A WEIGHT-CODED GENETIC ALGORITHM FOR THE MULTIPLE CONTAINER PACKING PROBLEM

Günther R. Raidl
Department of Computer Graphics, Vienna University of Technology
Karlsplatz 13/1861, 1040 Vienna, Austria
raidl@eiunix.tuwien.ac.at

Keywords: Combinatorial optimization, hybrid genetic algorithm, weighted coding, multiple container packing problem.

ABSTRACT

This paper presents a genetic algorithm (GA) approach to the multiple container packing problem (MCPP), which is a combinatorial optimization problem comprising similarities to the knapsack problem and the bin packing problem. A novel technique for encoding MCPP solutions is used within the GA: The genotype is a vector of numerical weights associated with items of the problem. The corresponding phenotype is obtained by temporarily modifying the original problem according to these weights and applying a greedy decoding heuristic for the MCPP to the new problem. This solution is then evaluated using the original problem data again. Four different decoding heuristics are discussed. They were tested in a weight-coded steady-state GA on a variety of MCPP instances. Two of the heuristics are clearly more effective than the others, and the GA using them found solutions of significantly higher quality than direct-encoded and order-based GAs from a previous work.

1. INTRODUCTION

The *multiple container packing problem* (MCPP) is a combinatorial optimization problem with applications in various fields such as truck loading and air baggage handling. A set of n items and a set of C equal containers are given. The value v_j ($j = 1, \ldots, n$) and (scalar) size s_j of each item are known. Each container can hold items with a total size up to a maximum container size $S_{\max}$. The goal is to pack a subset of the n items into the C containers in such a way that the total value of all packed items is maximized.

Formally, the MCPP can be stated in this way:

$$\text{maximize} \quad V = \sum_{i=1}^{C} \sum_{j=1}^{n} v_j x_{i,j}, \tag{1}$$

$$\text{subject to} \quad \sum_{i=1}^{C} x_{i,j} \le 1, \quad j = 1, \ldots, n, \tag{2}$$

$$\sum_{j=1}^{n} s_j x_{i,j} \le S_{\max}, \quad i = 1, \ldots, C, \tag{3}$$

$$x_{i,j} \in \{0,1\}, \quad i = 1, \ldots, C,\ j = 1, \ldots, n,$$

$$\text{with} \quad s_j > 0,\ v_j > 0,\ S_{\max} > 0\,.$$

The variables searched for are $x_{i,j}$ ($i = 1, \ldots, C$, $j = 1, \ldots, n$): If item j is to be packed into container i, $x_{i,j}$ is set to 1, otherwise to 0. The n constraints in (2) ensure that each item is packed into a single container only, the C constraints in (3) guarantee that the total size of all items packed into each container does not exceed $S_{\max}$.

The MCPP comprises similarities to the well known *knapsack problem* (KP) and *bin packing problem* (BPP). The KP is equivalent to the special case of the MCPP with only a single container ($C = 1$). This problem is NP-complete, but efficient approximation algorithms have been developed for obtaining near optimal solutions, see e.g. [15]. In the BPP, the goal is to minimize the number of containers necessary to pack all n items while not violating any size constraint. In contrast to the MCPP, the values of items do not play a role. The BPP in its general form is also NP-complete [6]. The MCPP can also be seen as a combination of the KP and the BPP, since the MCPP can be divided into two strongly dependent parts which must be solved simultaneously: (a) select items for packing, and (b) distribute chosen items over available containers. See [20] for a more detailed comparison of the MCPP to other combinatorial optimization problems.

Because of the NP-completeness of the KP and BPP it is obvious that the MCPP is NP-complete [7]. Therefore, exhaustive search algorithms such as branch-and-bound that lead to globally optimal solutions are in general too time-consuming and can only be applied to very small problems. For larger instances of such problems *genetic* *algorithms* (GAs) [1, 8, 16] have already proven to be suitable for finding near-optimal solutions.

