Cutting optimisation with variable-sized stock and inventory status data

Leon Kos[†][*]and Jože Duhovnik [†]

Keywords: cutting optimisation, variable-sized bin packing problem, inventory status data, hybrid genetic algorithm

Abstract

Many production environments require economical cutting of one-dimensional items according to bills of materials from objects of several standard lengths. However, even with optimised cutting substantial trim loss may occur. This trim loss should not be regarded as waste. It is returned to store and can be reused in future optimisations. Optimisation of packing linear items into standard lengths is presented for items which cannot be packed into available lengths from inventory status data. The core of the proposed optimisation tackles the variable-sized bin packing problem (VBPP). The article presents a hybrid genetic algorithm which packs items into both available objects from the inventory and variable-sized objects from the stock. The algorithm tries to minimise waste. Large trim-loss items are returned as *remnants* to the inventory for subsequent optimisations.

[*]To whom correspondence should be addressed. e-mail: leon.kos@lecad.uni-lj.si

[†]University of Ljubljana, Faculty of Mechanical Engineering, Aškerčeva 6, SI-1000 Ljubljana, Slovenia

1 Introduction

The problem of cutting optimisation with variable-sized stock and inventory status data arises particularly in steel structure production where the packing of item lengths of equal cross-sections into larger beams for cutting occurs. The beams cut into smaller pieces are of standard lengths and their cost per unit weight/length is constant. Any combination of standard lengths is feasible in order to minimise waste, i.e. not reusable trim loss. The number of beams and their lengths are not relevant: it is only the size of trim-loss items that matters in optimisation. The number of beams and their respective lengths can vary in the problem solution.

This method of production minimises costs, transport logistics and enables production flexibility. In so doing, cutting can be planned at the designing stage, with production material not yet available. The preferred practice is then to order beams of standard lengths and cut them with minimal trim loss. In addition to trim-loss minimisation, it is desired that *remnants*, i.e. reusable trim loss, is stored into the inventory for reuse in subsequent optimisations. The acceptable sizes of remnants are user-specified, based on the production type and production planning.

After dimensions have been established, production possibilities and orders are examined. A detailed production plan specifies the cutting source for each item from either the order or the inventory. The inventory typically supplies non-standard length beams remaining from previous cuttings.The inventory contains workable remnants, cutting failure items, and over-stocked items from previous orders' reserves.

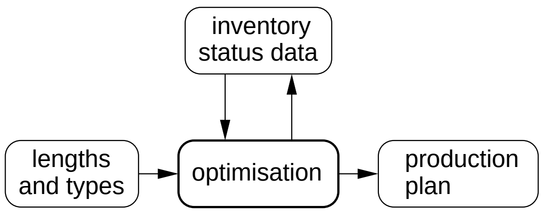

Figure 1: Information flow

The optimisation process is shown in Figure 1. The optimisation input consists of a list of beam sizes collected into subsets of different cross-sections. Optimisation is based on the inventory status data which provide available lengths with corresponding cross-sections. When the optimisation for each cross-section has been carried out, a production plan is drawn up and used items in the inventory status data are marked.

The assignment of the inventory items can have two results: (a) all required lengths can be packed into the items from the inventory status data, and therefore no order is needed, (b) only a limited number of the items can be used from the inventory status data, and additional beams are to be ordered to cover the unassigned lengths from the input. The latter is the preferred production alternative, as it both ensures the minimisation of the inventory and gains additional savings from ordering standard lengths.

Disregarding available items from the inventory status data, a simplified approach to cutting optimisation from a list of available lengths can be formulated as the *Variable-Sized Bin Packing Problem* (VBPP).

In the classical bin packing problem one is given a list of items L with a task to pack them into a minimum number of bins so that the total size of the items in the bins does not exceed the capacity of the bin c. The variable-sized bin packing problem differs from the classical one in that not all bins have the same capacity. There is an infinite number of bins of each capacity $c_1 > c_2 > \ldots > c_m$. An instance I consists of a list of items L to be packed along with the set of bin sizes available for packing. The objective is to minimise the sum of the capacities of the bins used in packing.

