A GENETIC ALGORITHM FOR TWO DIMENSIONAL STRIP PACKING PROBLEMS

V. Mancapa[1], **T.I. van Niekerk**[2] and **T Hua**[2]

[1]Department of Electrical Engineering
Nelson Mandela Metropolitan University, South Africa
mancapa@nmmu.ac.za

[2]Department of Mechatronics
Nelson Mandela Metropolitan University, South Africa
theo.vanNiekerk@nmmu.ac.za

ABSTRACT

Cutting and packing problems are combinatorial optimisation problems. In most manufacturing situations a raw material, usually in some standard size, has to be divided or cut into smaller items to complete the production of some product. It is therefore desirable that this raw material be used efficiently. A novel placement heuristic, hybridised with a genetic algorithm, is presented in this paper. A general solution encoding scheme, which is used to encode two dimensional strip packing problems, is also introduced in this study.

OPSOMMING

Die optimisering van sny- en pakprobleme vorm deel van die kombinasieleer. Dit is dikwels so by vervaardiging dat grondstof onderverdeel (gesny) word om te pas by die samestelling van ‘n gegewe produk. Sodanige onderverdeling moet doeltreffend verrig word. ‘n Veredelde heuristiese genetiese algoritme word hiervoor bekend gestel. ‘n Algemene koderingsmetode vir tweedimensionele strookverpakking word voorgehou.

[1] The author was enrolled for an MTech (Electrical Engineering) degree at the Department of Electrical Engineering, Nelson Mandela Metropolitan University.

1. INTRODUCTION

Cutting and packing (C&P) problems are combinatorial optimisation problems of practical significance. In most manufacturing situations it is required that the raw material be cut into smaller pieces. This process usually results in waste. It is therefore desirable to reduce the waste that results as much as possible, and hence maximise material utilisation. Examples of this can be seen in the glass, paper, steel, semiconductor, textile, and many other industries. The two-dimensional (2D) packing problems can be classified as bin packing problems and strip packing problems (SPPs). The bin packing problem is concerned with minimising the number of bins into which small items need to be packed. Strip packing involves packing rectangular or irregular items on to a strip of constant width and unlimited height, and the objective is to minimise strip height without overlapping the items.

Solution approaches to SPP can be divided into two categories: exact methods, and heuristics. SPP is a *non-deterministic polynomial complete* (NP-complete) problem[1]. The objective function of an NP-complete problem is often multi-modal, non-smooth, or even discontinuous, and thus causes exact gradient-based optimisation algorithms to fail[2]. Few exact methods for the 2D-SPP have been used so far, with applicability limited to piece sets within 200 pieces. Martello et al. proposed a branch-and-bound algorithm for the 2D-SPP[3], while Fekete and Schepers developed a general framework for exact approaches to more-dimensional packing problems[4], and Hifi presented an improvement of Viswanathan and Bagchi's exact algorithm based upon branch-and-bound procedures for strip cutting/packing problems[5].

In recent decades, heuristics have received considerable attention, and have been deemed most suitable for solving the 2D-SPP. Stochastic search techniques, such as genetic algorithm (GA), simulated annealing (SA), tabu search (TS), naïve evolution (NE), population heuristic (PH), and other meta-heuristics, have been combined with a placement algorithm (decoding algorithm) for solving 2D-SPPs[2]. Hopper and Turton provided an extensive up-to-date overview of the meta-heuristics that have been developed for the different variants of the 2D-SPPs. They hybridised two different placement algorithms (BL, BLF) with three meta-heuristic algorithms (GA, SA, and NE) and local search heuristic (hill-climbing) for the two-dimensional rectangular packing problem[1; 6]. Leung et al. applied a pure GA and a mixed SA–GA, which used SA to decide which two parents and children should be selected after crossover and mutation, for 2D orthogonal packing problems. The mixed heuristic SA–GA was found to produce better results in a test of 19 cutting problems[7].

Genetic algorithms utilise search and optimisation procedures that operate in a similar way to the evolutionary processes observed in nature. The GA search is guided, using the 'survival of the fittest' principle, by extracting the most desirable features from a generation of solutions, and combining them to form the next generation. The quality of each solution is evaluated, and the 'fitter' individuals are selected for the reproduction process. Continuation of this process through a number of generations will result in optimal or near-optimal solutions[8].

As a relatively new technique for the packing industry, GAs have already been used successfully in a variety of industrial applications. Jakobs developed an order-based GA with a bottom left (BL) algorithm to place rectangles and polygons on to a rectangular main object. The GA in the study was combined with embedding a shrinking algorithm in place of polygons[9]. Liu and Teng improved Jakobs' work by developing a more effective version of the BL algorithm for the orthogonal packing of rectangular pieces[10]. Hopper and Turton described two GAs hybridised with two heuristic placement algorithms – the bottom left (BL) algorithm and the bottom left fill (BLF) algorithm – for rectangular packing problems[11]. Yeung and Tang proposed a GA combined with a novel heuristic allocation method to transform the 2D-SPP into a simple permutation problem that could be effectively solved by a GA and a greatly reduced searching domain[12]. Bortfeldt suggested a GA working without any encoding of solutions by using specific genetic operators to manipulate the fully

defined layouts. The GA showed good performance on a comprehensive test using benchmark instances with up to 5,000 pieces[13].