This paper presents a GA that encodes candidate solutions via a technique called *weight-coding*: A solution to a combinatorial optimization problem is represented by a vector of weights that modifies the original problem. A non-evolutionary decoding heuristic is applied to get the actual solution which is then evaluated using the original, unmodified problem. Such encodings have already been used successfully in some other problems such as the optimum communications spanning tree problem [18], the rectilinear Steiner tree problem [10], the 3-satisfiability problem [5], the minimum weight triangulation problem [2], and the traveling salesman problem [12, 13]. An overview of various weight-coded GAs is given by Julstrom [11].

The next section provides a short overview of prior GA approaches to the MCPP and the related KP and BPP. Section 3 describes the weighted coding and four possible decoding heuristics. A steady-state GA that uses weight-coding is presented in section 4, and section 5 documents results of an experimental comparison. These experiments indicate that the performance of the weight-coded GA strongly depends on the decoding heuristic. Using a suitable heuristic, the new GA approach outperforms previous hybrid GAs with traditional encodings on nearly all test problems.

2. Prior approaches

Several researchers have developed successful GAs for the KP and the more difficult multi-constraint KP, including Chu [3], Chu and Beasley [4], Hinterding [9], Khuri et. al. [14], Olsen [17], and Raidl [19]. Falkenauer [6] presented a hybrid GA for the BPP.

In [20], Raidl and Kodydek observed that these GA approaches can be divided into two categories according to the solution encoding techniques: Some algorithms use *direct encoding* (DE), meaning that a chromosome of the GA contains a gene for each item indicating directly if the item is supposed to be packed into the knapsack (or into which container the item should be packed). In this case, infeasible solutions must be handled by using a repair algorithm or adding a penalty term to the objective function. On the other hand, some GAs use *order-based encoding* (OBE) in which a chromosome contains a permutation of all items. The actual solution is obtained by applying a first-fit algorithm: In the order given by the permutation, one item after the other is inserted into the initially empty knapsack (containers) until a capacity constraint is violated. When applying OBE, special recombination and mutation operators such as order crossover and swap mutation (see [1, 16]) must be used to generate new chromosomes that contain valid permutations again.

Starting from the ideas of the various GAs for the KP and BPP, Raidl and Kodydek [20] developed two effective GA variants for the MCPP based on these encoding schemes. Additionally, problem-specific knowledge was incorporated into both approaches using local improvement operators: Each newly generated solution may be improved by trying to pack its unpacked items into a container that has not reached its maximum total size yet. Such an improvement is performed in a Lamarckian way, therefore, the genotype is changed accordingly. The experimental comparison in [20] indicates better performance for the OBE approach in case of fewer items and for the DE approach in case of larger problems. Local improvement operators lead in many cases not only to better results, but also to shorter running times because of faster convergence.

```
procedure Heuristic $\mathcal{A}$;
for all containers $i$ do
  $p \leftarrow 0$; /* occupied size of container */
  for all unpacked items $j$ sorted according to
        decreasing $v'_j$ (or $r'_j$) do
    if $p + s_j \leq S_{\max}$ then
      pack item $j$ into container $i$;
      $p \leftarrow p + s_j$;
done;
```

Figure 1: *A greedy decoding heuristic for the MCPP.*

3. A Weighted Coding for the MCPP

In a GA, a solution to the MCPP can be represented by a weight vector $\vec{w} = (w_1, w_2, \ldots, w_n)$. Weight w_j is associated with item j. To decode such a chromosome, a modified problem is generated by adding these weights to the relative item values $r_j = v_j/s_j$ (v_j: absolute item value, s_j: item size):

$$r'_j = r_j + w_j \,. \tag{4}$$

While item sizes remain unchanged, new absolute item values are derived as follows:

$$v'_j = r'_j s_j = v_j + s_j w_j \,. \tag{5}$$

A second step applies a greedy decoding heuristic for the MCPP to the modified problem to obtain the phenotype. Finally, a fitness value is determined for the phenotype using the original, unmodified problem data.