Since the VBPP is an $\mathcal{NP}$-hard problem (Friesen and Langston 1986), several approximation algorithms have been studied, mostly for the online case (Zhang 1997) with emphasis on the worst-case analysis for variations of well-known classical bin packing algorithms, such as First-Fit, Next-Fit, Best-Fit, with various opening and closing rules (Burkard and Zhang 1997). The off-line VBPP, which is the subject of our research, has been sparsely studied so far. Murgolo (1987) presents an approximation scheme with a linear programming formulation of the VBPP, which reduces the number of item sizes by rounding. The linear objective function is implicitly constrained and solved with the ellipsoid method by the separation oracle based on a knapsack approximation. Unfortunately, there is still no method solving implicitly given linear programmes that would be both practically and theoretically efficient (Gröetschel et al. 1993).

Initial ordering can also have impact on the packing process and is normally used at a preprocessing stage. Random ordering is used only when packing is online. Descending ordering is the strategy chosen for approximation algorithms, as it tries to pack larger elements first.

The introduction of the reuse of remnants from previous cuttings into the optimisation process requires one to solve the *Multiple Subset Sum Problem* (MSSP) (Caprara et al. 1999), which again is an $\mathcal{NP}-hard$ problem. The MSSP can be tackled by methods similar to the (V)BPP.

Other approaches to the classical bin packing problem include heuristic meta-strategies, e.g. tabu search (Scholl and Klein 1997), simulated annealing and genetic algorithms (Whitley 1993). Generally, these methods offer satisfactory results, although their performance is hard to prove even for the asymptotic performance ratio (Suzuki 1998). Experimental results are normally given for such methods and show relative performance in comparison with other algorithms or improvements on the initial solution. Such methods normally produce experimental results showing relative performance in comparison with other algorithms or improvements to the initial solution.

Meta-heuristics can provide better results for the off-line case. Genetic algorithms (GA) have been successfully applied to a wide variety of applications. Unlike other heuristic approaches which try to improve a single solution, the genetic algorithm works with a number of current solutions and tries to improve them by combining them together to generate new, hopefully better, solutions.

The main body of the article is structured as follows: Section 2 describes the core algorithm which solves variable-sized bin packing. In Section 3 we extend the strategy from Section 2 to handle inventory status data and prepare a bill of materials for cutting optimisation. Experiments with proposed algorithms and results are presented in Section 4. Real-world application based on the presented algorithms is described in Section 4.1.

2 Hybrid genetic algorithm for the VBPP

According to Dyckhoff's typology (1990) the VBPP can be classified as 1/V/D/M type. This type is described as a one-dimensional problem (1) where all items are packed into a selection of bins (V). An assortment of bins of different sizes (D) is packed with many items of many different sizes (M). Note that the classical *cutting-stock* problem differs from the VBPP in the number of different bin sizes and the type/number of items.

Genetic algorithms are a stochastic procedure which imitates natural evolution processes of genetic inheritance. Standard GA operators like crossover and mutation can be combined with other heuristic algorithms to form a hybrid algorithm.

In our algorithm we devised chromosome encoding and genetic operators suitable for the VBPP. Falkenauer (1996) showed advantages of grouping encoding for classical bin packing. For the VBPP we extended encoding of bin groups to handle different capacities. Different encoding resulted in new VBPP GA operators, evaluation function, adaptation procedure and other parts of the GA framework.

2.1 Encoding

The efficiency of crossover can be improved by the use of the grouping method in the chromosome. If standard encoding is used in the Bin Packing Problem (BPP), the crossover of two parents will not generate satisfactory offspring, as the gene will be out of sequence, as soon as only one item is replaced from one parent into another. Item grouping is done through the lookup-table. A chromosome is represented as a series of bins. Each bin has the following properties:

Capacity prescribes the maximum sum of the item sizes. The capacity of a bin is normally assigned before any of the items are filled in. Several heuristic methods can be used in capacity assignment. The list of available capacities is usually short, but there is no limit for the VBPP on the number of bins of each capacity.

Items make up an indexed list with references to the item sizes. It can also contain other pieces of information such as assigned/available items.

