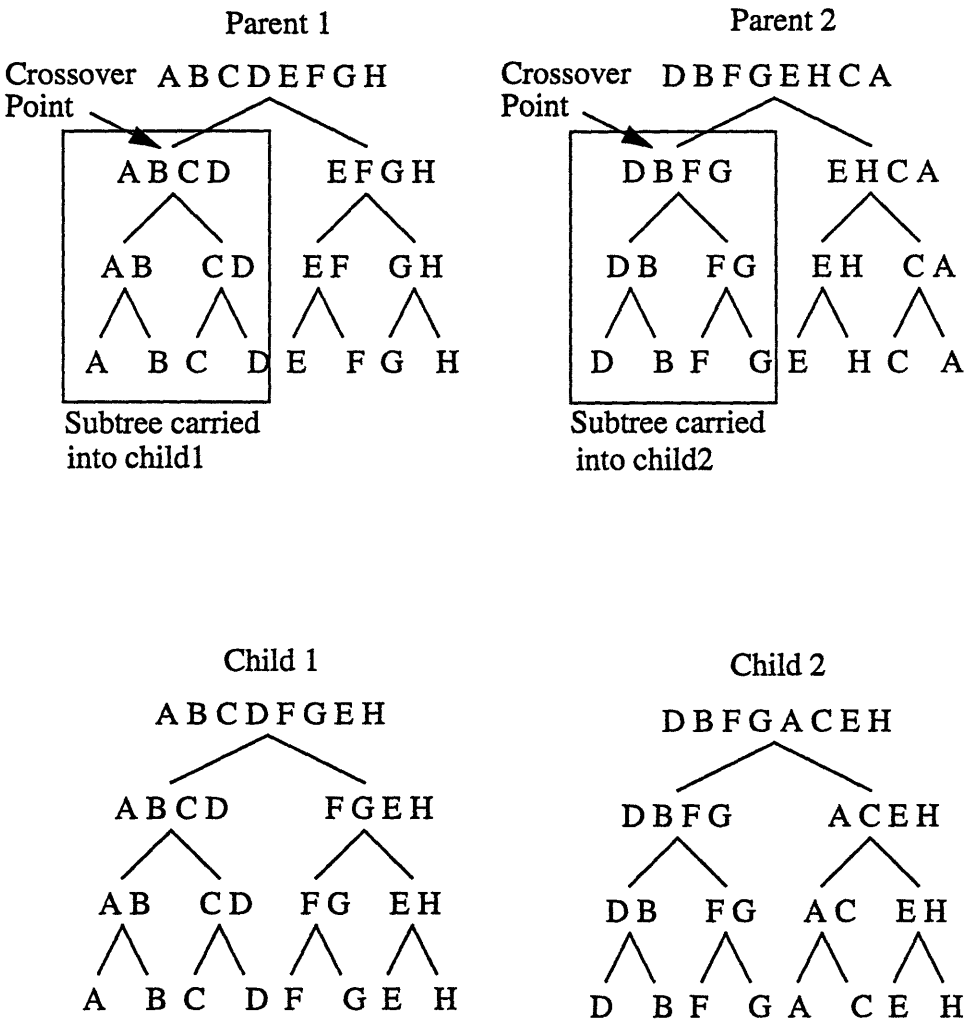
Figure 6.4.2 Order crossover

The mutation operator chooses at random a polygon and inserts this polygon at a randomly chosen location to its left as shown in figure 6.4.3. This is done so that the pop-

ulation does not remain sensitive to the initial, randomly created population. The crossover operator maintains the left part of each parent. Thus, substrings close to the left end of any chromosome are likely to be similar to those in the initial population. For example, if the initial population does not have any chromosome that has a particular polygon in the leftmost position, the crossover operator alone is insufficient to produce such a chromosome. A mutation operator is necessary to offset this bias.

Figure 6.4.3 Mutation

The crossover operator used in the tree based representation described above is a variation of the order crossover. The chromosome for this representation is as shown in figure 6.1.2 and 6.1.4; defining the bottom level of the tree defines the complete tree. In this case, the crossover points are randomly chosen nodes in the binary tree (see figure 6.4.4). The entire subtree under the crossover point node is carried into the offspring without alteration. This is based on the interpretation of the chromosome discussed in section 6.3, retaining geometrically meaningful schemata in the offspring. The remaining part of the tree is obtained by using the order in which the remaining polygons appear at the bottom of the tree defining the other parent. Again, this is more of a random shuffling of the remaining polygons as in the case of the String based representation. This obviates any need for a mutation operation. It is not possible to obtain the remaining part of the tree by directly using the corresponding subtree in the other parent; this leads to invalid chromosomes, similar to a single point crossover used for the string representation.



Result of binary tree crossover

Figure 6.4.4 Crossover operator for the binary tree representation

6.5 Reducing Computation

The local optimization computation used for both representations requires mainly intersection checks between sets of lines. Evaluation of one higher level chromosome (processing a list or tree of polygons) can be shown to be $O(n^2)$, when n polygons are to be

nested. However, the computation can be reduced by maintaining a record of previous optimizations done at the lower level. For the case of the binary tree representation, at each node, a tree optimally combines two polygons or two sets of polygons. The optimum result of such a combination will be the same every time it is encountered in the search. We compile a history of all such lower level optimizations as the higher level search proceeds. Thus, after evaluating a binary tree shown in figure 6.1.3, information for the following optimal arrangements is recorded: AB, CD, EF, GH, ABCD, EFGH, ABCDEFGH. At a later point in the search, if the chromosome BADCEHFG is to be evaluated, the subtrees BA, DC and BADC need not be evaluated again. The results of the previous evaluation of ABCD are inserted into the tree and repeated evaluation is avoided. Note that swapping the locations of the children of a node does not alter the result: the subtree AB results in the same packing as the subtree BA. This reduction in computation again is based on the definite mapping of the chromosome to the geometry. The geometrically distinct regions or the building blocks for the tree shown in figure 6.1.3 are invariant throughout the higher level search.

For the string based representation, a similar history of previous evaluations is maintained. The optimal locations of polygons on the blank will not change, provided the polygons are packed in the same order as before. After evaluating the chromosome shown in figure 6.2.2, the result at each stage of packing as shown in figure 6.2.3 is recorded: A, AB, ABC, ABCD, ABCDE, ABCDEF. When the initial part of any chromosome is the same as any of these schemata, the lower level optimization (of that initial part) can be avoided. If the chromosome to be evaluated is FABCDE, although the substring ABCDE was evaluated before, it must be evaluated again since it does not appear at the beginning of the chromosome and represents different relative locations of the polygons.

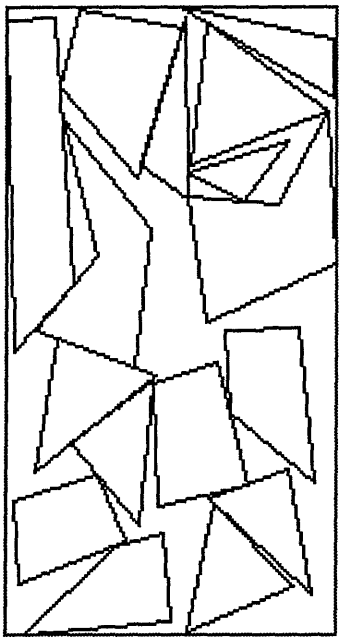
To maintain the history of computation, locations and orientations of all the polygons for each evaluation of a higher level chromosome are stored. The storage required for this purpose is bounded by the total number of evaluations carried out (number of generations times the population size) during the optimization run of the higher level GA.

6.6 Implementation details

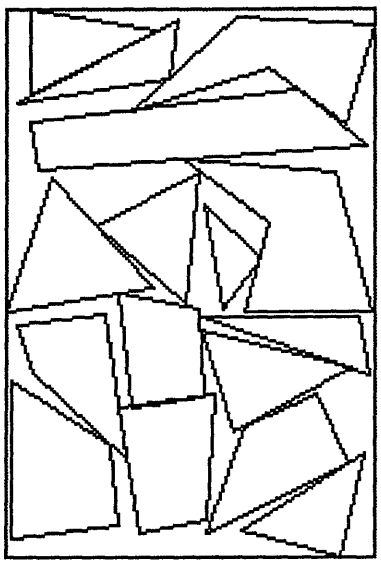
For the low level GA, a population size of 30 is used. A roulette-wheel based selection is used along with an elitist selection strategy. Probabilities of crossover and mutation used are 0.9 and 0.01, respectively. A termination criterion of 30 generations is used. For the high level GA, a population size of 60 is used. The termination criterion used is the number of generations (30). A roulette-wheel based selection is used along with an elitist selection strategy. Probability of crossover used is 0.65. As mentioned earlier, mutation is required only for the String based representation and its probability is 0.01.

6.7 Examples

In this section, results obtained by the two implementations of the hierarchical approach described above are illustrated. Figure 6.7.1.1 shows results obtained by the binary tree representation. (Note that fitness refers to packing density in the figures in this section). The plots of the best packing density as well as the average packing density vs. generation number are shown in figure 6.7.1.2. The plot of average packing density vs. generation illustrates the point made earlier that the average packing density is reasonably high from the very beginning of the run due to the local optimization at each stage. Figure 6.7.1.3 shows computation time per generation vs. generation number. The computation time reduces as generations progress since fewer and fewer subtrees need to be evaluated as explained in the previous section. A typical optimization run with 16 objects requires about 2.0 hours on a 100 mips Silicon Graphics Indigo Extreme workstation. The figures 6.7.2 show results obtained for a jigsaw puzzle, where the parts fit together to form a rectangle. In this case, only 4 optimal solutions corresponding to 4 orthogonal orientations exist. (Ideally, the packing density of an optimal packing should be 1.00, although due to clearance allowed between parts, optimal density is only about 0.91.) The tree based representation is unable to find a global optimum. This illustrates the limitation of the representation. By pairing objects at each stage, the possibility of finding the global optimum is eliminated, except in certain special cases. Figure 6.7.3 shows layouts of rectangular parts.



Fitness = 0.655



Fitness = 0.672

Figure 6.7.1.1 Sample layouts using binary tree representation

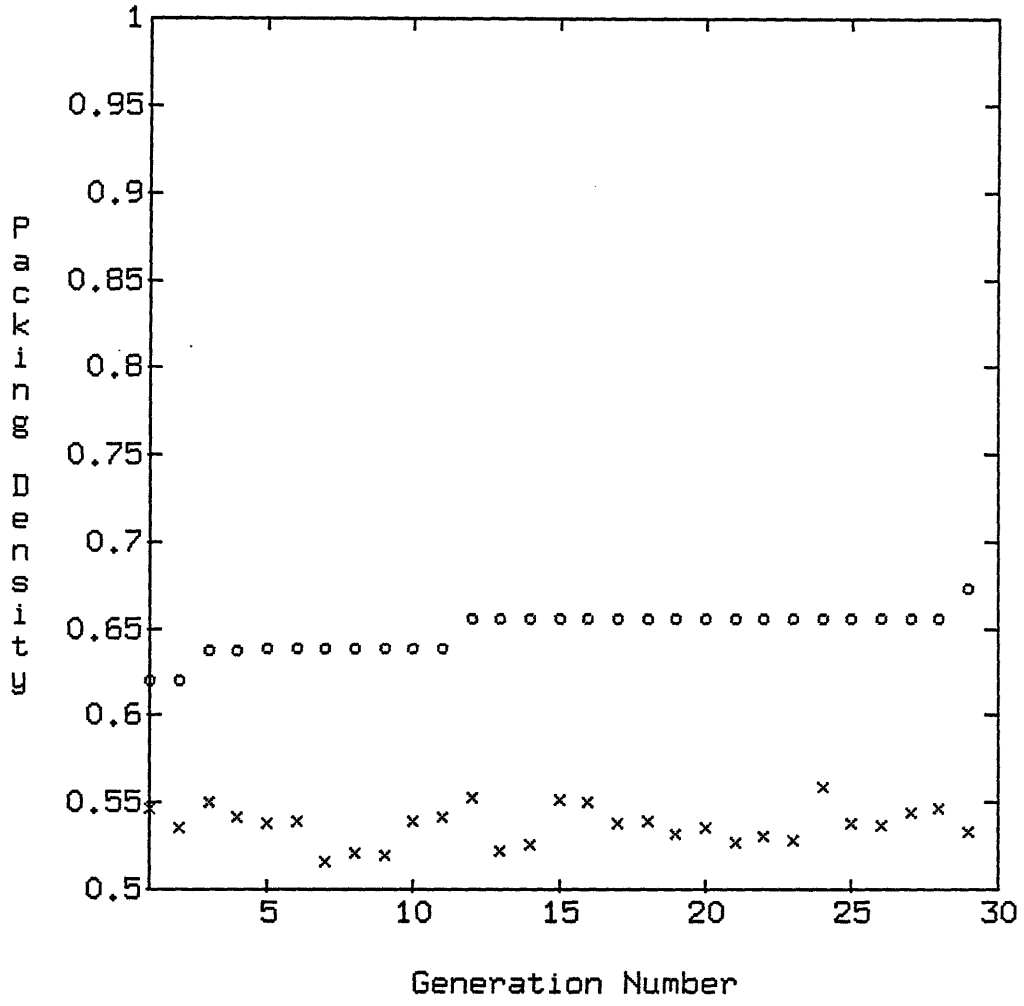


Figure 6.7.1.2 Plot of best-of-generation and average-of-generation vs. generation number