A suitable decoding heuristic should not be too time-demanding since it must be performed for each new chromosome in the GA. Further, it should provide strong locality in the sense that small variations of the genotype (and therefore of the problem) will usually lead to relatively small changes in the phenotype. The pseudo-code of two good heuristics is shown in Figs. 1 and 2 and discussed in the following sections.

3.1. Decoding Heuristic $\mathcal{A}$

The first greedy heuristic for the MCPP is straightforward. One container after the other is filled by going through all unpacked items and packing all items not violating the size constraint into the current container. An essential decision in this algorithm is the order of processing the items. Clearly, items processed first are more likely to fit into a container than items coming later. Since we want to maximize the total value of all packed items, valuable items should be favored and ranked at the beginning. An obvious processing order is obtained by sorting the items according to decreasing absolute values v'_j. Equally valuable items are ranked in random order.

On the other hand, favoring items with high values v'_j is not always a good decision: High-valued items can be expected to have above-average sizes, and fewer items will therefore fit into a container. Items with average or even small values but very small sizes might often be better choices. Therefore, it makes also sense to sort the items according to decreasing relative values r'_j.

To ensure that the GA is in principle capable of generating the globally optimal solution to an MCPP instance, all possible item permutations must be reachable during decoding. This means that the range $[-W, W]$ of possible weight values w_j must be large enough so that each item can become both most and least valuable in the modified problem. On the other hand, the range of values should not be unnecessarily large so that the search space of the GA remains as small as possible. For a GA with the relative value (r'_j) ordering heuristic $\mathcal{A}$, the smallest W is therefore

$$W = \left(\max_{j=1,\ldots,n} r_j - \min_{k=1,\ldots,n} r_k \right) \Big/ 2 \,. \tag{6}$$

The smallest W for a GA with absolute value (v'_j) ordering heuristic $\mathcal{A}$ is

$$W = \max_{j=1,\ldots,n} \left((v_{\max} - v_j)/s_j, (v_j - v_{\min})/s_j \right) \tag{7}$$

$$\text{with} \quad v_{\max} = \max_{k=1,\ldots,n} v_k, \quad v_{\min} = \min_{k=1,\ldots,n} v_k.$$

3.2. Decoding Heuristic $\mathcal{B}$

Decoding heuristic $\mathcal{B}$ (Fig. 2) is more sophisticated; it fills the containers in parallel. For one item after the other, the container where the item fits best is identified. This is the container where the capacity constraint would not be violated and the least space would remain when adding the item. If such a container exists, the item is packed into it; otherwise, it remains unpacked. The algorithm tries to simultaneously exploit the remaining space in all containers as well as possible. As in heuristic $\mathcal{A}$, the order of processing the items is critical and more valuable items should be tried first. Therefore, the items are again sorted according to decreasing absolute values v'_j or relative values r'_j.

Again, the globally optimal solution can in principle be generated by a GA using heuristic $\mathcal{B}$ if all possible permutations can be achieved as item processing orders. The range of values $[-W, W]$ for the weights w_j should therefore be the same as for heuristic $\mathcal{A}$, as described in Eqs. (6) and (7).

```
procedure Heuristic $\mathcal{B}$;
$\vec{p} \leftarrow \vec{0}$; /* occupied sizes of containers */
for all items $j$ sorted according to
        decreasing $v'_j$ (or $r'_j$) do
    /* search container $c$ where item $j$ fits and
        least space $cs$ remains */
    $c = 0$;
    $cs = S_{\max}$;
    for all containers $i$ do
        if $(p_i + s_j \le S_{\max}) \wedge (S_{\max} - p_i - s_j < cs)$ then
            $c \leftarrow i$;
            $cs \leftarrow S_{\max} - p_i - s_j$;
    if $c > 0$ then
        pack item $j$ into container $c$;
        $p_c \leftarrow p_c + s_j$;
done;
```

Figure 2: *A more sophisticated decoding heuristic.*

The computational effort of both decoding heuristics is $O(n \log n)$ for sorting the items plus $O(C\,n)$ for the two nested loops. For larger n and problems where most items can be packed into containers, heuristic $\mathcal{A}$ can be implemented slightly more efficiently than heuristic $\mathcal{B}$ if a heap is used for storing all unpacked items in sorted order during the inner loop.