There can be a different number of bins in the chromosome, which means that encoding is variable-sized. This differs from standard binary encoding, which has a fixed chromosome length. In binary encoding, the chromosome string often has to be corrected/adapted to represent a valid solution. Genetic operators like crossover can disrupt binary encoding. A single bit change can bring about a completely different and possibly invalid solution in bin packing difficult to adapt to a valid representation. Variable-sized encoding collects items into groups (bins) to which GA operators are subsequently applied. An example of chromosome encoding with two bins of capacity 100 and 120 can be represented as

```
{100; 2, 3, 4, 7}, {120; 1, 5, 6}
```

The braces group items into bins. Figures after the semicolon represent item numbers in the list of sizes. As the bin contains only references to the sizes it must be guaranteed that the sum of all item sizes is not larger than the capacity. In order to ensure valid chromosome encoding, the item number should occur only once. Furthermore, all item numbers should be used. Gene (bin) ordering has no impact on the solution. In addition, item references can be unordered in the gene.

2.2 The framework of the VBPP genetic algorithm

As a population heuristic method, GA works in various steps which are repeated several times to reach the desired aim when searching for the best solution. Several GA strategies can be applied. General steps used in the VBPP are described below:

Step 1. The initial population is generated as a set of valid chromosomes which are used later in the evolution loop. Any approximation algorithm can be used. The only requirement is that it is capable of producing valid chromosomes which are diversified in search space. We implemented VBB_k (Burkard and Zhang 1997), which was also used for adaptation described in 2.3.1.

Step 2. Evaluation determines the fitness of each individual with respect to the predefined goal.

Step 3. Selection (duplication) generates an intermediate generation with a doubled number of individuals. The probability of copying a member into the intermediate generation is defined by the member fitness. *Stochastic Universal Sampling* (Whitley 1993) was used in our implementation of the VBPP.

Step 4. Crossover is an operation which passes properties of the parents to their offspring. Genes from both parents are used when combining a chromosome. Pairs of parents from duplication are used to generate a single child. The following generation has the same number of individuals as the previous one.

Step 5. Mutation is applied to introduce diversity into the population by randomly changing some members of the population. The mutation rate is normally low to avoid a disruption of the sought goal.

Step 6. Stop criteria should be verified with Step 2. If the best member meets the stop criteria, GA stops, otherwise it continues with Step 3.

2.3 Crossover

In general, genetic algorithms use two types of operators: crossover and mutation. Non-binary encoding also employs non-standard implementation of GA operators. Item grouping into bins implies a special treatment of the crossover operator. Uniform grouping two-point crossover was developed for the VBPP. The procedure of this operator applied to two chromosomes is as follows:

Step 1. Choose two random numbers in the range $[0, s_1]$, where s_1 is the number of the bins in the first chromosome. These two numbers represent the crossing sites between the genes. The difference between the sites should be at least one. Note that every chromosome has $s + 1$ crossing sites.

Step 2. Select bins out of the first chromosome within the crossing sites specified in the previous step. Eliminate the second chromosome bins which contain items from the selected bins of the first chromosome.

Step 3. Inject the selected bins at the crossing site position specified by the random number in the range $[0, s_2]$, where s_2 is the number of the bins in the second chromosome.

Step 4. Some items could have been removed by Step 2 and are missing from the solution. Apply the adaptation procedure to repack the eliminated items. Adaptation can also be useful in local optimisation.

We next show an example of the described crossover on two chromosomes with the genes represented by letters: the first chromosome with uppercase letters `ABCDEF` and the second with lowercase letters `abcd`. Each letter represents one bin with the items as described in 2.1.

Step 1. Choose two random sites in the first chromosome. The crossing sites are represented by the bars `A|BCD|EF`. The selected part of the chromosome is `BCD`, with each letter containing several items.

Step 2. Some items in the genes can be duplicated and must be removed from the solution to have a valid chromosome after the injection. Remove all bins with duplicated items from the second chromosome. Let us suppose that some items from the bins `a` and `c` contain items from the selection `BCD`. We remove whole bins, what remains is the partial chromosome `bd`.

Step 3. Insert content between the sites from the first parent into the random site of the second partial parent `|bd`. Injection would produce a new chromosome `BCDbd`.

Step 4. Some items from `a` and `c` are missing from the new chromosome. Use the adaptation procedure to repack all unassigned items. If possible, correct the chromosome according to bin size limits and minimise the cost function. At this stage, local optimisation is performed with the application of a heuristic algorithm.

2.3.1 Adaptation

Local optimisation is used to repack the eliminated elements and possibly improve the already packed bins. Adaptation for the VBPP is divided into two consecutive procedures.