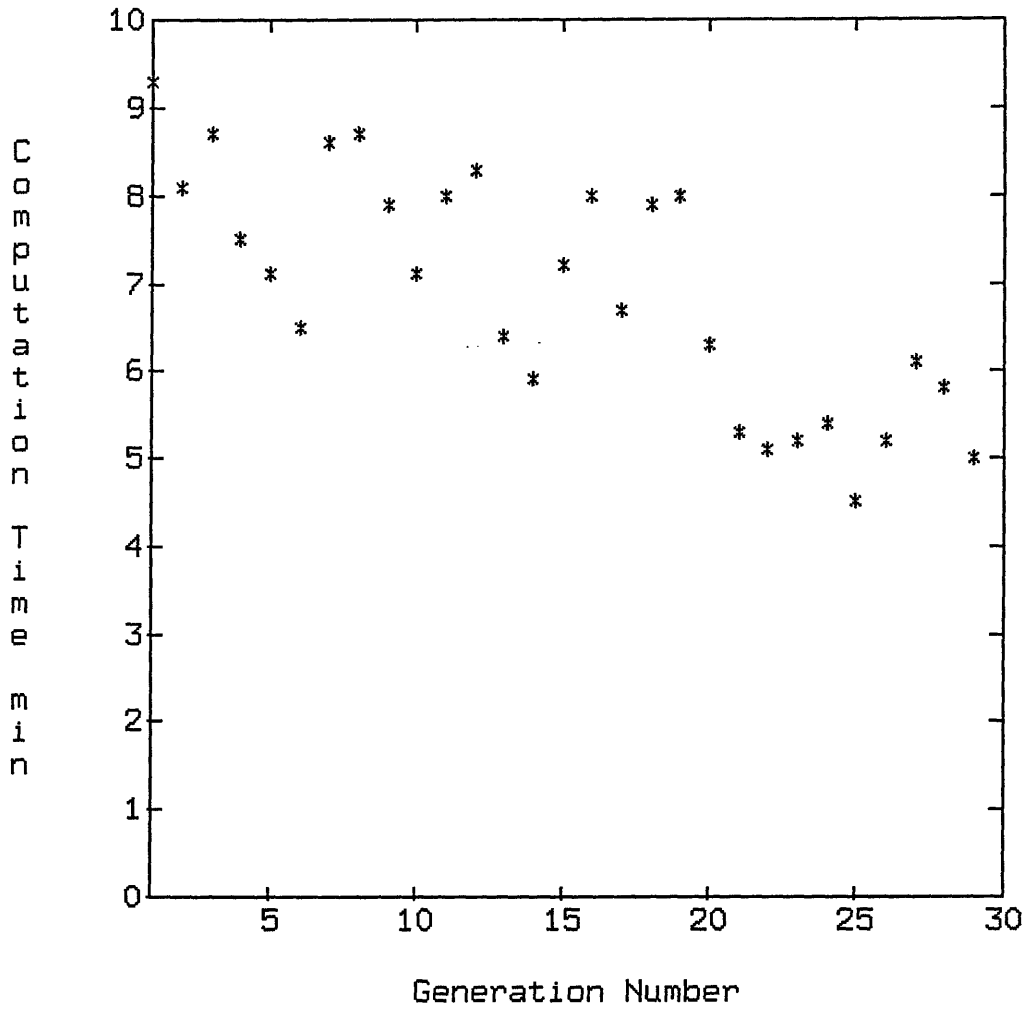


Figure 6.7.1.3 Computation time per generation for binary tree representation

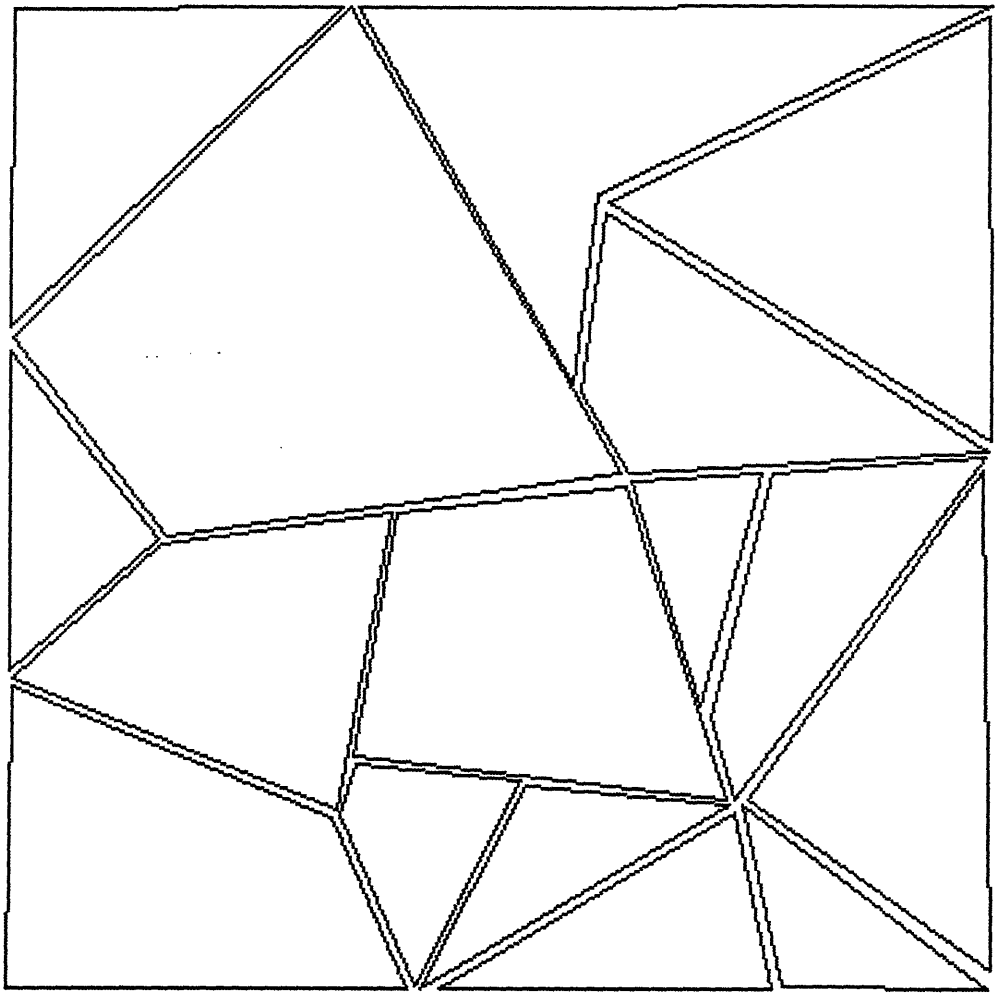


Figure 6.7.2.1 Jigsaw puzzle attempted with binary tree representation

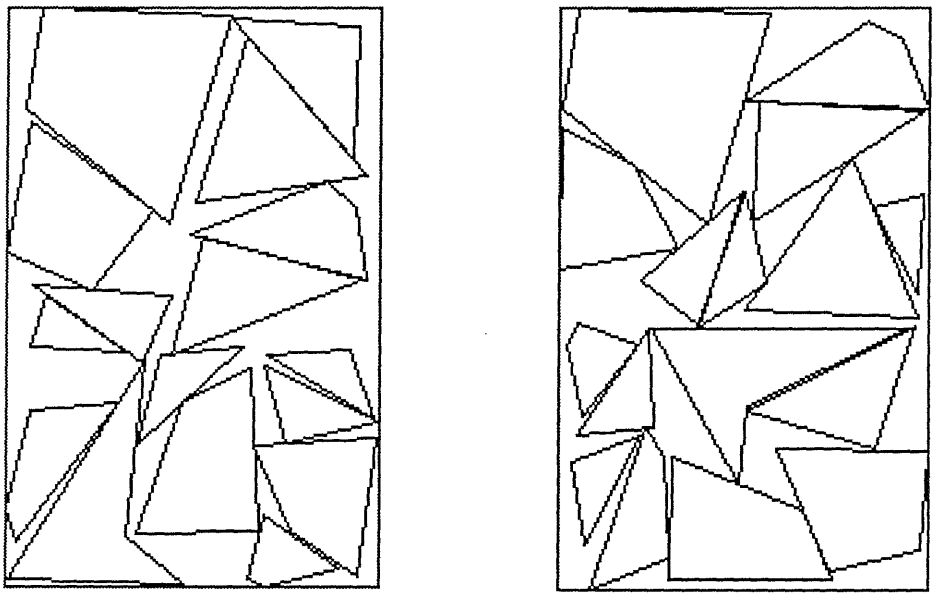


Figure 6.7.2.2 Layout of jigsaw puzzle using binary tree representation

Figure 6.7.3 Layout of rectangles using binary tree representation

With the String representation, seven different fitness functions were tried for the low level search. In optimally locating a single part, voids are created in the pack which are inaccessible to parts that are assembled later. The void area adds to the total waste in the packing. This area under the part that cannot be accessed later by other parts is termed the *shadow area* of that part. This is illustrated in figure 6.7.4.

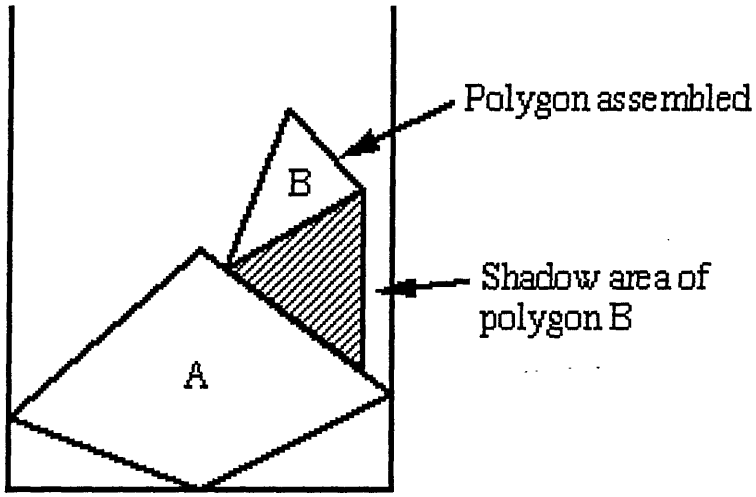


Figure 6.7.4 Shadow area of a polygon

Some of the lower level fitness functions described below specifically attempt to reduce this waste.

1. Minimize Height:

$F = \text{Minimize } (Y_i);$

where,

Y_i = Height of the highest vertex of the i^{th} polygon.

2. Optimize packing density at each stage:

$F = \text{Maximize } ((\Sigma A_k;\ k = 1,...,i)/(Y_i * W));$

This can be thought of as a normalized measure of height.

3. Reduce scrap at each stage:

$F = \text{Maximize } (S_i/S_{i+1});$

where,

S_i = Scrap at each stage;

$= (W * \text{Max } \{Y_k \mid k=1,...,i\} - (\Sigma A_k;\ k=1,...,i));$

This attempts to lower the scrap at each stage in comparison to the earlier stage.

Since the scrap of the previous stage is constant for the search, this also can be thought of as a normalized measure of height.

4. Minimize shadow area:

F = Minimize (As_i);

where,

As_i = shadow area of polygon i;

This attempts to minimize the total waste by reducing the waste at each stage.

5. Optimize packing density with penalty for shadow area:

F = Maximize $((\Sigma A_k; k=1,...,i)/(Y_i*W) - (As_i/A_i)^2)$;

6. Minimize height with upper limit on shadow area:

F = Minimize (Y_i);

s.t. $F = 1000.0$ if $As_i > 0.5*A_i$;

By assigning a very high height value, part locations that waste large areas are discarded.

7. Optimize shadow area and select lowest in height:

F = Minimize (Asi), and select solution with lowest height from three best solutions.

The search attempts to reduce the shadow area, but at the end of the optimization, out of three locations with the smallest shadow area, the lowest location is chosen.

Figures 6.7.5 show results of layouts restricted to blanks of fixed width using the string based representation for each of the fitness functions discussed above. The computation time per generation (see figure 6.7.5.9) does not drop as rapidly as in the case of the binary tree representation. This could be due to the fact that the number of possible different chromosomes is higher than in the case of the tree representation. A typical run for this representation using 10 polygons takes about 2.5 hours of computation, which is higher

than for the tree based representation. A possible reason for this could be the nature of the chromosome leading to less repetition of computation than in the case of the binary tree representation. Again, due to the limitations imposed by a single optimality criterion at the lower level of search, a jigsaw puzzle (see figures 6.7.6) cannot be solved by this representation. (The ideal density for this jigsaw puzzle is not 1.00, but 0.92). Results obtained for a set of rectangles are shown in figure 6.7.7.

Some trends about the fitness functions emerge from the examples in figure 6.7.5. The fitness function based only on height (figure 6.7.5.1) works the best, although normalizing height (figure 6.7.5.2) seems to work almost as well. Minimizing scrap at each stage (figure 6.7.5.3) works similarly, since the scrap quantity from the previous stage used in the fitness function is constant throughout the lower level optimization, and the scrap quantity for each stage is based on height. Trying to minimize the shadow area (figure 6.7.5.4) leads to several closely packed structures that do not attempt to minimize the height at each stage, leaving the higher level GA with few good candidates to choose from. By allowing only a certain amount of shadow area at each stage and minimizing height, well packed structures (figure 6.7.5.5) are obtained. Similarly, penalizing the shadow area (figure 6.7.5.6) provides satisfactory results. By minimizing the shadow area and choosing the lowest location from a few (in this case 3), of the best alternatives, the notion of maintaining a low overall height of the pack can be addressed (figure 6.7.5.7). This approach is rather ad hoc and the results reflect this fact. A more sophisticated method would require identifying a few local minima and selecting the lowest location from them.