Despite decades of academic research into regular packing problems, the work on two-dimensional irregular problems is recent. A major reason for this is the extra dimension of complexity generated by the geometry. However, irregular problems occur within several important industries; examples include die-cutting in the engineering sector, parts nesting for shipbuilding, and marker layout in the garment industry. Gomes and Oliveira presented a hybrid algorithm that uses simulated annealing (SA) to guide the search over the solution space, and linear programming models to generate neighbourhoods during the search process, to solve irregular strip packing problems[14].

In this study, a novel placement heuristic hybridised with GA, and a general solution encoding scheme used to encode two dimensional strip packing problems, were introduced. In the remainder of this paper, Section 2 provides a description of the proposed GA used for the 2D-SPP; Section 3 subjects the GA for 2D-SPP to benchmark tests; and Section 4 gives the study's conclusions.

2. THE GENETIC ALGORITHMS FOR STRIP PACKING PROBLEMS

The two strip packing problems addressed in this paper are described below:

- Given n items of small rectangles each having width w_i and height h_i, and one large rectangular strip with constant width W and infinite height, the objective is to minimise the packing height H of the strip such that all items can be packed into the strip without overlap. The small rectangular items can be rotated by $90°$. This problem is known as the two dimensional strip packing problem (2D-SPP).
- Given n items of arbitrary shapes and one strip with constant width W and infinite height, the objective is to minimise the packing height H of the strip such that all items are contained in the strip without overlap. The irregular items can be rotated at fixed $90°$ increments. This problem is known as the two dimensional irregular strip packing problem (2DISP).

GA is the mathematical procedure based on analogies to the natural evolutionary process. It is different from random algorithms, and combines elements of directed and stochastic search. Figure 1 illustrates the pseudo code of a simple GA.

Algorithm 1 Simple GA

```
begin
        t←0
        initialise P(t)
        evaluate P(t)
            While (!(halting condition)) do
                begin
                    t←t+1
                    select P(t) from P(t-1)
                    alter P(t)
                    evaluate P(t)
                end
            end
```

Figure 1: Pseudo code of a simple GA

The GA maintains a population of individuals $P(t)$ that are created and selected in an iterative process. Each individual consists of a genome, fitness, and possibly some auxiliary variables such as age and sex. The genome consists of a number of *genes* that altogether *encode* a solution to some optimisation problem. The *encoding* is the internal representation of the problem that is the data structure holding the genes. Every member of the population is evaluated to measure their fitness. A new population at iteration $t+1$ is formed by selecting those individuals who have more fitness. Some members of the population undergo transformations ('alter' step in the pseudo code). This is achieved by means of some variation operators such as *crossover operator* and *mutation operator*. The algorithm executes until some predefined halting condition - for example, the solution quality, number of generations, or simply running out of time - is reached[15].

The following subsections provide a detailed implementation of the proposed GA for 2D rectangular and irregular strip packing problems.

2.1 Solution representation

A representation (or encoding) for a problem is needed before the GA can be run. In this study, a potential solution to the 2D-SPP is represented as a set of 3-tuples $\{(x_1,i_1,\phi_1),(x_2,i_2,\phi_2),...,(x_n,i_n,\phi_n)\}$. These 3-tuples (known as genes) are joined together to form a string of values, referred to as an individual. Each 3-tuple depicts the position and orientation state from which the corresponding item starts being placed. Figure 2 illustrates each element in the 3-tuple of an example solution string $\{(x_3,3,90°),(x_1,1,0°),(x_2,2,90°)\}$ for the packing of three items.

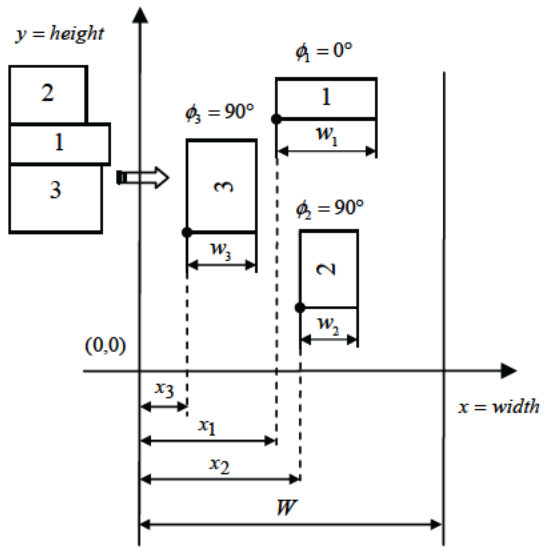

Figure 2: An example of solution representation

- The first element in the 3-tuple is the *x*-coordinate value x_k, the distance from the reference vertex of the k th item to the origin of the strip in *x* direction. The range of x_k of the *k* th item is $[0, W - w_k]$, where *W* and w_k is the width of the strip and the *k*th item, respectively. The value x_k determines the starting location of the k th item when placed. The reference vertex of an item is its bottom→left most point - for example, the bottom left corner for a rectangle item. Figure 1 illustrates the *x*-coordinate values for the three items.

- The identification of the k th item in a solution is given by its item index i_k. The item indices are randomly generated in the initial population, and will stay unchanged during packing. The appearance sequence of the item indices in the solution represents the placing order of the items. In the example solution $\{(x_3,3,90°),(x_1,1,0°),(x_2,2,90°)\}$ illustrated in Figure 1, the three items with an index of '1', '2', and '3' are to be placed in the order '3'→'1'→'2' – that is, item 3 is to be placed first, and item 2 last.