First, we try to improve the existing bins with the eliminated items. Each bin should be repacked by one, two or three items with one larger item or pair of items which are unassigned. An ordered table of all possible sums of sizes for the pair of unassigned items is created. The sums are associated to the pair of items. The heuristics repeated for every bin is as follows:

Step 1. If there is no spare size, skip the bin, as there can be no improvement.

Step 2. Try to replace up to three items which will be replaced with one item without splitting the size of the bin.

Step 3. Try to find two or three items which will be replaced by two items whose sum is larger and still fits into the bin. If replaced, update the lookup table for pair sums.

Step 4. If replacement occurs, repeat the procedure.

The described procedure prefers the replacement of a lower number of items.

After the improvement stage we proceed with the packing of yet unassigned items into new bins. The problem of packing unassigned items corresponds to the

VBPP with a substantially reduced problem-size from the original problem. There are several algorithms, mostly for online cases. We implemented VBB$_k$ (Burkard and Zhang 1997) and a subset sum on available bin sizes. The first algorithm opens a new bin of the maximum size if the current item size is smaller than half the maximum bin size. Otherwise, it opens the smallest bin that can accommodate an item. Subsequent items are packed into the best-fit one of the most k opened bins (the B rule). This k-bounded algorithm closes a bin which is not of the maximum size, if there is one. Otherwise it closes the most nearly full bin of the maximum size (the best-bin closing rule).

In the second algorithm we opened a new bin by trying to find the bin size for which a subset sum of item fills the best bin. When the best bin size is found, all unassigned items found in the subset sum are packed into it, the bin is then appended to the chromosome, the problem is relaxed and repeated for the remaining unassigned items. The algorithm for the subset sum is based on the separability property by Horowitz and Sahni (1974).

2.4 Mutation

Apart from crossover, mutation is another important GA operator which is used to disturb the convergence of the population with the aim of a broader search. Typically, the mutation of the chromosome occurs at a given rate of a very low percentage. The mutation procedure is as follows:

Step 1. Eliminate the worst-fitness bin in the chromosome.

Step 2. Randomly eliminate some bins. The number of bins removed should be small enough, so that the chromosome will not lose significant chromosome information.

Step 3. Apply the adaptation procedure as described in 2.3.1 on the eliminated items to build a valid chromosome.

2.5 Evaluation function

Approximation algorithms for the VBPP normally specify performance as the ratio of the item sum and the sum of the bin sizes. Using such evaluation function will lead into a huge amount of chromosomes with equal fitness. To promote well-filled bins, the evaluation function, which we tried to maximise, is defined as

$$f_{\text{VBPP}} = \frac{\sum_{i=1..N}(F_i/C_i)^k}{N} \quad , \tag{1}$$

where N is the number of bins used, F_i the sum of the items in i-th bin, and C_i the capacity of the i-th bin, and $k > 1$. The evaluation function thus averages individual bin fitnesses (F_i/C_i) which are also used in repacking. Constant k gives priority to well-filled bins. This is a user-selectable constant with practical values around two.

2.6 Illustrative example

Let us show the operations of the algorithm on a simple example. A bill of materials requires the following ten item sizes {22, 92, 10, 40, 33, 90, 36, 28, 86, 73}, which are indexed from 0 to 9. These items can be cut from any beam of a standard size {100, 120, 150}.

First, we generate an initial population using VBB_5. For the generation of individuals we shuffle items and then apply VBB_k. The result of VBB_k packing and bin size selection depends on the order of items. The first individual generated from the initial order of items is packed into five bins:

```
150(124): 0, 1, 2          (22, 92, 10)
150(137); 3, 4, 6, 7       (40, 33,  36, 28)
100(100); 2, 5             (10, 90)
100(86);  8                (86)
100(73);  9                (73)
```

The numbers in parenthesis are the sum of the item sizes in the bin, and individual item sizes. Using encoding described in Subsection 2.1 this individual can be written in a shorter way as {150; 3, 4, 6, 7}, {100; 2, 5}, {100; 8}, {100; 9}

For every individual the fitness function is evaluated. Using equation (1) for the first individual yields 0.72 fitness. Selection (duplication) into the intermediate generation is applied with stochastic universal sampling. The probability of copying a member into the intermediate generation is defined by member fitness.