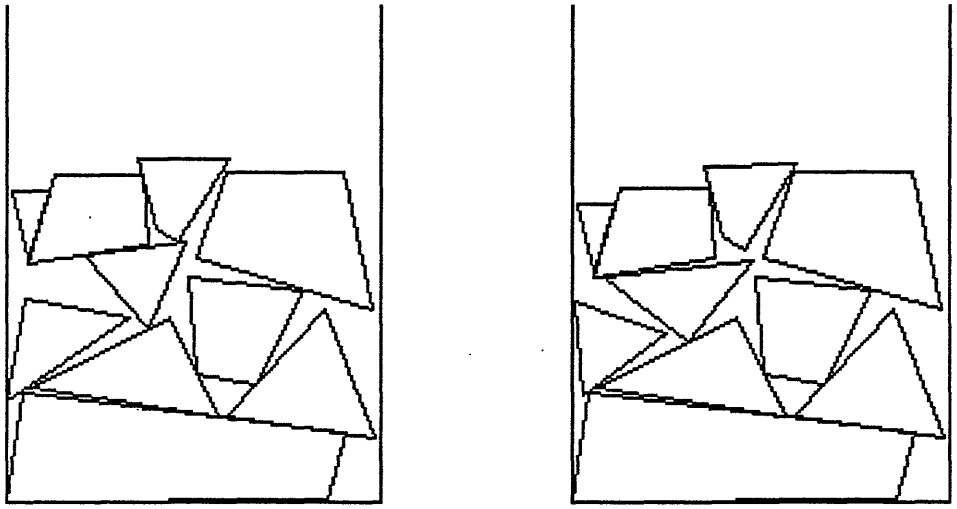


Figure 6.7.5.1 Sample layouts using height based fitness function

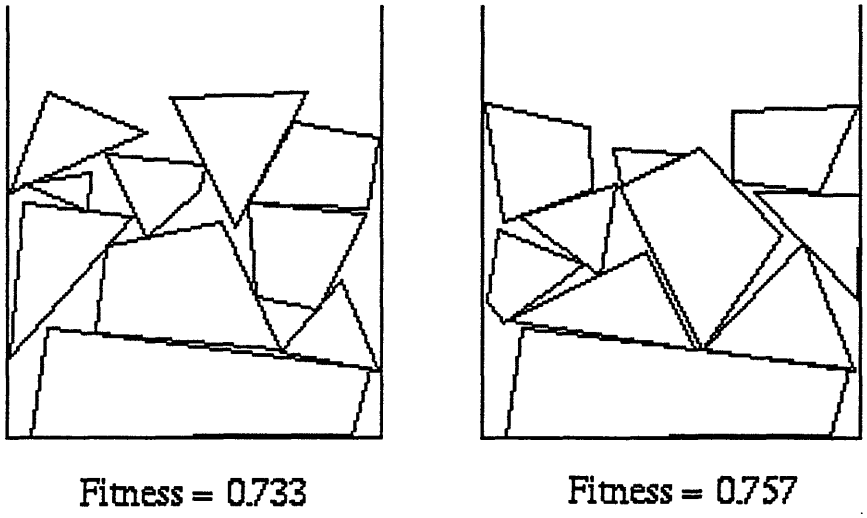


Figure 6.7.5.2 Sample layouts using normalized height as fitness function

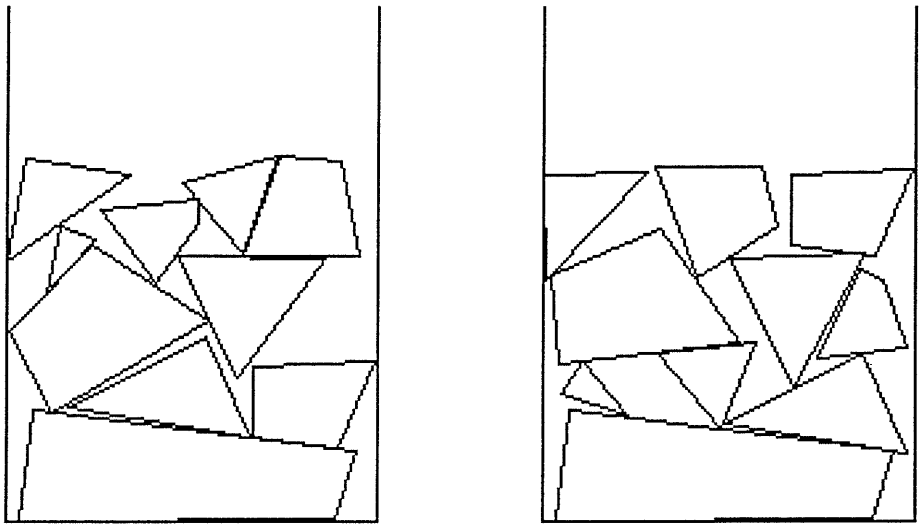


Figure 6.7.5.3 Sample layouts for minimizing waste at each stage

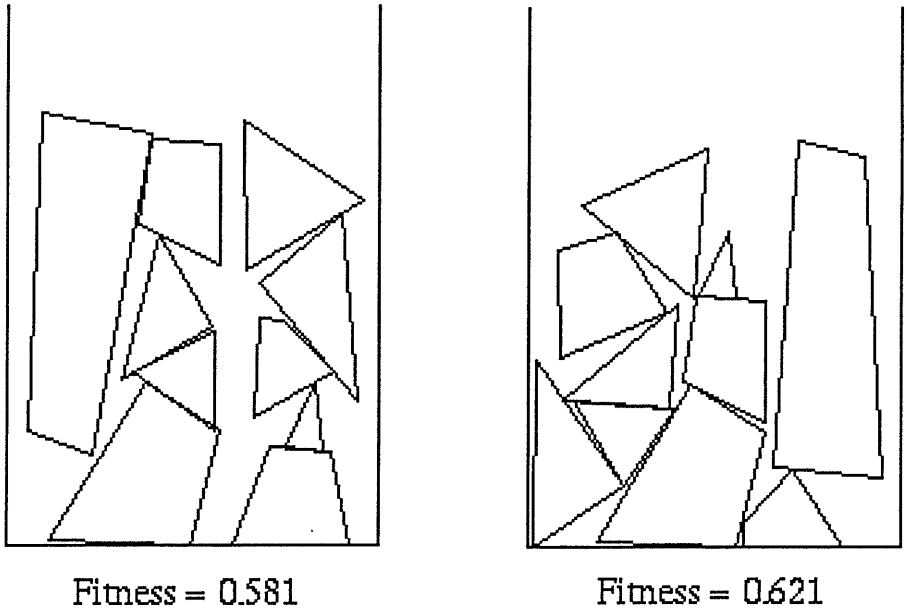


Figure 6.7.5.4 Sample layouts using shadow area based fitness function

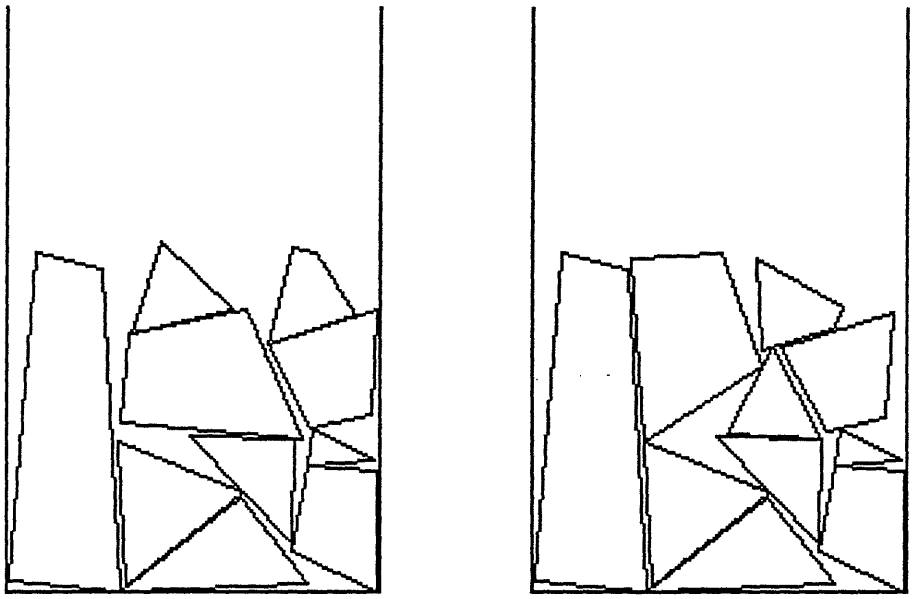


Figure 6.7.5.5 Sample layouts for minimizing height and penalizing shadow area

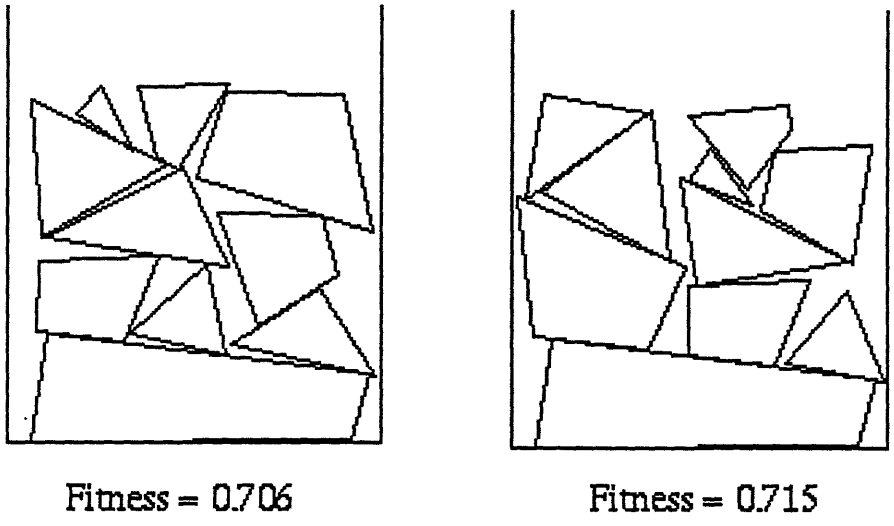


Figure 6.7.5.6 Sample layouts using height based fitness function
with shadow area as constraint

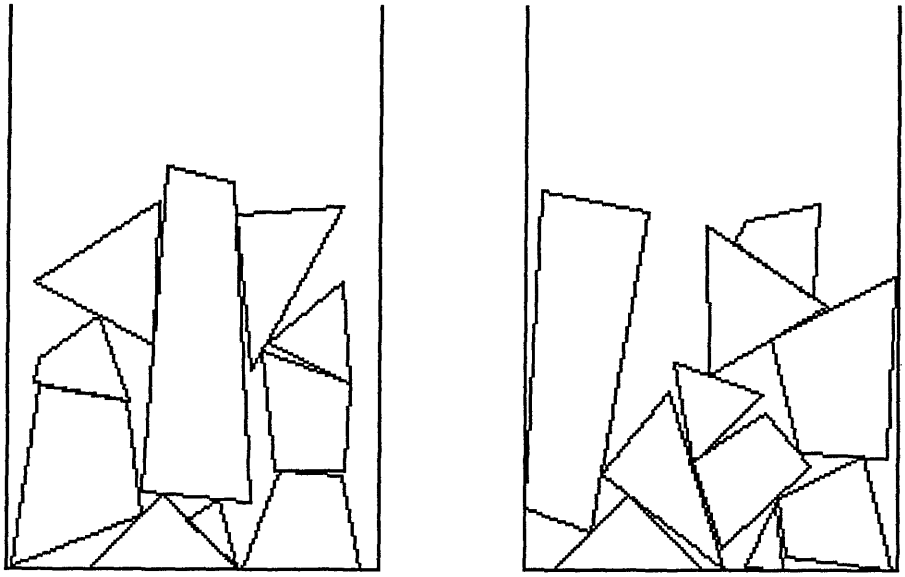


Figure 6.7.5.7 Sample layouts for minimizing shadow area, selecting the lowest location

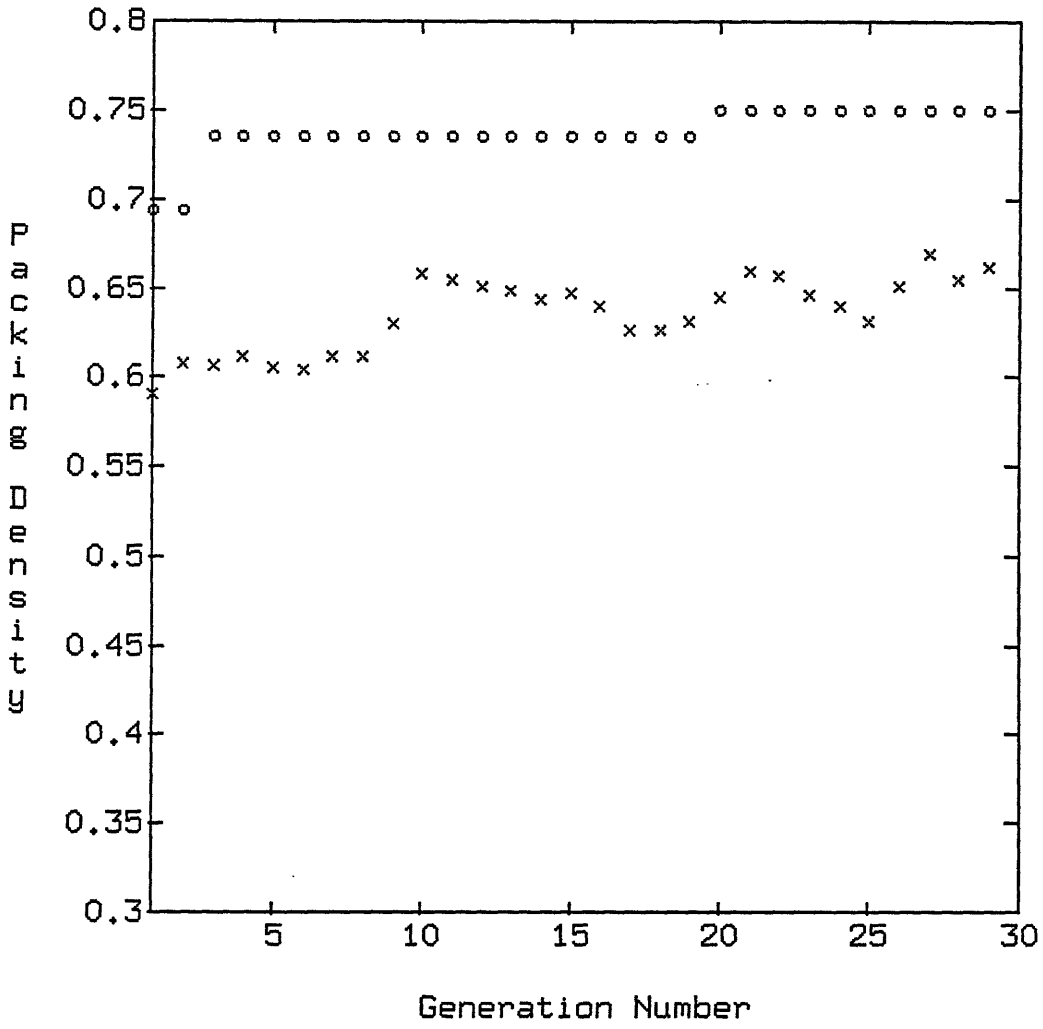


Figure 6.7.5.8 Plot of best-of generation and average-of-generation vs. generation number