- The orientation angle ϕ_k of the k th item reflects the angle that the item rotated from its initial orientation. In this study, the orientation angle is defined as $\phi_k \in \{0°,90°\}$ for a rectangle item and $\phi_k \in \{0°,90°,180°,270°\}$ for an irregular item. Figure 1 shows the orientation angles for the three items: $\phi_1 = 0°$ for item 1, and $\phi_2 = \phi_3 = 90°$ for item 2 and item 3.

2.2 Initial population generation

To generate the initial population, it is ensured that every individual belonging to the initial population of solutions is feasible. Every item is represented by a 3-tuple (x_k, i_k, ϕ_k), to generate initial population, and the following procedure is followed:

- Randomly order items.
- Randomly choose a feasible x-coordinate of the reference vertex for each item from the set of feasible x-coordinates.
- Randomly choose an orientation from the set of feasible orientation constraints for each item.

2.3 Crossover operator

Crossover operator is simply a matter of replacing some genes in one parent by the corresponding genes of the other. The crossover operation can involve more than two parents. In this study, the crossover operator is randomly chosen from two crossover variants: cross-var1 and cross-var2. A variable $CrossVar \in \{0,1\}$ is randomly generated to decide which of these two variants will be operational. If $CrossVar = 0$, cross-var1 will be operational, otherwise cross-var2 will be used. A partially mapped crossover (PMX) of Michalewicz and Fogel[16] is slightly modified and applied for that purpose.

Cross−var1

Of the three components in a solution, the ***cross−var*1** allows the orientation angle and the ***x***-coordinate value of the reference vertex to be directly inherited from one parent solution; while the ordering of the items is achieved through breeding between both parent solutions. For example, let X_{p1} and X_{p2} be the two-parent solution representing a layout of 6 items:

$$X_{p1} = \{(0,6,0^0),(9,3,90^0),(5,2,90^0),(2,5,0^0),(10,4,90^0),(3,1,90^0)\} \quad (1)$$

$$X_{p2} = \{(1,3,90^0),(5,1,90^0),(4,2,0^0),(6,5,90^0),(2,4,0^0),(8,6,0^0)\} \quad (2)$$

Let O_1 be the offspring that results from the breeding of the two parent solutions. The ***cross−var*1** works as follows:

(1) Copy the x-co-ordinate of the reference vertex and orientation angle of every item from solution X_{p2} to the offspring O_1. In this example, at this stage, the offspring becomes:

$$O_1 = \{(1,U,90^0),(5,U,90^0),(4,U,0^0),(6,U,90^0),(2,U,0^0),(8,U,0^0)\} \quad (3)$$

(The symbol '*U*' can be interpreted as 'at present unknown')

(2) Generate two random positions $p1$ and $p2$, such that $1 \le p1 \le p2 \le 6$. For example, say $p1$ is generated to be 1, and $p2$ to be 3.

(3) Create a one-to-one mapping between item indices in positions decided by $p1$ and $p2$ from both parents. For this example, the mapping is created between the item indices of X_{p2} and X_{p1} from the 1st (*p*1=1) to the 3rd (*p*2=3) item index. The series of mappings for this example is:

$$\begin{array}{lccc} & X_{p2} & & X_{p1} \\ index1 & 3 & \leftrightarrow & 6 \\ index2 & 1 & \leftrightarrow & 3 \\ index3 & 2 & \leftrightarrow & 2 \end{array} \quad (4)$$

(4) Copy every item index between positions $p1$ and $p2$ from X_{p2} to O_1 to corresponding positions. After copying the offspring, O_1 becomes:

$$O_1 = \{(1,3,90^0),(5,1,90^0),(4,2,0^0),(6,U,90^0),(2,U,0^0),(8,U,0^0)\} \quad (5)$$

(5) Item indices in the positions excluding $p1 - p2$ are copied from corresponding positions from X_{p1}. To avoid the conflict that might occur when the item index copied from X_{p1} already exists in O_1 in $p1 - p2$ positions, the mapping created in stage b is used to generate a new item index, replacing the conflicting item index, until no conflict occurs. For this example, the 4th item index '5' and the 5th item index '4' of X_{p1} are not in conflict with the ones of O_1 in $p1 - p2$ positions; thus they are copied as the 4th and 5th item index for O_1. The 6th item index in X_{p1} is '1' and it conflicts with the 2nd item index of O_1; thus the mapping 1↔3 is used to generate new item index '3'. However, the generated item index '3' conflicts with the 1st item index of O_1; thus the mapping 3↔6 is used to generate new item index '6', and '6' is not in conflict with the item indices of O_1 in $p1 - p2$ positions. Therefore '6' is used as the 6th item index for O_1, as illustrated in Figure 3.

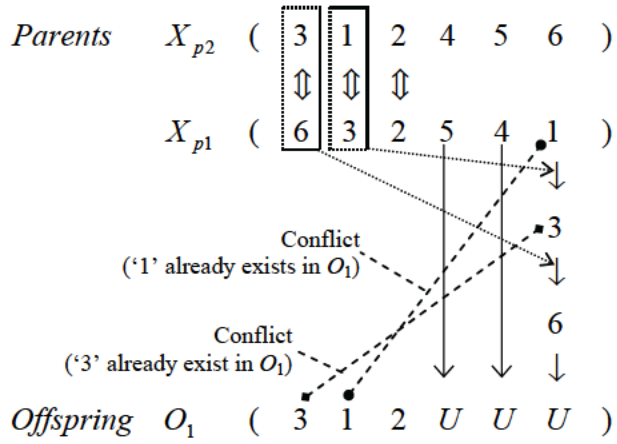

Figure 3: The process of generating item indices for offspring from both parents

The final solution of O_1 becomes:

$$O_1 = \{(1,3,90^0),(5,1,90^0),(4,2,0^0),(6,5,90^0),(2,4,0^0),(8,6,0^0)\} \quad (6)$$

There is a possibility that infeasible solutions might be introduced into the population with this variant of crossover. To counteract this possibility, a penalty function is used to degrade the quality of infeasible solutions, as can be seen in section 3.5.