Crossover is then applied to every pair of the intermediate generation. Let us suppose that the first individual is mated with the following bins also generated with VBB_k {100; 1}, {150; 3, 7, 9}, | {100; 2, 5}, | {100; 0, 4, 6}, {100; 8}.

The two bars represent the two crossing sites. The bins between the sites will be injected into the first chromosome. Before continuing, we must remove all the bins from the first chromosome which contain items 2 and 5. After eliminating the bins {150; 0, 1, 2} and {100; 5}, we insert the bin {100; 2, 5} at a random site of the first chromosome. The resulting chromosome {150; 3, 4, 6, 7}, {100; 2, 5}, {100; 8}, {100; 9} lacks items 0 and 1.

Before repacking the missing items, we apply the improvement stage in which we want to replace smaller items with larger unassigned items without splitting the bins. For the first bin we can find that items 3, 4 and 7 can be replaced with items 0 and 1. With this replacement the bin is filled to capacity. The chromosome now lacks items 3, 4, 7. The missing items cannot improve the existing bins.

The remaining items are packed into new bins whose sizes are determined by repeatedly solving the subset sum problem for all standard sizes and then select the bin with the best fitness. The bin size 120 is selected for the remaining items resulting in a valid chromosome {120; 3, 4, 7}, {150; 0, 1, 6}, {100; 2, 5}, {100; 8} {100; 9} with improved fitness ($f = 0.80$). Alternatively, one can apply VBB_k for the remaining items, as the subset sum can be a bottleneck for large problems. Fitness can be additionally improved by checking every bin for a smaller standard bin size that still fits the items.

Repeating crossover for the remaining part of the population as well as applying mutation will iterate the population to better individuals. For our example the following result with fitness $f = 0.97$ was found:

```
150(141); 0,  4,   8       (22, 33, 86)
150(149); 3,  6,   9       (40, 36, 73)
120(120); 1,  7            (92, 28)
100(100); 2,  5            (10, 90)
```

3 Inventory status data and cutting

In beam cutting, trim loss can be of significant sizes even when several standard lengths are used in optimisation. Trim-loss items can be so large that their storage and reuse in future optimisations is justified. According to their size, trim-loss items can be classified as:

1. Small items of trim loss – these are unlikely to be reused and are regarded as waste.

2. Medium-sized items of trim loss or *remnants* – these are of larger sizes than those of small trim loss items. They are are assigned to inventory status data for reuse in later optimisations.

3. Large items of trim loss or *large remnants* – these are normally generated from the last bin which has been newly opened and is not entirely filled. Like medium-sized remnants, they are assigned to inventory status data.

The store should only be used for a small number of items. In order to maintain a reasonable store size we must constantly seek possibilities for the reuse of the stored items. Thus it is justified to employ inventory status data in an optimisation process which first uses only the stored remnants. The problem of cutting with inventory status data can be formulated as the filling of a finite number of sacks with a number of item sizes which exceed the capacity of all the sacks. The MSSP, which covers cutting with inventory status data, is given a detailed description in Subsection 3.1. With this kind of optimisation one must take heed of special cases such as ruling out the items which are larger than any item in the inventory status data. Further on, we disregard the stored items which are smaller than the smallest item due for optimisation.

Optimisation with inventory status data can be regarded as a preprocessing stage to the VBPP and tries to consume as many items in the inventory status data as possible. The items which have not been used at the preprocessing stage are passed on to VBPP optimisation. Optimisation with inventory status data consists of the following salient steps:

Step 1. Find the minimum item size to pack for the selected beam cross-section. Select the items from the inventory status data with capacity that can accommodate the minimum item to pack. Rule out all the items which are larger than the largest available item in the inventory status data. This step removes evidently unfit items from optimisation. Ruled-out items are left over to VBPP optimisation.

Step 2. The MSSP algorithm is applied to the reduced set of items.

Step 3. Items not assigned in Steps 1 and 2 are used in VBPP optimisation.