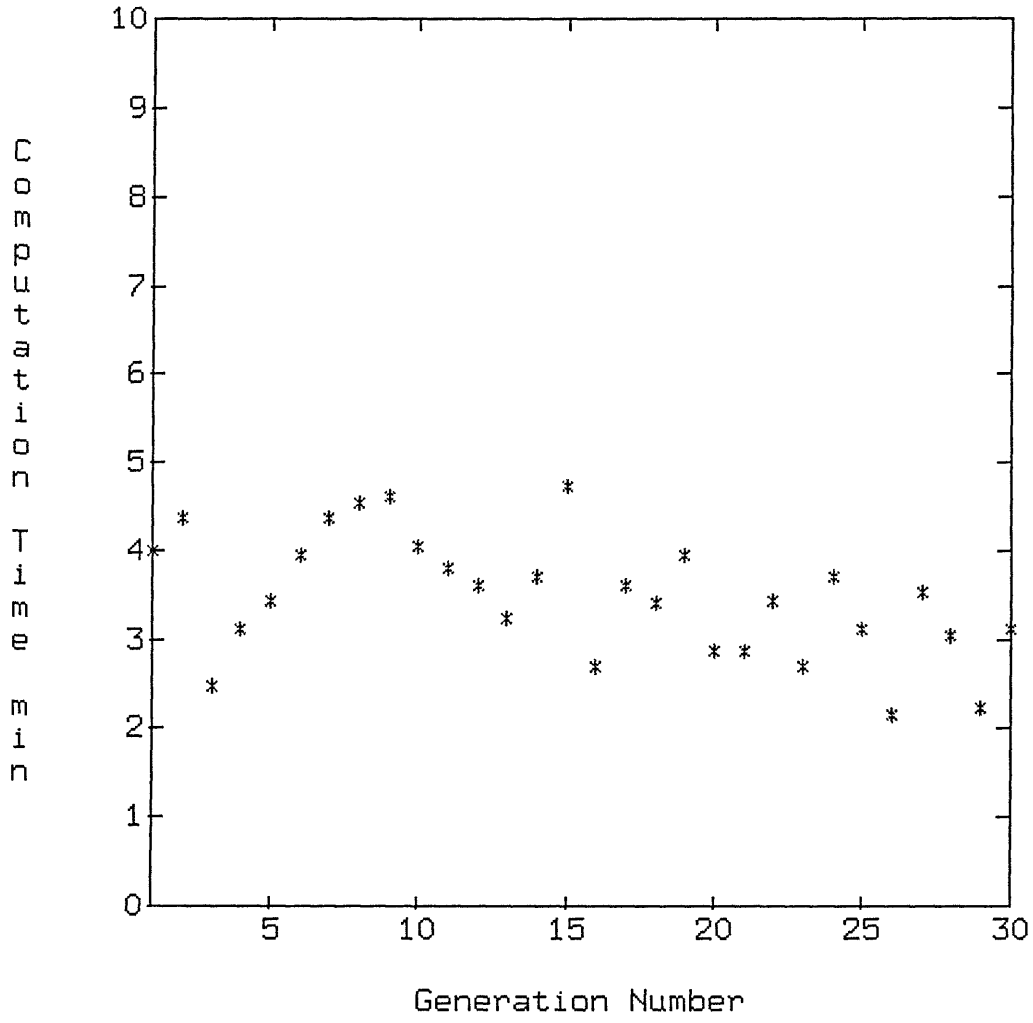


Figure 6.7.5.9 Computation time per generation for String representation

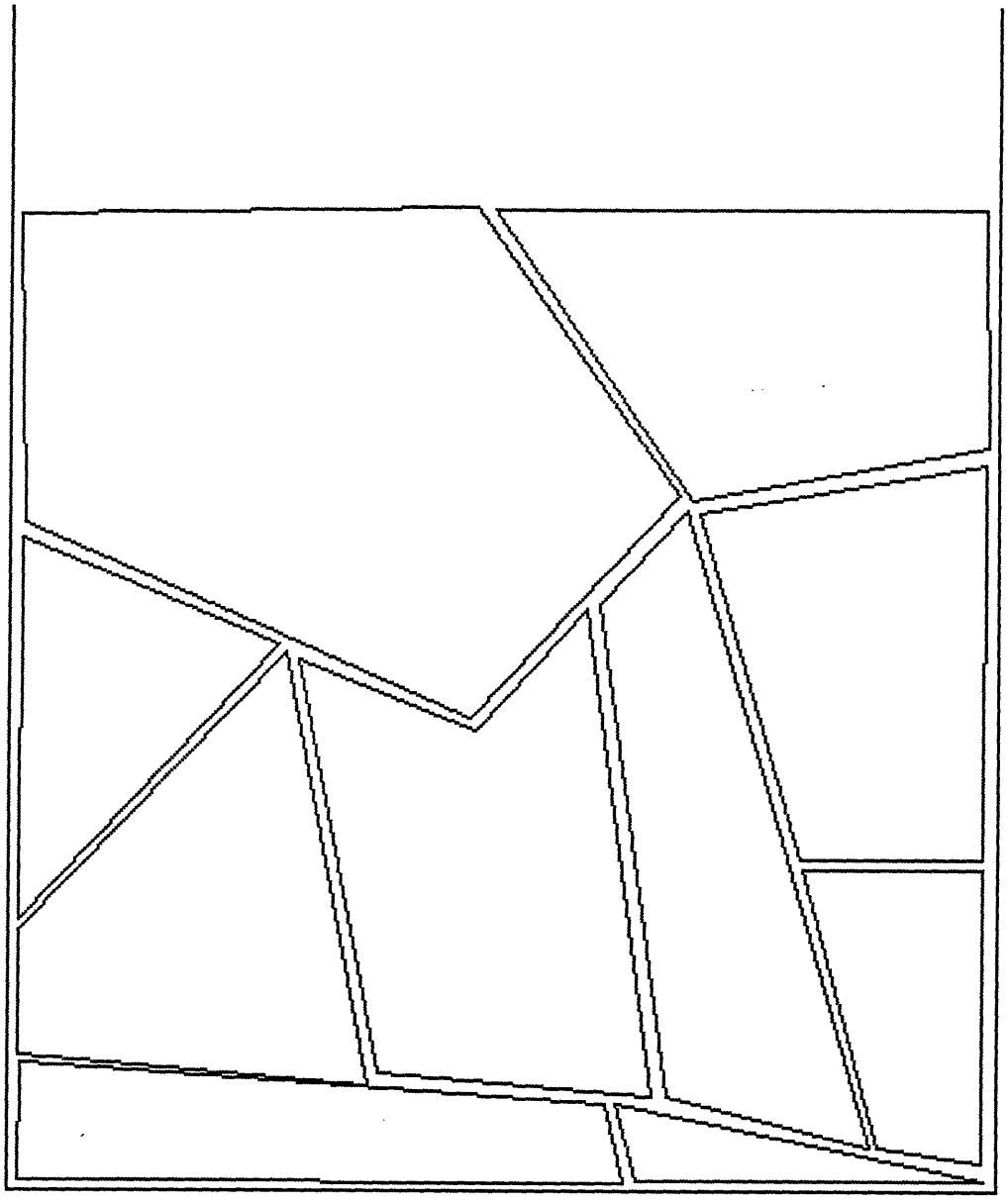


Figure 6.7.6.1 Jigsaw puzzle attempted with string representation

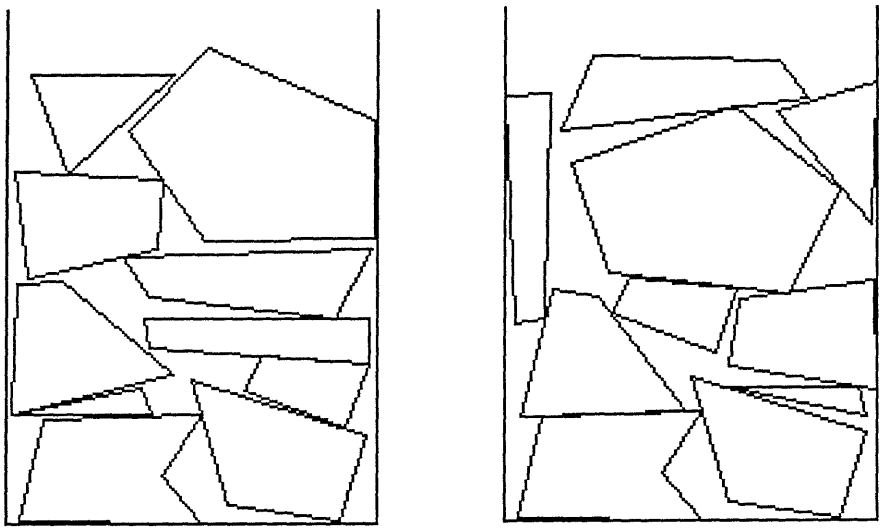


Figure 6.7.6.2 Layout of jigsaw puzzle using string based representation, height based fitness function

Figure 6.7.7 Layout of rectangles using string based representation, height based fitness function

Chapter 7

Incorporation of Human Strategies

7.1 Review of totality and hierarchical approaches

Both approaches described in the previous two chapters, totality and hierarchical, have characteristics that arise from the representations used. The totality approach is very general, in that it has the potential to reach any point in the solution space, except those obviated due to the level of descretization of the range of variables used. However, the representation and crossover operator used are largely ineffective. The hierarchical approach relies on a set of local optimizations to build good solutions. The representation and crossover operators allow the building block hypothesis to hold true for this approach. However, due to the local optimization based approach, the global optimum is not reached in many cases. Also, the local optimization requires a good criterion for the placement of parts.

7.2 Motivation

It has been observed that human pattern nesting experts outperform automated lay-out systems developed so far. The human eye can notice very critical characteristics of the shapes to be placed. In so doing, large amounts of visual information is processed. Often, such observation and information extraction processes are very complex and difficult to describe in a systematic manner that can be implemented as a computer algorithm. As an example, the observation of similarity between two shapes comes easily to humans, but it is a non-trivial task to do so computationally. The success of human experts in pattern nesting could be attributed to this capacity to process and extract visual information. The work described in this chapter is motivated by this success. It is an attempt to understand and model human intuition in order to develop effective pattern nesting algorithms. The work has focused on the problem of creating layouts of parts on a blank of fixed width.

Also, only translations of parts have been considered.

7.3 Human strategies

The approach taken by human experts to pattern nesting can be termed hierarchical or sequential; one part is placed at a time to develop a complete layout. The layouts thus created are partially modified to create better ones. The extent of modification depends on the quality of the layout and the nature of the layout: better layouts tend to be modified less and vice versa. The nature of modifications done to a layout depend largely on the shapes of the parts and their arrangement in the layout. While the overall nature of the placement and modifications strategies is very complex, certain characteristics and trends emerge. These strategies can be broadly classified into three categories: location identification, location evaluation, and sequence search reduction.

7.3.1 Location identification

Consider the placement of part B shown in figure 7.3.1.1 after part A has already been placed on the layout. The part A is considered fixed at the location shown. Human experts generally consider the locations B1, B2, B3 shown in figure 7.3.1.1 for placement. Locations such as B4 and B5 are not considered. We term the locations B1, B2, B3 as "corner locations" or simply "corners". This method of considering only the corner locations for placement seems rather ad-hoc at first glance and one suspects that this could eliminate potentially good layouts. We investigate this placement method and its scope and limitations.

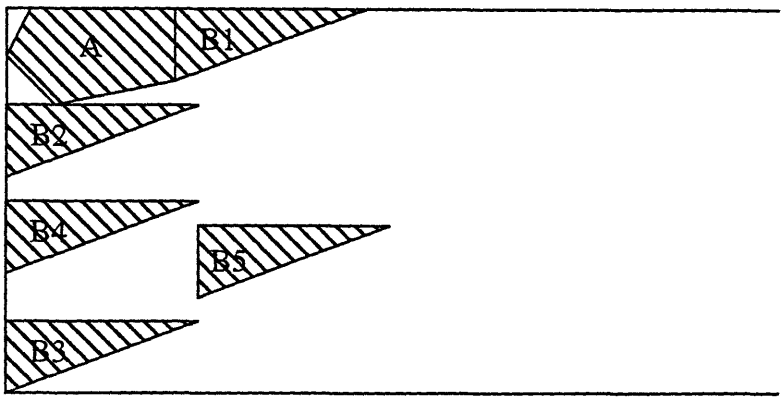


Figure 7.3.1.1 Part placement at corner locations

Consider the trivial, one dimensional problem of placing three rectangles on a blank such that the length of the layout is minimized. We follow the simple strategy of placing the parts in the leftmost location and varying the sequence in which the parts are placed. The sequence ABC can be evaluated by placing the parts sequentially as shown in figure 7.3.1.2. Similarly, all the possible (3!) sequences can be evaluated leading to the same length of layout. In figure 7.3.1.3, we extend our strategy to a more complex one

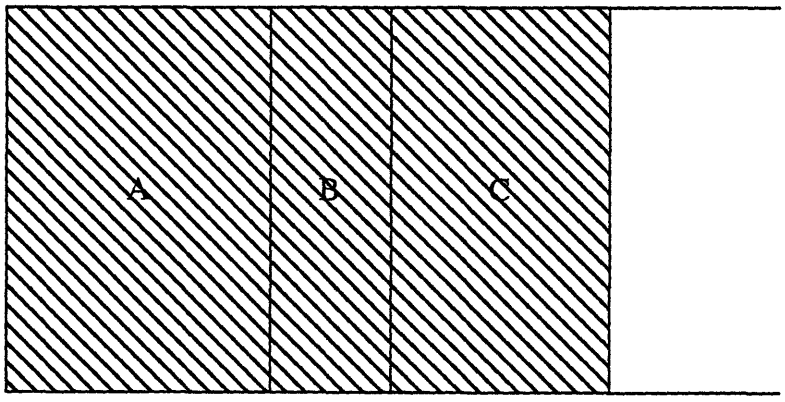
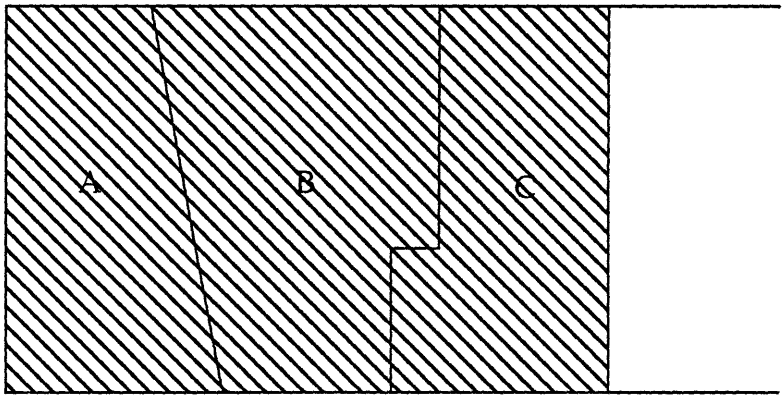


Figure 7.3.1.2 Corner placement strategy for one dimensional nesting problem.

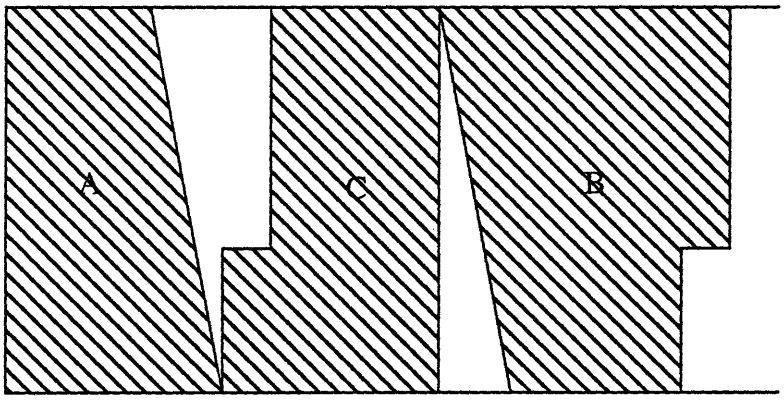
dimensional case. The parts shown have the same width, although they have shape details that require a particular arrangement to achieve the minimum length layout. Consider the evaluation of the sequence ACB. After placing A in the leftmost possible location, C is

placed in the leftmost available location. Part B follows similarly to produce the suboptimal layout shown in figure 7.3.1.3 (b). It would seem that the strategy of placing part C at the leftmost location is the cause of the failure to produce the optimal layout. If part C is placed away from the corner location as shown in figure 7.3.1.3 (c), then part B can be placed in the gap created between A and C to produce the optimal layout. However, we also need to consider other sequences such as ABC. If we follow the simple strategy of placing the parts in the leftmost possible location at each stage, the resulting layout is optimal. The following reasoning explains the success of the strategy:

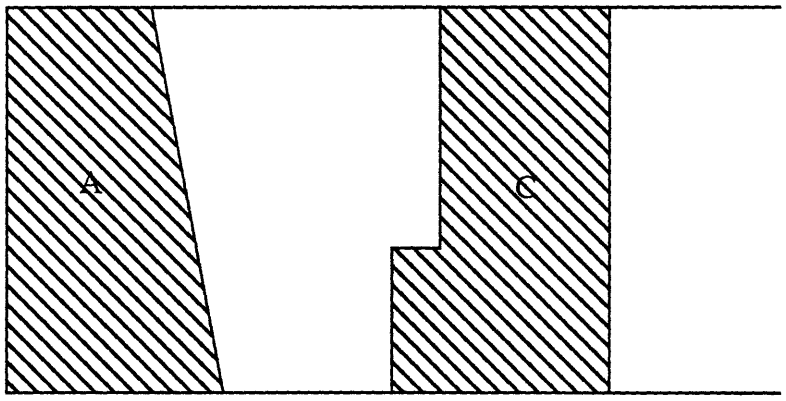
For one-dimensional layout, if a candidate location (l) of a part in the evaluation of a sequence (Si) is not a corner location but corresponds to the global optimum, that location (l) can be reached via a different sequence (Sj) wherein it will be a corner location.