Cross–var2

The major difference between ***cross–var1*** and ***cross–var2*** is that ***cross–var1*** allows a situation where breeding involves both item characteristics in the solution representation and the ordering of the items for the packing. The ***cross–var2***, on the other hand, is mainly concerned with the ordering of the items without separating the item characteristics and the ordering - that is, when items change positions in the ordering, the item moves with the characteristics that define it. In other words the whole 3-tuple (x_k, i_k, ϕ_k) moves. The parents used to demonstrate ***cross–var1*** will again be used to demonstrate ***cross–var2***.

The ***cross–var2*** breeds offspring as follows:

(1) Generate two random positions $p1$ and $p2$, such that $1 \le p1 \le p2 \le n$: For example, say the random process results in $p1 = 2$ and $p2 = 4$.

(2) Create a one-to-one mapping of item indices from both solutions in positions $p1 - p2$. For this example the mapping between the item indices of X_{p1} and X_{p2} is:

$$\begin{array}{lccc} & X_{p1} & & X_{p2} \\ index2 & 3 & \leftrightarrow & 1 \\ index3 & 2 & \leftrightarrow & 2 \\ index4 & 5 & \leftrightarrow & 5 \end{array} \quad (7)$$

(3) Copy from parent X_{p1} items in position $p1 - p2$ with their related characteristics. This results in a partial offspring solution, which is:

$$O_2 = \{(U,U,U),(9,3,90^0)(5,2,90^0),(2,5,0^0),(U,U,U),(U,U,U)\} \quad (8)$$

(The symbol '*U*' can be interpreted as 'at present unknown').

(4) The solution is completed by copying items from parent X_{p2}, starting from left to right, excluding those items in $p1-p2$ positions. To avoid the conflict between the item indices copied from X_{p2} and those that already exist in O_2 at p_1-p_2 positions, the mapping list generated in stage (2) is used to obtain new 3-tuples to replace the conflicting ones. For this example, the 1st, 5th, and 6th 3-tuples are copied from X_{p2} to O_2 at corresponding positions. The item *index 4* in the 5th 3-tuple $(2,4,0^0)$ and the item *index 6* in the 6th 3-tuple $(8,6,0^0)$ from X_{p2} are not conflicting with those already existing in O_2. However, the item index '3' in the 1st 3-tuple $(1,3,90^0)$ from X_{p2} conflicts with the item index '3' in the 2nd 3-tuple $(9,3,90^0)$ of O_2; thus the mapping 3↔1 is used to take the whole 6th 3-tuple $(3,1,90^0)$ of X_{p1} to replace the conflicting one from X_{p2}. The resulting offspring O_2 for this example finally becomes:

$$O_2 = \{(3,1,90^0),(9,3,90^0),(5,2,90^0),(2,5,0^0),(2,4,0^0),(8,6,0^0)\} \quad (9)$$

2.4 Mutation operator

Mutation is a one-parent variation operator. The mutation operator is an over-simplified analogue from natural evolution. It usually consists of making small random perturbations to one or a few genes. One of the major reasons for the mutation operator in GAs is the introduction of population diversity during the genetic search. Originally, with binary encoding, a zero would be changed into a one and vice versa. With alphabets of higher cardinality, there are more optional changes that can be made at random or following a set of rules.

The 2-swap mutation operator that is usually used in sequencing problems has been adapted and modified as the mutation operator for 2D problems. The operator works as follows:

(1) Randomly choose two items, item1 and item2;

(2) Randomly generate a number $num \in \{0,1\}$ to decide if the orientation of the chosen items will be randomly perturbated;

(3) If $num = 1$ changes the orientation of both items randomly (this applies if more than one orientation is allowed);

(4) Exchange the position of item1 with that of item2.

2.5 Evaluation function

The evaluation function is the mechanism used to judge the quality of the evolved solutions. For the evaluation of the 2D SPP, a placement heuristic is used that considers one item at a time. Items are placed on the strip in the order in which their item indices appear in the solution string. For each item k, the placement heuristic carries out the following steps in turn:

(1) Item k with index i_k is placed at the topmost position at horizontal position x_k, with the orientation of item k being that reflected by the orientation angle ϕ_k;

(2) Item k is then slid as far down as possible, until it collides with either the bottom edge of the strip or another item;

(3) Thereafter item k is slid as far left as possible until it collides with another item or the left edge of the strip. This becomes the final position of item k. In this study, because of the arbitrariness of the geometry of the pieces, the shift left stage is not part of the placement heuristic for irregular items - that is, the placement of irregular items is only carried out in step (1) and step (2).

Figure 4 illustrates the placement-heuristic procedure for the packing of three items, using the example from Figure 2 with the solution $\{(x_3,3,90^0),(x_1,1,0^0),(x_2,2,90^0)\}$.