3.1 The Multiple Subset Sum Problem

The Multiple Subset Sum Problem (MSSP) (Caprara et al. 1998) is a special case of the *Multiple Knapsack Problem*, where the number of sacks (bins) is a fixed small number. We are dealing with a problem where n items are packed into m sacks of different capacities $c_i, i = 1, \dots, m$. The problem is to choose m different subsets of items in such a way so that every subset i fits into the sack i and that the profit is maximised. Formally, the MSSP can be defined as

$$\begin{aligned}
\text{maximize} \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} w_j x_{ij} \\
\text{subject to} \quad & \sum_{j=1}^{n} w_j x_{ij} \le c_i, \quad i = 1, \dots, m, \\
& \sum_{i=1}^{m} x_{ij} \le 1, \qquad j = 1, \dots, n, \\
& x_{ij} \in \{0, 1\}, \quad i = 1, \dots, m,\ j = 1, \dots, n
\end{aligned} \tag{2}$$

Applying the VBPP approach to the problem, one can also adapt the algorithm to handle the MSSP. The difference is in handling items and bins. A new bin that is being allocated has been selected from the pool of unassigned bins, with the VBPP limiting only the bin sizes, not the number of bins. Packing is finished when there are no available bins, yet there can still be some unassigned items left to the VBPP procedure. Crossover and mutation operators are basically the same, except that the second stage of the adaptation employs modified packing, as the need for selecting the best bin is ruled out.

Normally, not all of the items can be packed into the available bins. Even with a trivial case where the items are all packed from inventory status data it is recommended to use the optimisation algorithm as the fitness function promotes well-fit bins.

3.2 Cutting addition

When cutting beams, it is also the cut width that has to be taken into consideration. This is important when there are several cuts whose sum total exceeds beam size tolerances. Optimisation with a cutting addition can be performed using a simple transform as shown in Figure 2.

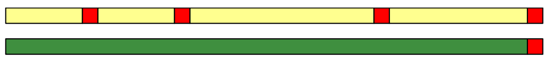

Figure 2: The cutting addition transform adds the cut size to both the items (top) and the bin (bottom)

This transform shows that every item gets a cutting addition. In order to ensure that the last item manages to fit into the bin even if there is no cut at the end, the bin capacity is extended by a cutting addition. This transform is applicable to VBPP and MSSP optimisations. The inverse transform is straightforward.

4 Experiments

Several experiments were performed to verify the search performance of the proposed algorithm. However, no publicly available test data could be found that would cover the problem as described above. There are several test data (Beasley n.d.) for classical bin packing that can be used as the VBPP input. Bin capacities must be re-specified separately, since only one capacity is needed for the classical problem. In order to test the MSSP part of the algorithm, data problems of the Multiple Knapsack Problem (MKP) can be used. The problem is relaxed to the MSSP by setting the profit part of the MKP equal to the weight. In general one can use BPP data for item sizes applied to both subproblems.

In dealing with the described problem we used data by Falkenauer (1998) and by Scholl and Klein (1997). Both sources provide data for a classical problem. This test problem was chosen for its availability and comparability with a similar approach. Note that a comparison with classical bin packing can be implemented only for a special case of the VBPP, i.e. single bin capacity. Selected tests consist of a uniform distribution of item sizes in the range $[20 \dots 100]$ and a bin capacity of 150. There are no small items in order to make test instances harder, since small items only improve the overall performance. Problems are grouped according to the number of items (120, 250, 500 and 1000). Each group consists of 20 problem instances.

The experiments were carried out on an office computer with 450 MHz processor (Intel Celeron). The programme code for optimisation engine is written in C++ with standard template library and thus portable to several platforms.

Memory requirements for a running process are low (2 to 10MB, depending on the problem size).

Basically, we were interested in the quality of packing. This means that we tried to improve the initial solution generated by VBB_k heuristics. The first tests were aimed to verify the performance of our approach with the results reported by Falkenauer (1996), where he claims the superiority of his approach over the best heuristics in the operational research. We set our algorithm parameters to match those of Falkenauer in terms of the population size (67) and limited the maximum number of generations to 2000. The mutation rate was one percent, with 10 percent of the chromosome affected.

Our algorithm found all optimal solutions for problems with 120 instances. This also included two solutions for which Falkenauer conjectures that there is no optimal solution. The running time for these instances was better even on inferior architecture, since the two optimal solutions did not hit the limit of the maximum number of generations. With 250 instances the quality and time were similar. With larger instances our algorithm was notably slower but with a comparable quality of packing. Experimenting with the programme code showed that with larger instances subset sum repacking consumes more time. This is due to a larger number of unpacked items. In faster repacking of larger unassigned instances we introduced simpler heuristics VBB_k, which can be selected at the user's request.