(a) Optimal layout of three parts

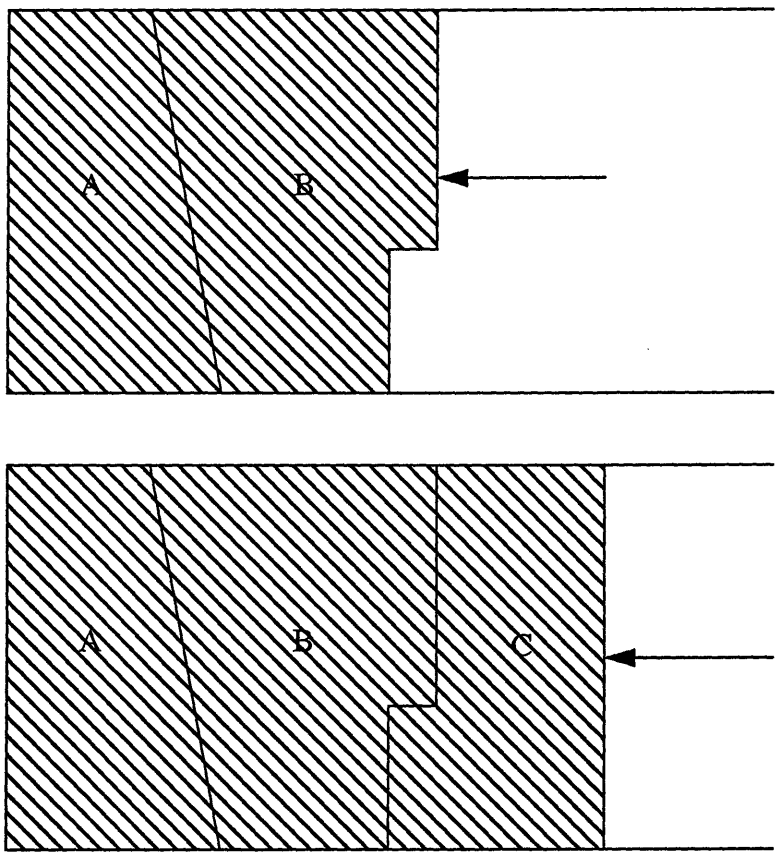


(b) Evaluation of sequence ACB using corner placement strategy



(c) Alternative location of C in the evaluation of sequence ACB leading to optimal layout.

Figure 7.3.1.3 Corner placement strategy for extended one dimensional nesting problem



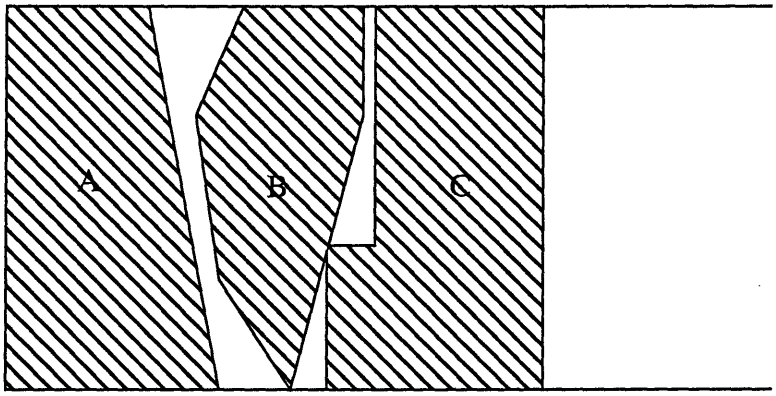
(d) Evaluation of sequence ABC using corner placement strategy leading to optimal layout.

Figure 7.3.1.3 (continued) Corner placement strategy for extended one dimensional nesting problem

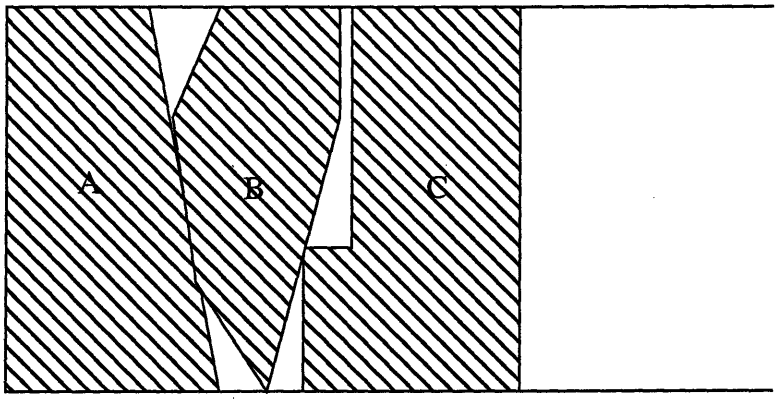
Thus, while placing a part during the evaluation of a given sequence, we need only consider the corner locations. We need to consider one more possibility before extending our reasoning to the two dimensions. Consider the layout shown in figure 7.3.1.4. The location of part B is not a corner location. However, B can be moved to the leftmost (corner) location in the gap between A and C as shown, without affecting the quality of the layout. In fact, B can be moved to any location within the gap between A and C without affecting the quality of the layout. If B is moved to the leftmost location, part C can be

moved to its left, producing a better layout. Thus, the locations of any part that are not corner locations can be reduced to corner locations by moving that part in the leftward direction. If this process is applied to all the parts that are not at corner locations, the layout quality can be improved. Thus, only corner locations need to be considered in placing a single part; other locations do not correspond to the global optimum. The reasoning developed earlier needs to be modified in the following manner:

For one-dimensional layout, if a candidate location (l) of a part in the evaluation of a sequence (Si) is not a corner location but corresponds to the global optimum, that location (l) can be reached via a different sequence (Sj) wherein it will be a corner location, else that location (l) does not correspond to the global optimum, since the reduction of that location to the nearby corner location (l') leads to improvement in the layout quality.



Location of part B away from the corner



Part B moved to local corner without affecting layout quality

Figure 7.3.1.4 Non-corner placement reduced to corner placement without affecting layout quality

For the two dimensional case, we can identify the corners by locally moving the parts to the leftmost as well as the uppermost or the lowermost locations. This is shown in figure 7.3.1.5. Starting from the location B1", part B is moved upward (to location B1') until it comes in contact with the upper boundary of the layout. It is then moved to the left until it comes in contact with part A (at location B1). Starting from other locations and

similarly moving part B, the locations B2 and B3 (figure 7.3.1.1) are reached. The loca-

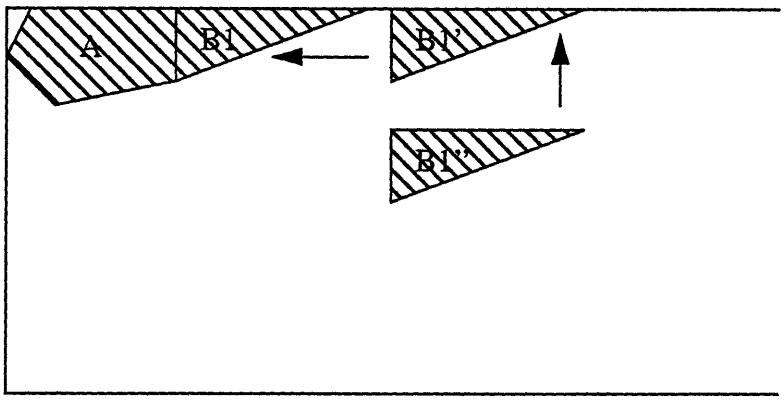


Figure 7.3.1.5 Reaching corner locations by moving along two perpendicular directions.

tions B1, B2 and B3 correspond to locations where the boundary of part B is in contact with at least two different points of the existing boundaries in the layout, such that the normals of the existing boundaries at the contact points are non parallel. This is illustrated in figure 7.3.1.6. Each normal serves to reduce one degree of freedom of placement. Since we are considering only translations, each part has two degrees of freedom of placement. By ensuring contact at two different points, these two degrees of freedom are eliminated. It is as if the part is forced into the corners by applying two forces Fx and Fy, and the existing boundaries provide normal reactions that balance these forces. The normals obviously must be nonparallel to balance the two nonparallel forces applied. Since we allow only translations of the parts and not rotations, we need not consider moment imbalance about the center of mass of the part. As a result of the translational constraint, we assume that sufficient torque is supplied at the center of mass to balance the moment imbalance created by the applied forces and the normal reactions provided by the walls. This process enables us to identify the corners directly.

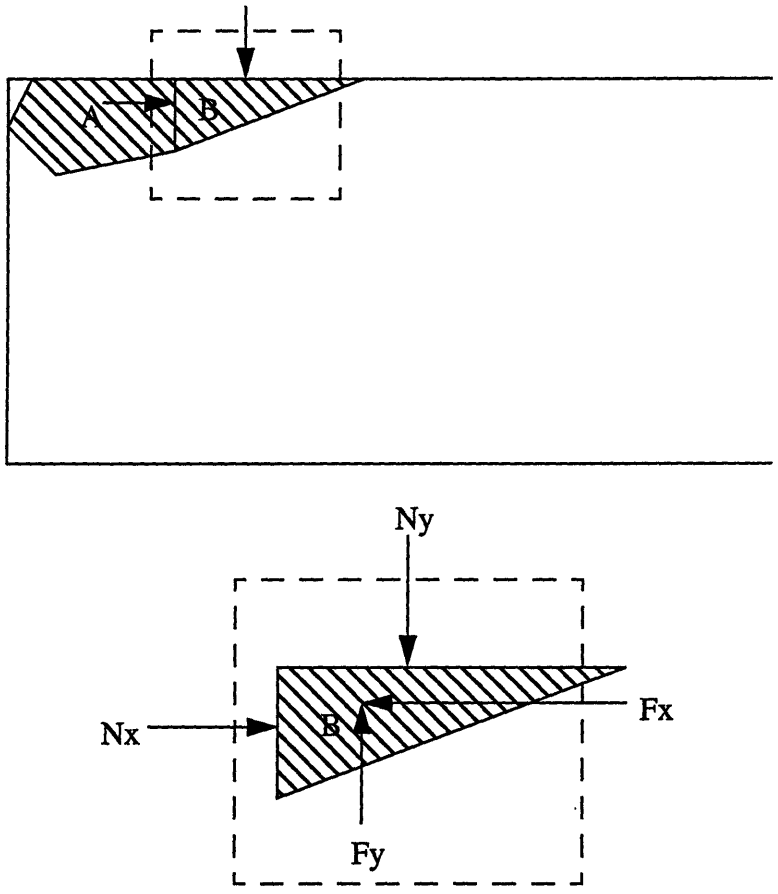
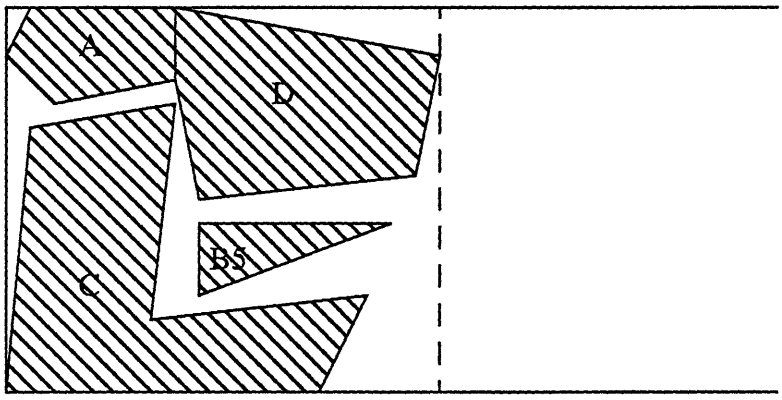


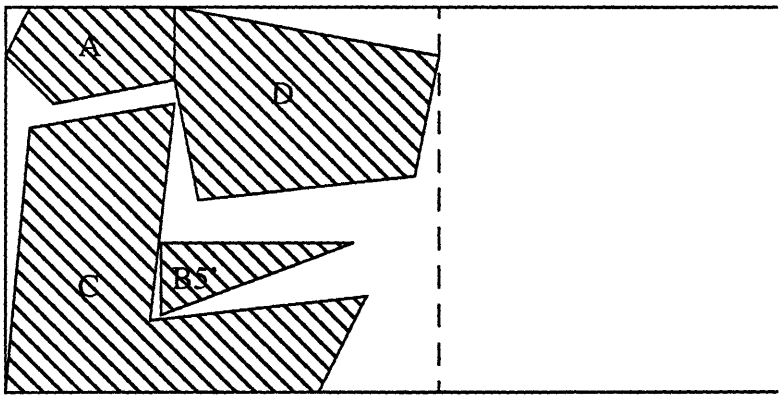
Figure 7.3.1.6 Two non parallel normal forces at contact points on existing boundaries balance applied forces.

We try to extend the reasoning developed earlier to the two-dimensional case. Consider the location B5 in figure 7.3.1.1. This location can be reached after placing other parts before placing part B during the evaluation of a different sequence. For example, the sequence ACDB leads to the location B5 (see figure 7.3.1.7 (a)). Notice that B5 is not a corner location in that sequence. We can move part B from B5 to reduce that location to a corner location (B5') as shown in figure 7.1.3.7 (b). Notice that the quality of the layout is unaffected. This is true since the part D governs the length of the layout, not part B. If the shape of part D is different as in figure 7.3.1.7 (c), the layout length is governed by B. In this case, notice that we can actually improve the layout by moving B to a nearby corner from B5. Thus, we can see that in the evaluation of a particular sequence, a location that is not a corner location, can be ignored since it will occur as corner location in a different

sequence, or can be reduced to a corner location either without affecting the quality of the layout or actually improving the quality of the layout. There are

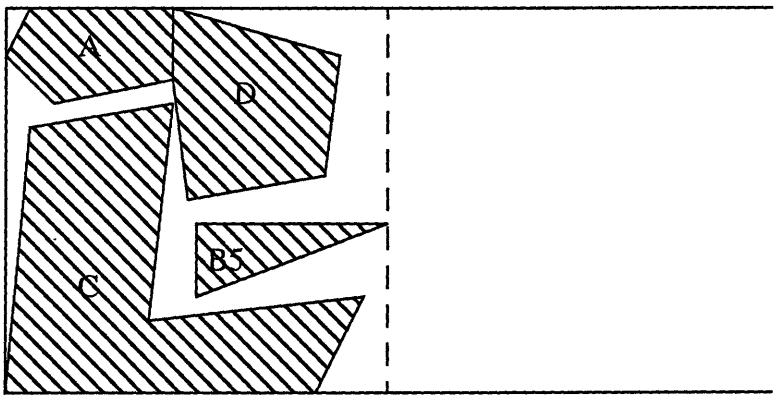


(a) Non corner location of part B.