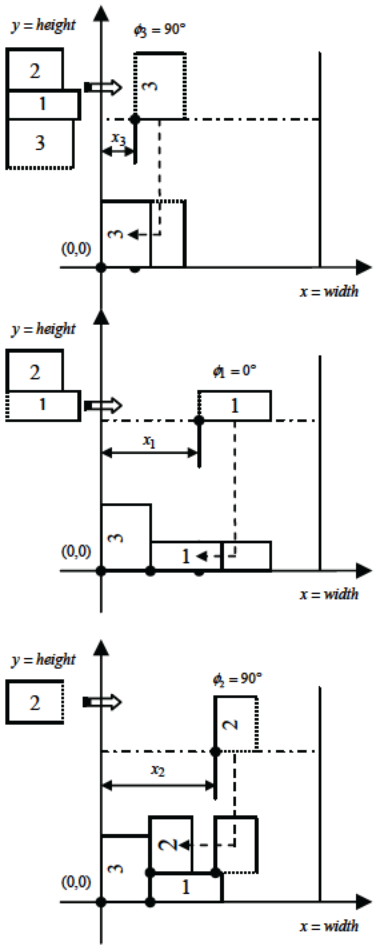

Figure 4: Placement-heuristic example for strip packing problem

The search space S consists of two subsets: the feasible part $\mathcal{F} \subseteq S$ and the infeasible part $\mathcal{U} \subseteq S$. In the discussion on cross-over operator in sub-subsection 2.3 above, it is

mentioned that ***cross_var1*** might introduce infeasible solutions into the population. There are two possible violations of constraints that can occur in 2D SPP, namely overlap constraint and the containment constraint. ***Cross_var1*** is guilty of violating the latter constraint - that is, placement of items outside the borders of the strip.

In this study, the evaluation function for the 2D rectangle strip packing problem is given by:

$$F_1(X) = \begin{cases} P_1(X) & X \in \mathcal{U} \\ Eff_1(X) & X \in \mathcal{F} \end{cases} \tag{10}$$

The penalty function $P_1(X)$ is used as a constraint handling mechanism. Any solution in violation of the above-mentioned constraint is 'killed' - that is, the solution is made undesirable. The function $Eff_1(X)$ measures the efficiency of the packing. The total area of items to be packed is given by:

$$A = \sum_{k=1}^{n} w_k h_k \tag{11}$$

where w_k and h_k are the width and height of the kth item respectively. Ideally the total area of the strip occupied by the items is supposed to be A, but in most instances this is not the case, and A is a continuous lower bound for every instance of this problem. Let A_p be the area that results after all items have been placed on the strip. A_p is given by:

$$A_p = h_p W \tag{12}$$

Where h_p is the packing height and W the width of the strip, as illustrated in Figure 5, with the packing result from Figure 4.

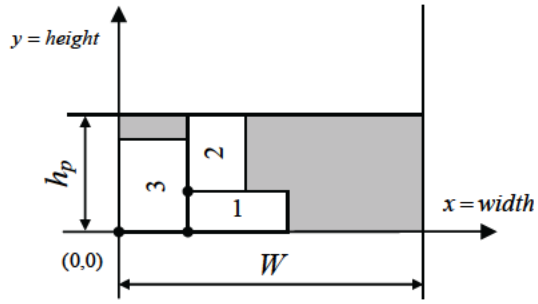

Figure 5: Example of packing height

The efficiency function $Eff_1(X)$ is given by:

$$Eff_1(X) = \frac{A}{A_p} = \frac{\sum_{k=1}^{n} w_k h_k}{W h_p} \tag{13}$$

The function $Eff_1(X)$ reflects the efficient use of the strip - that is, those individuals in the population who use the strip efficiently are rewarded the most. The lowest packing height possible is given by:

$$h_L = \frac{A}{W} \tag{14}$$

It is desirable that an individual packing height be as close as possible to this height. To make infeasible solutions undesirable, they are moved as far as possible from this bound by a factor *K*, such that a penalty packing height $h_{penalty} = Kh_L >> h_L$ is chosen. The penalty function $P_1(X)$ is:

$$P_1(X) = \frac{A}{Wh_{penalty}} \tag{15}$$

The evaluation function $F_2\ (X)$ for the 2D irregular strip packing problem is given:

$$F_2\ (X) = \begin{cases} P_2(X) & X \in \mathcal{U} \\ Eff_2(X) & X \in \mathcal{F} \end{cases} \tag{16}$$

Let A_k be the area of the *k*th irregular piece; the total area A of the pieces is then $A = \sum_{k=1}^{n} A_k$. Let h_p be the packing height; the packing area A_p is given by (4). Function $Eff_2(X)$ is given by:

$$Eff_2(X) = \frac{A}{A_p} = \frac{\sum_{k=1}^{n} A_k}{Wh_p} \tag{17}$$

The penalty function $P_2(X)$ for an irregular packing problem is computed similarly to that for a rectangular packing problem as:

$$P_2(X) = \frac{A}{Wh_{penalty}} = \frac{\sum_{k=1}^{n} A_k}{Wh_{penalty}} \tag{18}$$

3. NUMERICAL TESTS

In order to evaluate the performance of the GA presented in this work, problematic instances have been collected from the literature. All experiments were conducted using a 3.4 GHz Pentium 4 processor. The algorithm was coded in Matlab and run using Matlab's *Genetic Algorithm and Direct Search Toolbox*[17]. For every problem the population size was set at 100 individuals, although this tended to slow down the speed of the algorithm. After repeated runs for most problems, it was decided that the crossover fraction should be kept at 0.3 for all problems. Two individual spots in the population were reserved for elite children – that is, the two best individuals in every generation. A tournament of size 2 was used as a selection criterion. The GA was run for 2,000 generations for every problem;

however, if the best fitness did not improve after 1,000 generations or 2,000 seconds, the algorithm stopped.