It should also be noted that in classical bin packing, the stopping criterion is easily set as a theoretical number of bins calculated as the quotient of the sum of item sizes over the bin capacity. If the required optimisation is set to prefer well-filled bins as is stated in the evaluation function (1), the stopping criterion cannot be a simple fraction. For this kind of optimisation the maximum number of generations is the most straightforward stopping criterion.

The second tests were performed on the same data with additional capacities. Compared with the initial population, these tests showed a relative improvement in packing. We performed runs for each problem group and analysed the fitness function progress through the generations. Figure 3 shows courses of evolution of 20 test instances with 120 items. Similar courses were obtained for larger instances. Evolution produced up to a 5% improvement on the initial solution, which can be considered as a valuable improvement in production. Note that the test instances lack small items, which improve fitness. In addition, figure 3 shows that it is in the first few generations that improvement can be substantial. The following

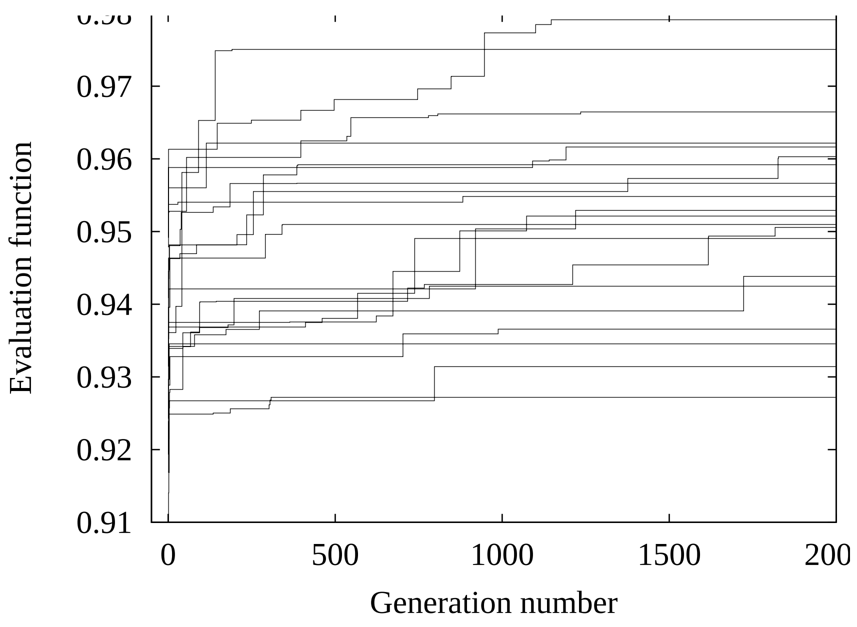

Figure 3: Courses of evolution for the VBPP (120 items)

generations also provide improvements, which leads to the conclusion that the algorithm does not suffer from premature convergence, which is sometimes a major obstacle for the GA and other stepping algorithms. In some instances improvement is considerably constant over the whole evolution. The time complexity roughly depends on the number of items, the number of bin capacities, the population size and the number of generations. The running time for 120 items with 3 capacities is about one minute. A larger number of instances/capacities can increase this time by hours.

The MSSP benchmark was not performed on the described test data for a simple reason that the problem instances do not describe a typical item-size distribution which is production dependent. The type of inventory handling (the minimum remnant size for reuse) also has impact on the distribution. In general, we can conjecture that item sizes will be small with some large items, i.e. remnants from the last bin which has been only partially used and returned to the inventory. As the difference in the MSSP is only in the limited number of capacities, one can expect similar performance as in the VBPP. In order to experiment with all parameters of practical importance we implemented an application which was installed at the local producer of structural steel buildings.

4.1 Case study

The application programme in industrial implementation involves an optimisation programme which is able to consider the majority of input data situations. This is especially the case with the MSSP part of optimisation, with optimisation depending on the size and type of items in the inventory.