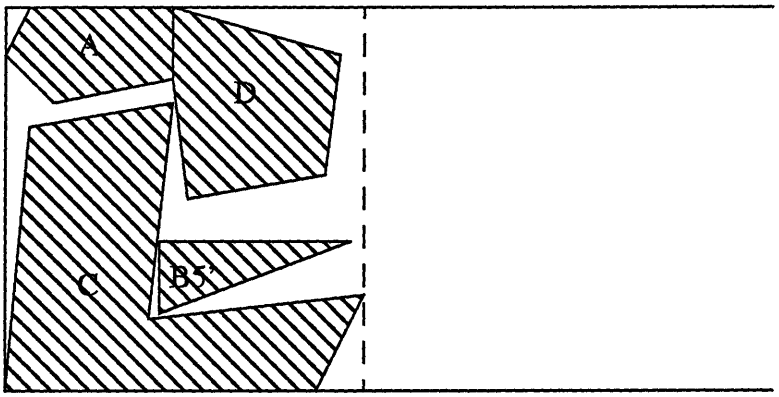


(b) Part B moved to local corner location; layout quality is unaffected.

Figure 7.3.1.7 Non corner locations reduced to corner locations for two dimensional nesting problem. Strictly leftward part movement ensures layout quality is unaffected or improved.



(c) Non corner location of part B for a different shape of part D.



(d) Local corner location for part B; layout quality is improved since movement is strictly leftward.

Figure 7.3.1.7 (continued) Non corner locations reduced to corner locations two dimensional nesting problem. Strictly leftward part movement ensures layout quality is unaffected or improved.

exceptional cases where placement of a part at a corner location prohibits another part from reaching a desired location in the layout. As an example, the optimal layout shown in figure 7.3.1.8 cannot be reached using a corner placement strategy. However, such cases are rare and most good layouts can be reached using corner placement.

This reasoning about the corner placement strategy provides an important simplifi-

cation:

Out of the infinite possible layouts, the search can be restricted to a finite number of layouts, generated by considering only the corner locations while placing each part.

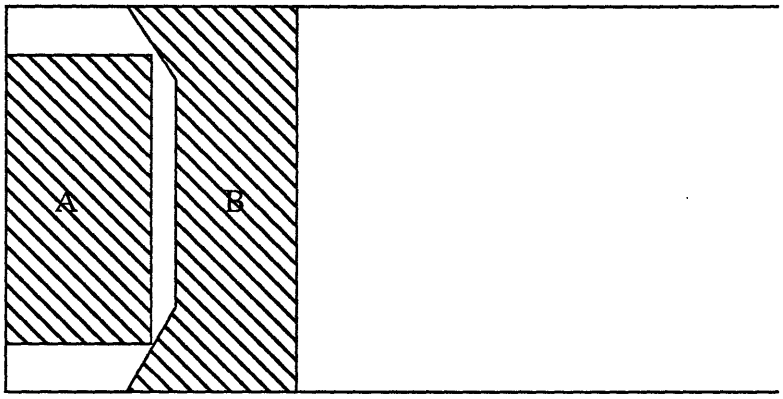


Figure 7.3.1.8 Optimal layout unreachable using corner placement strategy

Thus, the possibilities for each sequence can be denoted as: (A1,A2,...) (B1, B2,...), etc. There are n! possible sequences for n parts. The number of possible layouts cannot be predicted; this number depends on the actual shapes of the parts. A number of corners are usually available for placing a single part.

In figure 7.3.1.9, a simple jigsaw puzzle is solved using this corner placement strategy, following the sequence DACB for placing the parts. For placing part D, two corners are available. If the upper corner is chosen, two corners are available for the placement of part A. Choosing the lower corner creates three corners for the placement of part C. If the lowest corner is chosen, two corners are available for the placement of part B. The upper corner corresponds to the global optimum. Thus, following the sequence DACB solves the jigsaw puzzle, if the appropriate corner is chosen at each stage. Notice that other sequences (DBAC, DABC, ACDB, ADBC, ADCB) lead to the same result if the appropriate corner is chosen at each stage. Also, for the placement of each part in any sequence, an infinite number of locations are available; by focusing on corner locations, the optimal layout can be created very quickly.

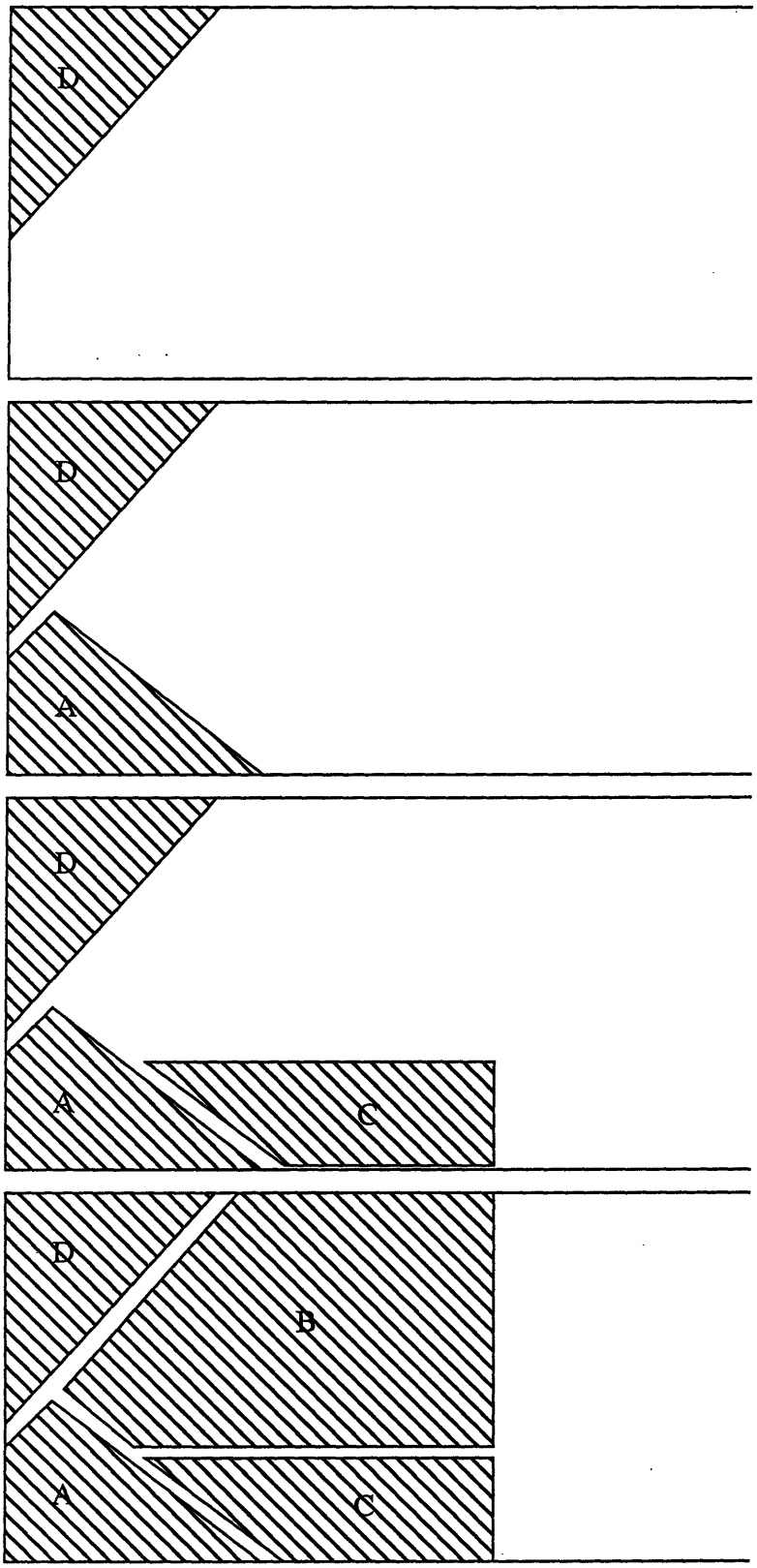


Figure 7.3.1.9 Jigsaw puzzle solved using corner placement approach

7.3.2 Location evaluation

It is impossible to evaluate each possible layout, even when the corner placement strategy is used. Out of the several corners possible at each stage, it is often useful to consider only one location. Humans choose locations that reduce gaps between part boundaries in order to produce tightly packed layouts. In this section, we describe a model can be used to reduce gaps between boundaries of parts and minimize the waste area that is created in part placement.

Consider again the corner locations shown in figure 7.3.1.1. Out of the three locations shown, the location B1 appears to be a good choice in order to produce a tightly packed layout. This choice is very intuitive to human eye if the sole criterion is to place part B at a location where the gaps between the part boundaries are minimized. However, we need to quantify this intuition in order to use it in an automated nesting system.

While placing part B on the layout, it is important not to create gaps (between boundaries) that cannot be occupied by parts that are to be placed subsequently. The areas that cannot be accessed by the subsequent parts can be treated as waste areas. A simple measure of this waste is not obvious if the shapes the parts to be placed subsequently are not known. Thus, we develop a measure based on the probability of the surrounding areas being occupied later. Consider the points P1, P2 and P3 in the layout shown in figure 7.3.2.1. After placing part A at the location shown, the likelihood of the region around point P1 being occupied by subsequent parts appears to be lower than that of the region around point P2. This could be attributed to the fact that point P1 is surrounded (although not from all sides) by part boundaries closer to it compared to point P2. If a subsequent part is to occupy the region around point P1, that part must have regions that fit between the boundaries surrounding P1. Similar reasoning holds for point P2. At this point, we cannot make any assumptions about the sizes and shapes of the parts that are to follow; we can only make a reasonable guess about which of the points is more likely to be occupied.

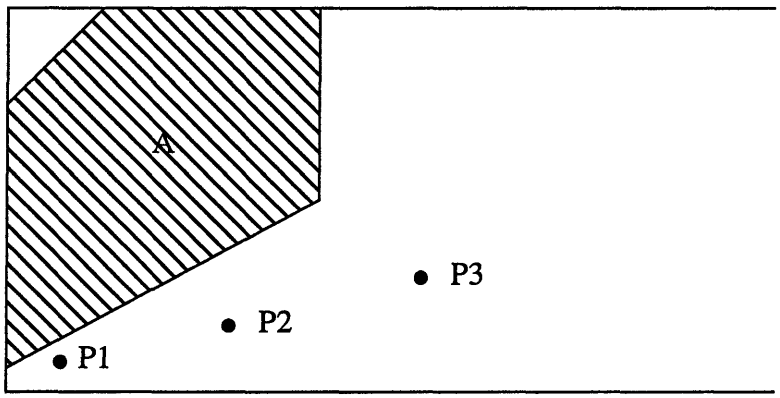
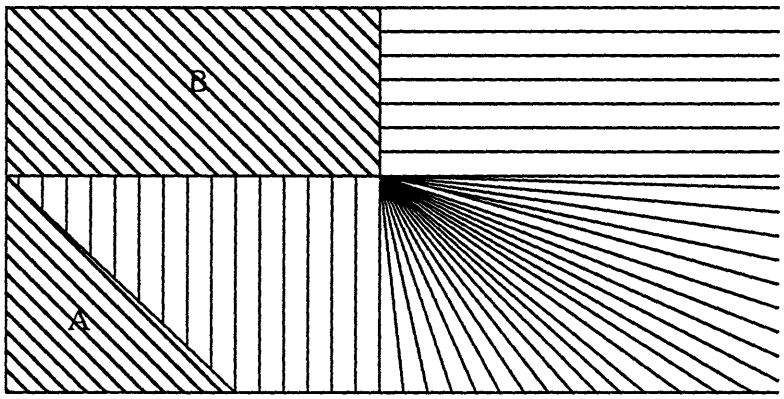


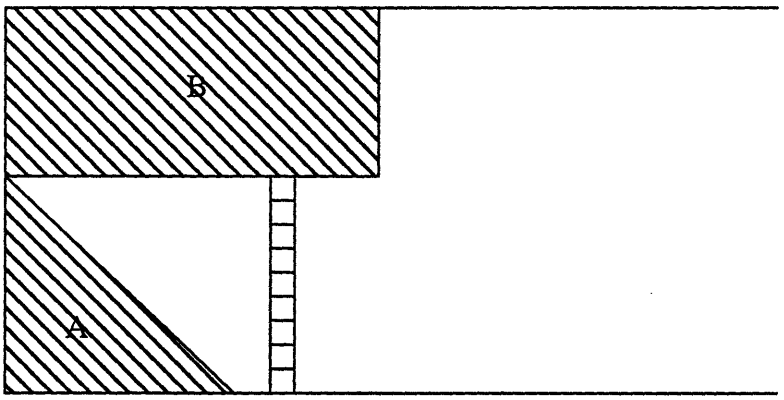
Figure 7.3.2.1 Points on the layout with different accessibility.

Placement of a part at any location can be considered to affect the likelihood of the regions close to its boundaries being occupied by subsequent parts: the regions closer to the boundaries are more adversely affected. Thus, the placement of any part creates an effect of inaccessibility of areas near its boundaries in the layout. The regions covered by the part, of course, are absolutely inaccessible to any subsequent parts. We estimate the total waste by taking a weighted sum of the areas surrounding the part boundaries. This is illustrated in figure 7.3.2.2. Part A is already placed on the layout and the placement of part B at the location shown is considered. Moving away from the boundary of part B, area elements of equal width are created until another boundary (of the layout or another part) is reached. The width of these area elements is chosen based on the characteristic part size[1]. A similar process is carried out starting from the vertices of the part boundaries, using cylindrical area elements.

1. Choosing area elements with dimensions smaller than 1/100th of the characteristic parts size can provide sufficiently accurate resolution.



Areas around the part boundary contribute to waste area.



Area elements start from the boundary of the part.

Figure 7.3.2.2 Creation of area elements to quantify waste area.
Weights attached to elements decrease with distance from the boundary of the part being placed.

Each of these area elements is considered to contribute a decreasing amount to the total waste created by the placement of part B. The contribution to waste or the penalty factor of each area element is obtained from the function e^{-X} as shown in figure 7.3.2.3. The distance from the boundary has been normalized by a characteristic part size to maintain uniformity of the penalty factor regardless of the actual part sizes[1]. A maximum penalty of 1.0 is attributed to the elements next to the boundary of part B. The function approaches zero asymptotically since a small effect of the boundary can be assumed to be felt even at large distances from it. A linearly decreasing penalty function would reduce to

1. This characteristic part size can be obtained in several ways. One possible method is to use the square root of the average of the areas of the parts.

zero at a particular value of the normalized distance. The choice of this value would have to be arbitrary. To avoid this, a function that approaches zero asymptotically is chosen.