4. The Weight-Coded GA

The GA used for the experiments the next section describes is a traditional steady-state GA with binary tournament selection. Within a chromosome, weights w_j are directly stored as real values. Initial solutions are generated by assigning each weight a random value from the interval $[-W, W]$. In early experiments, uniform crossover applied with a probability of 50% proved to be slightly better than one- or two-point crossover. The mutation operator randomly replaces a value with a new random value with a probability of $3/n$. A smaller probability increases the danger of premature convergence; a much larger probability degrades performance.

It proved to be essential to disallow duplicates in the population. This is accomplished by using a replacement scheme that only accepts new solutions different from all others in the population. The test for equality is efficiently performed on phenotype level using a hash table. If a new solution is not a duplicate, it always replaces the solution with the worst fitness.

Preliminary experiments indicated that a population size of 100 works well with problems of very different sizes and properties. Each GA run terminated when 200,000 solutions have been evaluated without finding a new best

Table 1: *Gaps of best-of-run solutions with needed numbers of evaluations evals and CPU times t obtained by weight-coded GAs using heuristics $\mathcal{A}$ and $\mathcal{B}$ with absolute and relative value item ordering (average values from 10 runs/problem).*

Problem	WE$\mathcal{A}v$			WE$\mathcal{B}v$			WE$\mathcal{A}r$			WE$\mathcal{B}r$		
n / C / $S_{\max}$	*gap*	*evals*	t[sec]	*gap*	*evals*	t[sec]	*gap*	*evals*	t[sec]	*gap*	*evals*	t[sec]
30 / 3 / 100	2.74	5680	0.6	2.74	8860	1.2	2.74	380	0.1	2.74	540	0.1
30 / 6 / 100	2.32	11280	1.7	2.73	9520	1.5	2.32	4280	0.7	2.32	3040	0.5
30 / 9 / 100	2.90	2220	0.3	2.90	1440	0.2	2.90	5620	0.9	2.90	3160	0.6
30 / 12 / 100	1.05	27740	4.4	1.15	98900	16.9	1.05	12280	1.9	1.05	6520	1.2
50 / 5 / 100	3.89	34940	7.4	3.89	40260	9.0	2.08	3760	0.9	2.08	3420	0.9
50 / 10 / 100	3.38	41420	8.5	3.38	14000	3.4	0.87	79840	20.7	0.87	104860	29.2
50 / 15 / 100	2.16	44560	10.3	2.16	62660	16.2	1.67	42740	10.4	1.67	17400	5.2
50 / 20 / 100	1.05	41080	9.3	0.99	28260	8.0	1.05	144860	41.5	0.94	74460	24.1
200 / 20 / 100	2.45	25260	28.6	2.44	43180	49.7	1.34	119980	135.6	1.34	52080	65.7
200 / 40 / 100	2.00	123280	144.6	1.91	79900	110.6	1.26	404800	522.5	1.18	251540	382.4
200 / 60 / 100	1.67	326520	492.2	1.58	280500	459.7	1.06	819100	1237.0	0.87	446520	804.8
200 / 80 / 100	2.14	160220	263.9	2.01	109640	213.7	1.64	810300	1346.9	1.38	267920	574.5
30 / 3 / 200	1.67	21980	3.0	1.67	17480	2.5	0.56	4280	0.7	0.56	5920	1.0
30 / 3 / 300	1.45	78780	9.8	1.47	69540	10.3	0.40	79280	11.1	0.40	71200	11.8
30 / 3 / 400	0.33	8940	1.1	0.30	83680	12.4	0.26	55400	9.1	0.26	67060	11.0
50 / 5 / 200	1.16	28620	5.7	1.09	71420	15.4	0.18	46360	11.6	0.18	55160	14.3
50 / 5 / 300	0.80	168540	34.3	0.83	99480	21.5	0.06	93980	24.4	0.06	111340	28.9
50 / 5 / 400	1.34	216520	49.5	1.25	114320	25.4	0.17	112920	28.4	0.17	91880	23.8
200 / 20 / 200	1.42	338160	320.1	1.36	300660	353.2	0.27	316380	342.4	0.25	274480	339.5
200 / 20 / 300	1.91	586920	551.5	2.00	581160	670.5	0.14	594560	583.9	0.14	371940	450.5
200 / 20 / 400	1.37	559760	501.4	1.36	446600	505.4	0.13	402040	467.0	0.10	338400	416.4
Average	1.87	135830	116.6	1.87	121974	119.4	1.06	197769	228.5	1.02	124707	151.7