Results for the 2D rectangular strip packing problems

The strip packing problems for rectangles that cannot be rotated were first tested with the GA developed in this study. The test problems used in this work were considered in Martello et al.[3]. Altogether, 38 test problems were collected from various sources. The number of items to be packed ranged from 10 to 200. (These test problems can also be downloaded from the ESICUP (Euro special interest group on cutting and packing) website home page[18].) Using the GA proposed in this paper, the test results for these 38 problems are shown in Table 1. For each problem Table 1 gives:

No.	n	LB	z	Time	PR	Eff
1	16	20	23	3581.	1.15	87%
2	17	20	23	2212.	1.15	87%
3	16	20	23	2026.	1.15	87%
4	25	15	18	2121.	1.2	83%
5	25	15	18	2042.	1.2	83%
6	25	15	17	2967.	1.13	88%
7	28	30	36	2150.	1.2	83%
8	29	30	37	3166.	1.23	81%
9	28	30	39	2333.	1.3	77%
10	16	23	25	2204.	1.08	92%
11	23	63	72	2653.	1.14	88%
12	62	636	730	8011.	1.15	87%
13	10	1016	1016	1105.	1	100%
14	20	1133	1215	4028.	1.07	93%
15	30	1803	1866	3818.	1.03	97%
16	50	2934	3340	5658.	1.138	88%
17	10	23	23	1165.	1	100%
18	17	30	30	2082	1	100%
19	21	28	31	2494.	1.11	90%
20	7	20	20	869.8	1	100%
21	14	36	35.3	2245.	1.02	98%
22	15	31	35	1699.	1.13	89%
23	8	20	20	971.2	1	100%
24	13	33	34	3009.	1.03	97%
25	18	49	56	3192.	1.14	88%
26	13	80	80	1754.	1	100%
27	15	52	61	1959.	1.17	85%
28	22	87	87	2246.	1	100%
29	20	30	34	2383.	1.13	88%
30	40	57	65	2623.	1.14	88%
31	60	84	100	4363	1.19	84%
32	80	107	130	4703.	1.21	82%
33	100	134	167	4086.	1.25	80%
34	40	36	44	3121.	1.22	82%
35	80	67	85	3001.	1.27	79%
36	120	101	133	4858.	1.32	76%
37	160	126	160	6903.	1.27	79%
38	200	156	209	4352.	1.34	75%

Table 1: Strip packing problem results where items cannot be rotated

- *no* the problem number;
- *n* the number of rectangles in each problem;
- *LB* the lower bound – that is, the ideal lowest packing height h_L as defined in (14);
- *z* the best solution found by the GA – that is, the lowest real packing height;
- *Time* the total search time in seconds taken by the GA to find the best solution;
- *PR* the performance ratio – that is, the ratio of *z*/*LB*.
- *Eff* the packing efficiency of the best solution given by $Eff = A / A_p = (W \times LB) / (W \times z) = LB / z$, where *W* is the width of the strip.

Table 1 shows that generally the packing efficiency of the GA solution decreases with the number of packed items. The GA, hybridised with the novel placement heuristic, shows good performance for solving the 2D rectangular strip packing problems where items cannot be rotated. Some of the problems are solved with a high packing efficiency of 100%. Figure 6 shows an example layout, in which the gray areas are waste, generated by this hybrid algorithm for problem 21. The item number and packing efficiency of this problem are 14 and 98% respectively. Table 2 shows the width and height of each item shown in Figure 6.

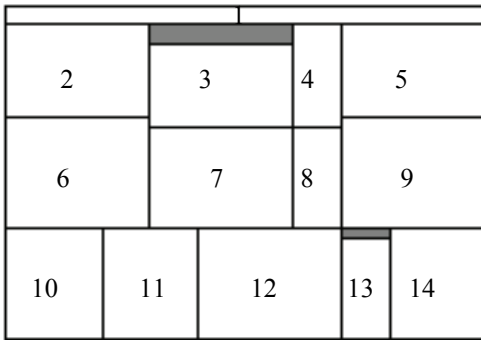

Figure 6: Layout example of rectangular items that cannot be rotated

Item	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Width	50	15	15	5	15	15	15	5	15	10	10	15	5	10
Height	2	10	9	11	10	12	11	11	12	12	12	12	11	12

Table 2: The position of items in GA solution for problem no. 21

The strip packing problems for rectangles that can be rotated by 90° were also tested using the hybrid GA developed in this study. The test data, which consists of 21 problems presented in seven different-sized categories, are taken from Hopper and Turton[6]. These test problems are very difficult to solve as they are 'perfect packings' obtained by cutting a given rectangle of fixed dimensions into smaller rectangular items. Table 3 gives the test results obtained by the GA developed in this study.