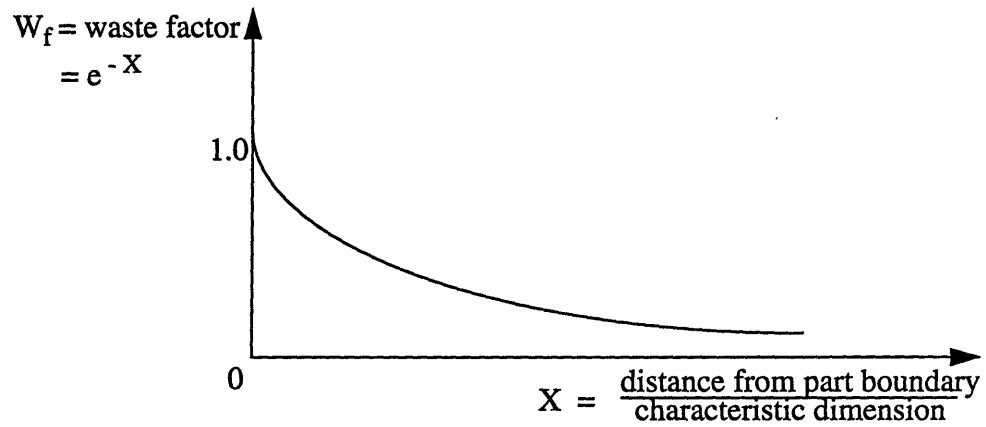


Figure 7.3.2.3 Waste evaluation function

This evaluation can explain human intuition about the choice of location B1 over B2 and B3 as shown earlier in figure 7.3.1.1. The location B1 appears to create less waste between boundaries as compared to B2 and B3. Thus, from the candidate corner locations for the placement of a single part, we choose the one that minimizes the waste calculated in the manner described above. This restricts the search to strictly n! possible layouts. Revisiting the example illustrated in figure 7.3.1.9, notice that at each stage, the corner location that appears to intuitively reduce the gaps between part boundaries is chosen. For example, from the two possible corners for part D, the choice of the upper corner is intuitive. The measure of waste area developed quantifies this intuition about placement and facilitates the automation of the process of pattern nesting.

For the jigsaw puzzle solved in figure 7.3.2.4, the selection of a corner from those available for each stage using the waste quantification does not lead to the optimal layout shown. The sequences (with appropriate corner selection) that can lead to the optimal layout are the following: DACB, DABC, DBAC, ACDB, ADBC, ADCB. If part D is placed first, the upper corner creates less waste than the upper corner, although the lower corner corresponds to the global optimal. Similar reasoning holds true if part A is placed first. Thus, for a given set of shapes, if the global optimum is reachable using the corner place-

ment strategy, restricting the placement to a single corner location (that corresponds to the

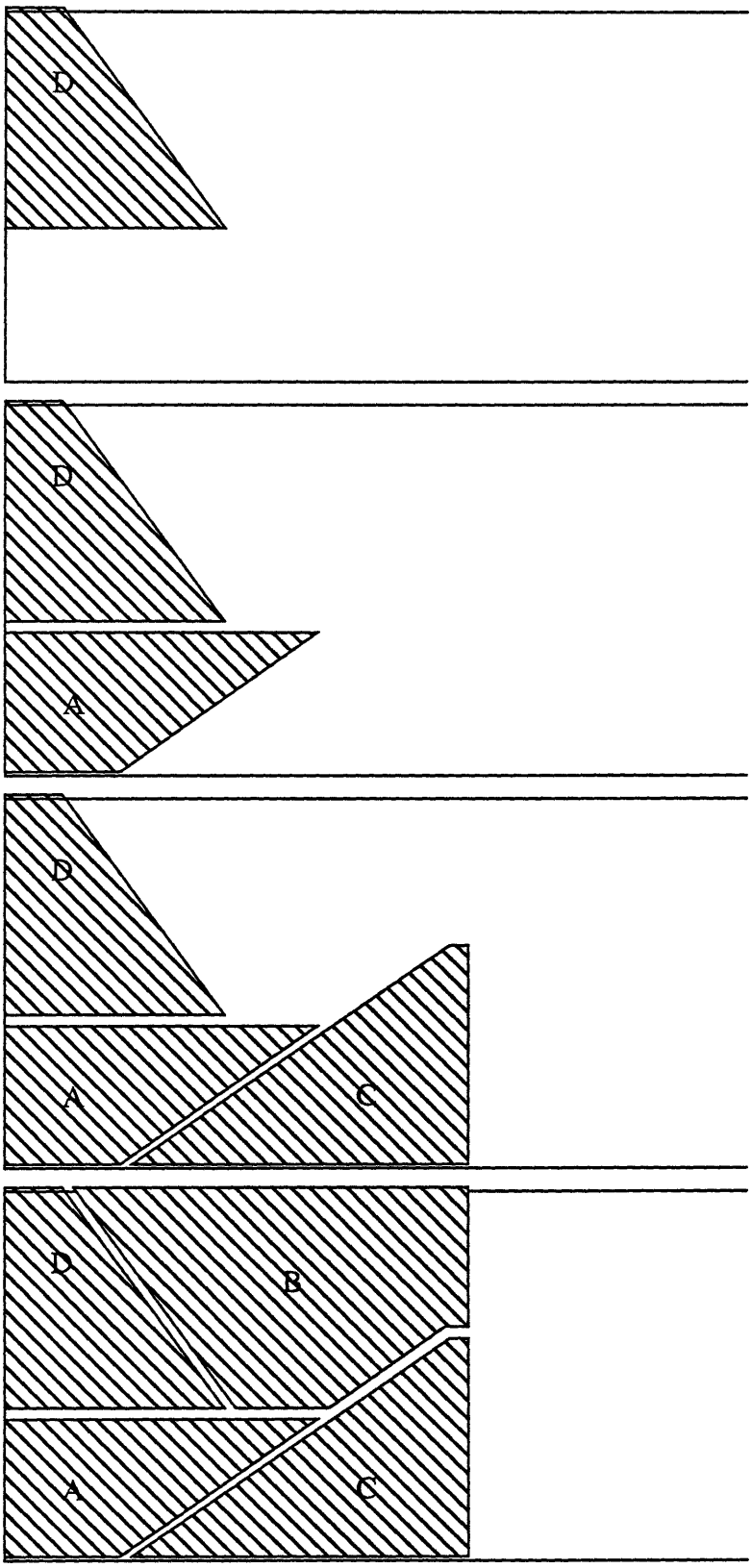


Figure 7.3.2.4 Jigsaw puzzle solved using corner placement approach

lowest amount of waste as defined earlier), can prevent the search from reaching that optimum. However, by choosing the location that corresponds to the minimum amount of waste, we ensure that tightly packed layouts are produced. This allows good layouts to be produced quickly.

7.3.3 Sequence search reduction

By placing parts at corner locations the search is limited to a finite number of possible layouts. By selecting one corner of those possible for the placement of each part, the search is restricted to n! sequences. Genetic algorithms can be used as the basic search method to find the sequence that corresponds to the minimum length of the layout. The chromosome used for this optimization is the same as that developed for the string based representation (along with the crossover and mutation operators) described in chapter 6. This genetic search can be augmented by incorporating sequence search reduction strategies that are motivated by the approach of human experts. By using these strategies, human experts evaluate very few sequences and layouts. Some of the strategies can be easily adapted into the probabilistic framework of genetic algorithms, while many are very complex. In this section, the incorporation of simple strategies is described. As mentioned earlier, humans are aided by large amounts of visual information processed very rapidly. Such information is unavailable to computer algorithms, since the acquisition and processing of this information is a difficult task. It is necessary to rely on the trial and error of genetic algorithms to make up for the lack of useful information. The incorporation of the search reduction strategies described in this section is probabilistic, and simple quantities such as length, area related to the parts are used for estimating relevant probabilities.

7.3.3.1 Biased Initial Population

Heuristics can be used to create the initial population for a genetic search. An example of such heuristics is placing large parts earlier. Instead of classifying the parts as large and small, the initial population is created by first assigning a probability to the parts proportional to their area, or a characteristic dimension. After assigning a probability to the parts, a roulette-wheel selection is used to select and order the parts; larger parts have a higher probability of being selected. Of course, after selecting a part, the next roulette-

wheel selection is limited to the remaining parts.

Other heuristics such as pairing of parts that occur in mirror images of each other can be used similarly. Different heuristics are expected to work well in different application domains. If a heuristic is known to work well in a particular domain of application, it can be similarly incorporated using probabilities calculated based on simple quantities and characteristics. Heuristics alone cannot be guaranteed to produce good layouts. However, they can be used as good starting points for a genetic search.

7.3.3.2 Selection of crossover point

Human experts tend to modify layouts based on their quality: layouts that are longer are modified more and vice versa. This strategy is incorporated by choosing the location of the crossover point in the chromosome based on the quality of the layout. The crossover point has a higher probability of being closer to the left end of the chromosome for a longer layout and vice versa. This probability is calculated as follows:

The length of the layout corresponding to the chromosome selected for crossover is known. A factor based on the length termed *crossover bias factor* (CBF) is calculated.

CBF = L/Lm;

where,

L = Length of the layout;

Lm = Length of the minimum area layout possible.

Length of the layout based on a nesting efficiency of 1.00 is the minimum length possible. This minimum length can be used as a normalizing factor. The probability of the crossover point being placed at the leftmost end of the chromosome is CBF times the probability of the crossover point being placed at the rightmost point in the chromosome. The probability of selection of the crossover point at locations between the two ends reduces linearly from the leftmost to the rightmost end of the chromosome. In this manner, the selection of crossover point can be biased, yet probabilistic. A deterministic selection of the crossover point based directly on the length of the layout can eliminate some possi-

bly good candidate solutions.

7.3.3.3 Estimation of layout length

It is often possible for a skilled human to quickly estimate the expected length of a good layout for the given shapes by inspecting the shapes. This judgement is developed by experience in an application domain. For an automated system, such estimation is not possible. However, during the optimization process, the best layout obtained at any stage can be used as a guideline. Such estimation is important since it is then possible to modify the criterion for selection of a corner location to account for the expected length. This is shown in figure 7.3.3.3.1. The part G is the last part to be placed and the location G1 creates less waste compared to the location G2. However, at this stage, it is important to limit the length to the minimum. This can be achieved if the evaluation function has a term to account for the estimated length of the layout:

F = Minimize $(A_w + P*(L - L_e)/L_e)$;

where,

A_w = Area waste created at the location considered;

L = Length of the layout if the location considered is chosen;

L_e = Estimated length of the layout;

P = Penalty factor for exceeding the expected length.

$= 0$ if $L < L_e$.

This evaluation function restricts the placement of parts to the area limited by the expected length. The smallest layout length obtained up to that point can be used as the expected length.

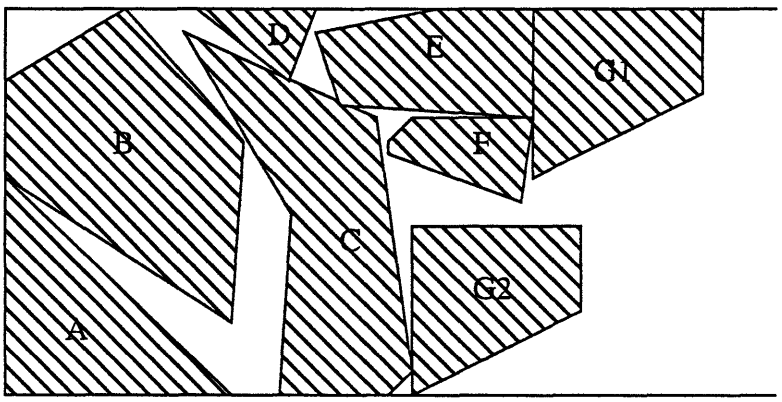


Figure 7.3.3.3.1 Minizing length as well as waste area towards the completion of layout.

7.3.3.4 Other strategies

Several other strategies are used by human experts to limit their search to relatively few layouts. An example of such strategies is the exchange of similar parts within the layout to reduce the length. This exchange of parts requires identification of similarities between the patterns to be exchanged and also the ability to reason about the shape details. Another strategy is to maintain tightly packed groups of parts in the layout without alteration. A measure for the extent to which parts are tightly packed could be very useful, although such a measure is not obvious. Such strategies are difficult to automate and their potential benefits may not justify the computational effort.

7.4 Example layouts

In this section, layouts generated by the automated system that uses the human strategies described so far are illustrated. Layouts are created sequentially: the automated system places one part at a time on a blank of given width. All the corner locations available for placement at each stage are identified. These locations are compared with each other based on the waste quantification model developed above. The corner location that minimizes the area waste is selected for placement. Each layout is created by placing all the parts sequentially. Genetic algorithms, enhanced by the incorporation of the sequence search reduction strategies, are used to search for the best sequence of part placement.

The automated system comes close to matching the performance of human experts. The layouts illustrated in figures 7.4.1 and 7.4.2 were generated using this automated system. The part shapes are used in an industrial application, and were also nested by human experts. Figures 7.4.1 show layouts of 28 parts generated by the system over 2 hours of computation on a 100 mips workstation. A layout expert achieved an efficiency of 80.02% for the same data set. Figures 7.4.2 show similar results for a larger data set (36 parts) over 8 hours of computation on a similar workstation. Again, the system comes close to the efficiency achieved by an expert, 82.41%.