The application programme was implemented to provide the described optimisation functionality. Since optimisation is numerically intensive, the application programme was divided into the engine and the user interface. The engine interface written in C++ provides anonymous pipe communication to other programmes (GUI). The user interface in Visual Basic enables the navigation of data flow from various input sources from/to a common company database or local ODBC connection. After the initial installation it transpired that the user needs full control over the optimisation process and that process tuning is an important issue. The selection of items to optimise can be specified by using SQL which offers the user a choice of a complex specification of the WHERE clause. This typically includes the shape, order and the drawing number. Several shapes can be selected, which leads to several consecutive optimisations which can be easily parallelised on machines with multiple CPUs and supported operating systems such as Windows NT.

Additional problems were caused by inventory communication, since item storing must be tailored to the needs of reservation, trial runs and optimisation discards. The selection of items from the inventory can be another critical point which often requires the user's decision. The number of stored items can grow to the point where MSSP optimisation can choke. Further, there can be numerous items of standard lengths among them. There are also cases where the MSSP can dominate the selection, leaving a small number of items to the VBPP, but the packing quality is unacceptable, and the user is forced to drop some parts or the whole of MSSP propagating optimisation to the VBPP.

When faster response time is preferred for a preliminary calculation, the user can limit the number of generations and vary the population size. While observing the fitness function, the user can stop the process and settle with the last best solution. This also means that the optimisation time can be manually controlled, depending on the user's needs.

5 Conclusions

We have shown how the hybrid genetic algorithm can be used as heuristics which provides quality packing for cutting. To efficiently tackle the problem, practical requirements have to be taken into consideration. Domain-specific knowledge and local hill-climbing in the genetic algorithm has turned out to be helpful in many aspects of the optimisation process.

It is important to notice that the presented algorithm tends to minimise the waste, as it promotes well-packed bins. This kind of optimisation thus leaves larger remnants for storage. The search robustness of the GA can be easily adopted for specific fitness functions. For example, disregarding bins with very large remnants when evaluating fitness function.

The GA presented is robust and easy to use, as there are only a few parameters with a large range of successful operation.

The case study has shown that optimisation can substantially help the design and production process in cutting material of steel structures, metal sheets, panels and other one-dimensional elements. Every single percent of improvement in cutting can be beneficial for the producer. Reductions in time and cost can be significant, and the motivation for coding better algorithms to meet specific needs can be easily estimated. A major requirement for such software is an effective support of the user's interaction.

References

Beasley, J. E., Or-library. http://www.ms.ic.ac.uk/info.html.

Burkard, R. E. and Zhang, G., 1997, Bounded space on-line variable-sized bin packing. *Acta Cybernetica*, **13**(1), 63–76.

Caprara, A., Kepler, H. and Pferschy, U., 1998, The multiple subset sum problem. Technical report, Faculty of Economics, University of Graz.

Caprara, A., Kepler, H. and Pferschy, U., 1999, A PTAS for the multiple subset sum problem with different knapsack capacities. Technical report, University of Graz, Austria.

Dyckhoff, H., 1990, A typology of cutting and packing problems. *European Journal of Operational Research*, **44**(2), 145–159.

Falkenauer, E., 1996, A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, **2**(2), 5–30.

Falkenauer, E., 1998, *Genetic Algorithms and Grouping Problems.* John Wiley & Sons, Wiley Computer Books.

Friesen, D. K. and Langston, M. A., 1986, Variable sized bin packing. *SIAM Journal on Computing*, **15**(1), 222–230.

Gröetschel, M., Lovász, L. and Schrijver, A., 1993, *Geometric algorithms and combinatorial optimization.* Springer-Verlag, Berlin, New York, etc., 2nd corrected edition.

Horowitz, E. and Sahni, S., 1974, Computing partitions with applications to the knapsack problem. *Journal of the ACM*, **21**(2), 277–292.

Murgolo, F. D., 1987, An efficient approximation scheme for variable-sized bin packing. *SIAM Journal on Computing*, **16**(1), 149–161.

Scholl, A. and Klein, R., 1997, Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research*, **24**(7), 627–645.

Suzuki, J., 1998, A further result on the markov chain model of gas and their application to sa-like strategy. *IEEE Trans. on Systems, Man, and Cybernetics*, pp. 95–102.

Whitley, D., 1993, A genetic algorithm tutorial. Technical Report CS-93-103, Department of Computer Science, Colorado State University.

Zhang, G., 1997, A new version of on-line variable-sized bin packing. *Discrete Applied Mathematics*, **72**, 193–197.