Category	no.	n	LB	z	Time	PR	Eff
C1	P1	16	20	22	2587.2	1.1	90%
	P2	17	20	23	2112.5	1.15	87%
	P3	16	20	23	2346.3	1.15	87%
C2	P1	25	15	19	2207.4	1.27	79%
	P2	25	15	19	2211.7	1.27	79%
	P3	25	15	19	2525.4	1.27	79%
C3	P1	28	30	36	3045.9	1.2	83%
	P2	29	30	34	3173.8	1.13	88%
	P3	28	30	36	2496.2	1.2	83%
C4	P1	49	60	70	10122	1.17	86%
	P2	49	60	72	3136.3	1.2	83%
	P3	49	60	75	2661.9	1.25	80%
C5	P1	73	90	117	2567.4	1.3	77%
	P2	73	90	124	3764.8	1.38	73%
	P3	73	90	109	8170.9	1.21	83%
C6	P1	97	120	159	3796.5	1.33	75%
	P2	97	120	160	3422.1	1.33	75%
	P3	97	120	160	3387.5	1.33	75%
C7	P1	196	240	330	6249.2	1.38	73%
	P2	197	240	346	10911	1.44	69%
	P3	196	240	352	5294.4	1.47	68%

Table 3: Strip packing problem results where items can be rotated by 90°

The test results show that the hybrid GA is also suitable for solving 2D rectangular strip packing problems where items can be rotated by 90°. The packing efficiency of the GA solution generally decreases with the number of packed items. Figure 7 shows an example layout, in which the gray areas are waste, generated by this hybrid algorithm for problem P1 of category C1. The item number and packing efficiency of this problem are 16 and 91% respectively. Table 4 gives the width, height, and rotation angle of each item shown in Figure 7.

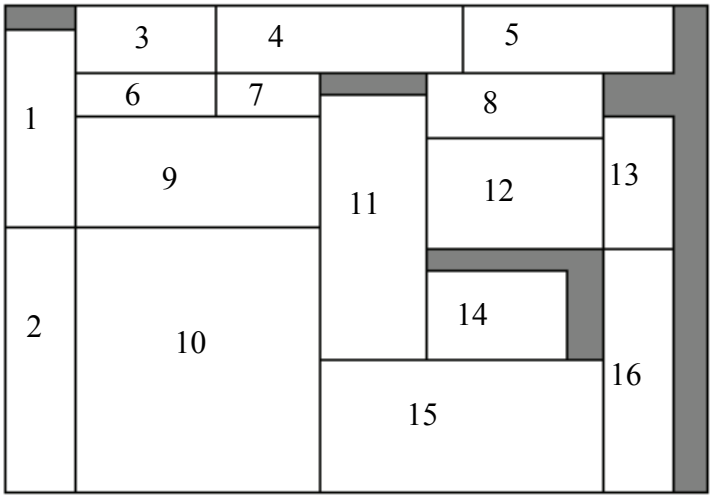

Figure 7: Layout example of rectangular items that can be rotated by 90°

Item	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Width	3	3	6	11	9	6	5	8	11	11	5	8	3	5	11	3
Height	9	12	3	3	3	2	2	3	5	12	12	5	6	4	6	11
Angle	90	90	0	0	0	0	0	0	0	90	90	0	90	0	0	90

Table 4: The position and orientation of items in GA solution for P1 in Category 1

Results for 2D irregular strip packing problem

To test the hybrid GA on 2D irregular strip packing, problems of this type (also featured on the ESICUP website) were used[18]. The four test problems, all derived from the textile industry, have been brought to the attention of the academic community. These problem details, including the literature source, the problem name, the number of shapes to be packed, the sheet width, and the orientation constraints, are listed in Table 5.

Problem Source	Problem Name	Shape number	Sheet width	Rotational Constraints
Oliveira et al[19]	Shirts	99	40	0,180 Absolute
Oliveira et al.[19]	Trousers	64	79	0,180 Absolute
Albano and Sapuppo[20]	Albano	24	4900	90 Incremental
Marques et al[21]	Marques	24	104	90 Incremental

Table 5: Details about irregular test problems

The test results on this type of problem are listed in Table 6. For the test problems on which this algorithm was used, the packing efficiencies have been above 60%. For optimum results an interesting study would be to compare current problems with the best results available to date.

Problem name	*Packing efficiency*	*Time (seconds)*
Shirts	61%	3409.8
Trousers	64%	4005
Albano	74%	2889
Marques	72%	3001

Table 6: Summary of results for irregular problems

Figure 8 shows an example layout in which the white areas are waste, generated by this hybrid algorithm for the Albano problem. The item number and packing efficiency of this problem are 24 and 74% respectively. Table 7 gives the rotation angle, with the best solution for each irregular item shown in Figure 8.

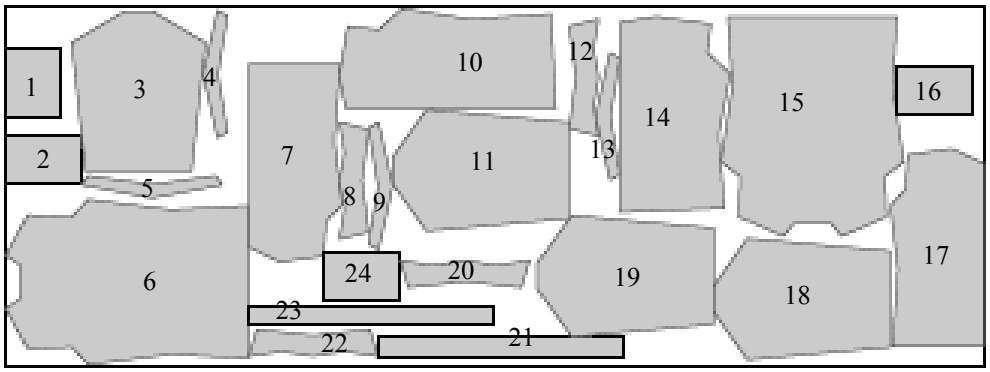

Figure 8: A textile marker layout generated by the GA for the Albano problem

Item	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Angle	90	0	90	90	180	180	270	270	270	180	180	90
Item	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
Angle	90	90	270	0	90	180	180	180	0	0	0	0