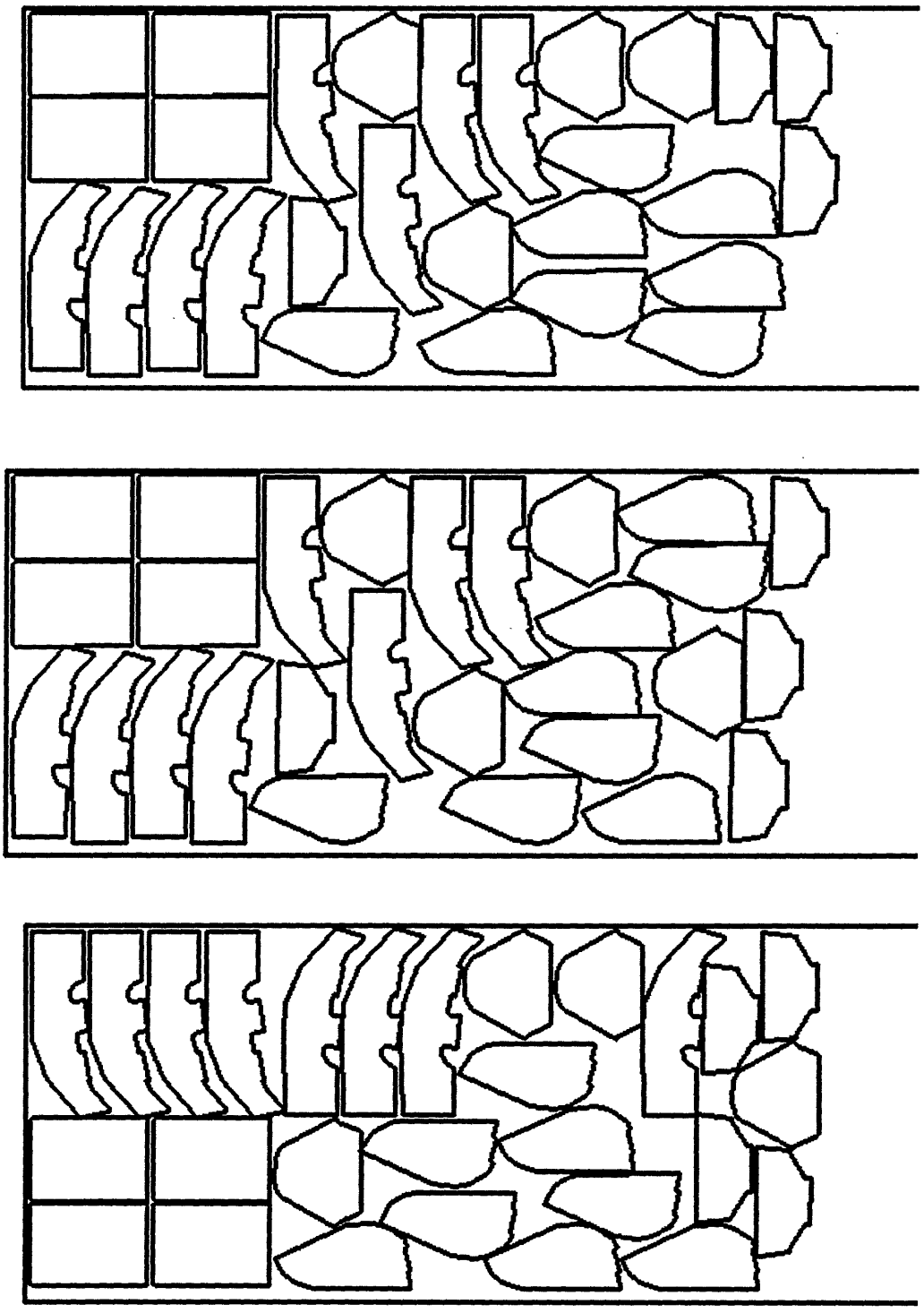


Figure 7.4.1 Layouts created by the automated pattern nesting system (28 parts)

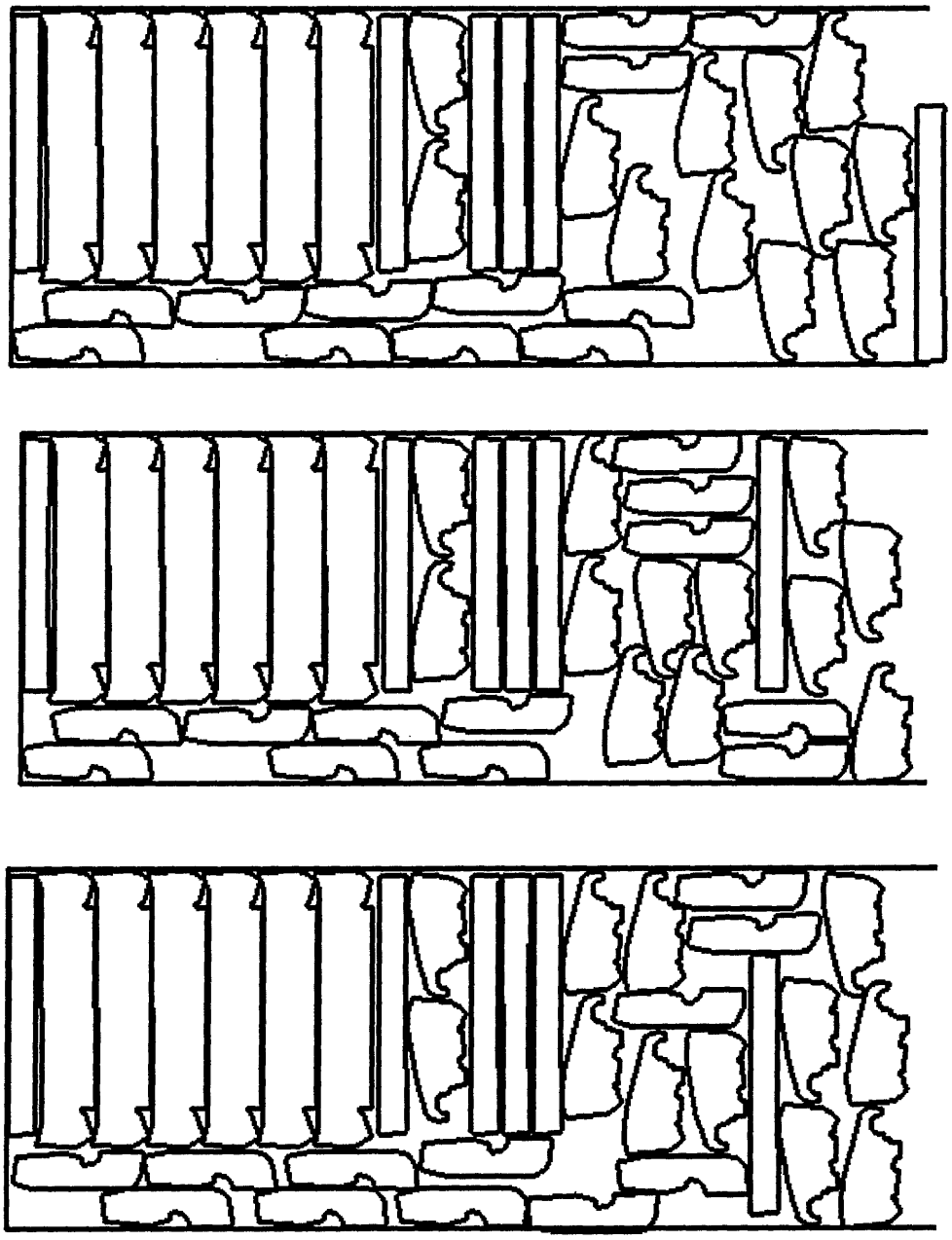


Figure 7.4.2 Layouts created by the automated pattern nesting system (36 parts)

Chapter 8

Conclusions and future work

8.1 Conclusions

Pattern nesting is a problem of industrial and commercial importance. Although the specific requirements for each application are different, the main concern remains the same: occupation of the least amount of space. Due to the complexity of the problem, it is necessary to use general, probabilistic search methods to obtain good solutions with minimum computational effort. The basic philosophy of genetic algorithms proves effective to this end: the search effort is concentrated probabilistically in the regions of the solution space that appear promising. This is achieved in moving from one solution to another by maintaining some of the characteristics of the original solution in a resulting solution. For pattern nesting, the relative arrangements of some of the parts in a layout can be retained in layouts resulting from it. The arrangement of the remaining the parts is varied to sample a nearby point in the solution space. The hierarchical representation and the appropriate crossover operators facilitate this process. The lower level of optimization selects only valid locations in the layout. Reduction of repeated computation is possible with such a representation, since a meaningful schema of the higher level chromosome results in the same arrangement of parts in any chromosome. Also, via the lower level of optimization, the parts are placed at locations where the gaps created between the part boundaries are minimum.

Human experts limit the part placement to certain key locations which we term as corners. This placement process can be justified intuitively by examining one dimensional cases and by extending the reasoning to two dimensions. The corner locations of parts can be compared against each other on the basis of the waste area created between them. Again, the intuition of human experts can be modeled in a simple manner to effectively quantify the waste areas. It is possible to develop a model that is very general and does not

make any assumptions about the specific part shapes. This effectively restricts the search to a finite set of solutions enabling the creation of good layouts in a computationally effective manner. Genetic algorithms provide an effective probabilistic framework for the search of sequences of part placement that lead to good layouts. The basic genetic search can be augmented by probabilistically incorporating human strategies that depend on very simple and general quantities such as length, area, etc. The probabilistic incorporation of these strategies allows the occurrence of good solutions that are cannot be reached by such strategies. The use of simple part characteristics to evaluate probabilities maintains generality of approach and domain independent applicability. Complex strategies that depend on inspection of part shapes cannot be modeled effectively; it is essential to rely on trial and error.

This thesis offers the first known models of human intuition for pattern nesting. These models are a step toward matching human performance in pattern nesting. Of the pattern nesting systems reported so far, the use of human expertise has been limited to the symbolic level. e.g., classification of parts into a few categories, and the subsequent placement strategies that depend on the use of such classification. The pairing of parts that occur as mirror images of each other is another example of incorporation of human strategies only at a symbolic level. Such incorporation of human strategies is limited in its scope and applicability. The choice of location for a part in a layout is difficult to describe by the use of a vocabulary such as "concave with convex", "small with large", etc. The model of waste area between part boundaries is very general. This model works at the very basic level of relations between part shapes pertaining to the area gaps between their boundaries. Since the model depends on a large number of points around the boundary to quantify the waste area, the model accounts for minute shape details.

The application of human strategies in a deterministic manner limits the type of layouts that can be generated; only those layouts that can be described by a limited vocabulary are generated. With probabilistic incorporation of human strategies, it is possible to achieve good layouts that cannot be described by a set of heuristics. The philosophy of genetic algorithms is retained in the incorporation of human strategies: the exploration of

the solution space is concentrated in the regions that show promise. For problems of the nature of pattern nesting, this intuitively is the best philosophy, lacking knowledge of provable facts about the problem domain.

8.2 Future Work

Several extensions of this work are possible. In the immediate future, additional search reduction strategies can be incorporated. Also, a method to identify tightly packed subgroups can be developed. These subgroups would be retained probabilistically in subsequent layouts with the possibility of effectively reducing the search space. The effectiveness of each individual sequence search strategy described in chapter 7 can be studied.

In the present system, part locations are selected based on the waste area quantification. The best location according to this criterion is selected. Some of the other corners that also create tightly packed layouts could be considered for placement. The symbolic chromosome could be augmented to specify the choice of location along with the sequence of parts. Again, the selection of location could be made probabilistic, with the probability of selection being governed by the amount of area waste created.

The current system works under translational constraints. The generalization of the identification of corner points when parts can rotate would be interesting. It would be interesting to see if the idea of force balance can be extended to moment balance when the parts can rotate. Since the waste area quantification is not restricted to part placement under translational constraints, it could be used directly even when parts are rotated.

Extensions to three dimensions would be similar. As discussed in chapter 6, the method of assembly based placement would be really advantageous over other methods, since it simulates the process of getting parts into their desired locations. The quantification of waste area could be extended to volumes, by using volume elements that descretize the space around the parts.

References

Adamowicz, M., Albano, A. (1976). "Nesting Two-dimensional Shapes in Rectangular Modules", Computer-Aided Design, Vol. 8, No. 1, pp. 27-33.

Albano, A., Sapuppo, G. (1990). "Optimal Allocation of Two-dimensional Irregular Shapes Using Heuristic Search Methods", IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-10, No. 5, pp. 242-248.

Baker, J. E. (1985). "Adaptive Selection Methods for Genetic Algorithms", Proceedings of the 1st International Conference on Genetic Algorithms and Applications, pp. 101-111, July 24-26.

Bethke, A. D. (1981). "Genetic Algorithms as Function Optimizers", Doctoral Dissertation, University of Michigan, Dissertation Abstracts International 41(9), 3503B. University Microfilms No. 8106101.

Cai, Y., Lujun, L., Wei, W., Jianwen, S. (1987). "An Expert System for Automatic Allocation of 2D Irregular Shapes", Proceedings of Expert Systems in Computer-Aided Design Conference, IFIP, pp. 407-423.

Cheok, B. N., Nee, A. Y. C. (1991). "Algorithms for Nesting of Ship/Offshore Structural Plates", Advances in Design Automation, Vol. 2, DE-Vol. 32-2, pp. 221-226, ASME.

Daniels, K., Milenkovic, V. J. (1984). "Multiple Translational Containment, Part I: An Approximate Algorithm", Submitted to the Algorithmica special issue on Computational Geometry in Manufacturing, June.

Fourman, M. (1985). "Compaction of Symbolic Layout using Genetic Algorithms", Proceedings of the 1st International Conference on Genetic Algorithms and Applications, pp.

141-153, July 24-26.

Fujita, K., Akagi, S., Hirokawa, N. (1993). "Hybrid Approach for Optimal Nesting Using a Genetic Algorithm and a Local Minimization Algorithm", Proceedings of the ASME Conference on Advances in Design Automation, DE-vol. 65-1, pp. 477-484, Albuquerque, New Mexico.

Garey M., Johnson D. (1979). "Computers and Intractability: A Guide to the Theory of NP-Completeness", W. H. Freeman and company, New York.

Goldberg, D. (1989). "Genetic Algorithms in Search, Optimization, and Machine Learning", Addison-Wesley, Reading, Massachusetts.

Holland, J. (1975). "Adaptation in Natural and Artificial systems", University of Michigan Press, Ann Arbor, Michigan.

Jain, P., Fenyes, P., Ritcher, R. (1990). "Optimal Blank Nesting Using Simulated Annealing", ASME Conference on Advances in Design Automation, DE-Vol. 23-2, pp. 109-116.

Kroger, B., Schwenderling, P., Vornberger, O. (1990). "Parallel Genetic Packing of Rectangles", Proceedings of Parallel Problem solving from Nature, 1st Workshop, Springer-Verlag, pp. 160-164.

Li, Zhenyu. (1994). "Compaction Algorithms for Nonconvex Polygons and Their Applications", Ph.D. Thesis, Technical Report TR-15-94, Center for Research in Computing Technology, Division of Applied Sciences, Harvard University, May.

Milenkovic, V. J., Daniels, K., Li, Z. (1994). "Multiple Containment Methods", Technical Report TR-12-94, Center for Research in Computing Technology, Division of Applied Sciences, Harvard University, May.

Oliviera, J. F., Ferreira, J. S. (1993). "Algorithms for Nesting Problems, Applied Simulated Annealing", Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, pp. 256-273.

Qu, W., Sanders, J. (1987). "A Nesting Algorithms for Irregular Parts and Factors Affecting Trim Losses", International Journal of Production Research, Vol. 25, No. 3, pp. 381-397.

Qu, W., Sanders, J. (1989). "Sequence Selection of Stock Sheets in Two-dimensional Layout Problems", International Journal of Production Research, Vol. 27, No. 9, pp. 1553-1571.

Reeves, C. (1994). "Hybrid Genetic Algorithms for Bin-Packing and related problems", submitted to a special issue of Annals of Operations Research.

Sarin, S. C. (1983). "Two-dimensional Stock Cutting Problems and Solution Methodologies", Proceedings of the ASME Design and Production Engineering Technical Conference, Dearborn, Michigan, 83-Prod-5.

Smith, D. (1985). "Bin Packing with Adaptive Search", Proceedings of the 1st International Conference on Genetic Algorithms and Applications, pp. 202-207, July 24-26.