Table 7: The rotation angle of items in the GA solution for the Albano problem

4. CONCLUSION

A genetic algorithm was developed in this study to solve two-dimensional cutting and packing problems. A novel general solution coding, which makes use of a set of 3-tuples to represent the position, identification, and orientation of the items to be placed, and a novel heuristic placement procedure have been introduced in the design of this genetic algorithm. An initial population was generated such that every individual solution belonging to the initial population of solutions was ensured feasibility. A partially mapped crossover, using two crossover variants, was slightly modified and applied in the crossover operator. The mutation operator and the evaluation function were used to introduce population diversity during the genetic search and to judge the quality of the evolved solutions.
Computational tests were carried out using the genetic algorithm developed in MATLAB language for a variety of strip packing problems, including rectangles packing and irregular items packing. The test results have shown that the algorithm returned quality solutions for most of the problems. Further study on an alternative implementation of this algorithm by speeding up the computing time, continued testing of the algorithm on problems from both the literature and the real world, and a comparative study between layouts generated by a human expert and those by the general genetic algorithm are needed to improve the genetic algorithm.

5. REFERENCES

[1] **Hopper, E. and Turton, B.C.H.** 2001. A review of the application of meta-heuristic algorithms to 2D strip packing problems. *Artificial Intelligence Review* 16, pp. 257–300.

[2] **Soke, A. and Bingul, Z.** 2006. Hybrid genetic algorithm and simulated annealing for two-dimensional non-guillotine rectangular packing problems. *Engineering Applications of Artificial Intelligence*, 19, pp. 557–567.

[3] **Martello, S., Monaci, M. and Vigo, D.** 2003. An exact approach to the strip-packing problem. *Informs Journal on Computing* 15, pp. 310–319.

[4] **Fekete, S.P. and Schepers, J.** 1997. On more-dimensional packing III: Exact algorithms. *Technical Report ZPR97-290*, Mathematisches Institut, Universität zu Köln.

[5] **Hifi, M.** 1998. Exact algorithms for the guillotine strip cutting problem. *Computers Ops Res*. Vol. 25, No. 11, pp. 925-940.

[6] **Hopper, E. and Turton, B.C.H.** 2001. An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. *European Journal of Operational Research*, 128, pp. 34-57.

[7] **Leung, T.W., Chan, C.K. and Troutt, M.D.** 2003. Application of a mixed simulated annealing-genetic algorithm heuristic for the two-dimensional orthogonal packing problem. *European Journal of Operational Research* 145, pp. 530–542.

[8] **Davis L.** 1991. *Handbook of genetic algorithms*. New York: Van Nostrand Reinhold.

[9] **Jakobs, S.** 1996. On genetic algorithms for the packing of polygons. *European Journal of Operational Research* 88, pp. 165–181.

[10] **Liu, D. and Teng, H.** 1999. An improved BL-algorithm for genetic algorithm of the orthogonal packing of rectangles. *European Journal of Operational Research* 112, pp. 413–420.

[11] **Hopper, E. and Turton, B.** 1999. A genetic algorithm for a 2D industrial packing problem. *Computers & Industrial Engineering*, 37, pp. 375-378.

[12] **Yeung, L.H.W. and Tang, W.K.S.** 2004. Strip-packing using hybrid genetic approach. *Engineering Applications of Artificial Intelligence*, 17(2), pp. 169-177.

[13] **Bortfeldt, A.** 2006. A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces. *European Journal of Operational Research*, 172(3), p. 814-837.

[14] **Gomes, A.M. and Oliveira, J.F.** 2006. Solving irregular strip packing problems by hybridising simulated annealing and linear programming. *European Journal of Operational Research*, 171, pp. 811–829.

[15] **Mitchell, M.** 1998. *An introduction to genetic algorithms*. Cambridge, MA: MIT Press.

[16] **Michalewicz, Z. and Fogel, D. B.** 2000. *How to solve it: Modern heuristics*. Heidelberg: Springer-Verlag Berlin.

[17] MATLAB Version 7, 2006. *Genetic algorithm and direct search toolbox 2: User's guide*. Natick, MA: The Mathworks Inc.

[18] Euro Special Interest Group on Cutting and Packing, *Listing Gallery: Data Sets 2D - Rectangular*. Available from http://paginas.fe.up.pt/~esicup/tiki-list_file_gallery.php?galleryId=3 (Accessed 8 August 2007).

[19] **Oliveira, J., Gomes, A. and Ferreira, J.** 2000. A new constructive algorithm for nesting. *OR Spektrum*, 22, pp. 263–284.

[20] **Albano, A. and Sapuppo, G.** 1980. Optimal allocation of two-dimensional irregular shapes using heuristic search methods. *IEEE Trans. Syste., Man, Cybern., SMC* **IEEE**

Transactions on **Systems**, **Man** and Cybernetics **SMC-10**, 5, pp. 242-248.

[21] **Marques, V., Bispo, C. and Sentieiro, J.** 1991. A system of compaction of two-dimensional irregular shapes based on simulated annealing. In *IECON-91(IEEE)*.