solution. This criterion ensures sufficient convergence in practice. For many of the experiments, the GA could have been stopped much earlier, but we were primarily interested in finding high-quality solutions and only secondarily in CPU times.

5. Experimental Comparison

The test problem set of Raidl and Kodydek [20][1] was used to evaluate the new weight-coded GA variants. This set consists of 21 problems with different numbers of items ($n = 30, 50, 200$), different numbers of containers ($C = 3, \ldots, 80$), and different container capacities ($S_{\max} = 100, \ldots, 400$). Item sizes s_j were randomly chosen out of the interval $[5, 95]$, giving an average item size of $\overline{s} = 50$. The item values v_j were generated by multiplying the size s_j of each item by a relative item value r_j randomly taken from $[0.8, 1.2]$. More details about these test problems can be found in [20].

Since the optimal solution values for most of these problems are not known, the quality of a final solution is measured by the percentage difference (the *gap*) between the solution's total value of packed items V and the optimal value $V_{\max}^{\mathrm{LP}}$ of the LP-relaxed problem. This upper bound can be determined for any MCPP by sorting all items according to their relative values r_j and summing up the item values v_j starting with the most valuable item until a total size $C\,S_{\max}$ is reached. The last item is counted proportionately. Knowing the LP optimum, the percentage difference is: $gap = 100\%\,(V - V_{\max}^{\mathrm{LP}})/V_{\max}^{\mathrm{LP}}$.

[1]This test problem set is publicly available from: http://www.apm.tuwien.ac.at/pub/TestProblems/mcpp

For each problem and each GA variant, 10 independent runs were performed and averaged. Table 1 shows results for weight-coded GAs using the $\mathcal{A}$ and $\mathcal{B}$ decoding heuristics with absolute values v'_j and relative values r'_j as criteria for the order of processing items (algorithms WE$\mathcal{A}v$, WE$\mathcal{B}v$, WE$\mathcal{A}r$, and WE$\mathcal{B}r$). The table contains *gaps* of best-of-run solutions and the numbers of evaluations together with the CPU times for obtaining these solutions. The *gaps* of all four test series are presented graphically in Fig. 3.

In all test problems, the GAs with the heuristics based on relative value item ordering outperformed the heuristics that used absolute value item ordering. WE$\mathcal{A}r$ and WE$\mathcal{B}r$ found solutions of much higher qualities for problems with many items ($n = 50, 200$) and relatively few containers. No significant quality and CPU time differences could be observed between WE$\mathcal{A}v$ and WE$\mathcal{B}v$. Concerning the CPU time, WE$\mathcal{A}r$ and WE$\mathcal{B}r$ proved to be faster than WE$\mathcal{A}v$ and WE$\mathcal{B}v$ for smaller problems and slower for problems involving many items and small containers. In most cases, WE$\mathcal{B}r$ found slightly better solutions or—for the simpler problems—the same solutions than WE$\mathcal{A}r$. On average, but especially for large problems ($n = 200$), WE$\mathcal{B}r$ was also faster and needed fewer evaluations than did WE$\mathcal{A}r$.

Table 2 shows the *gaps* of solutions obtained by applying all four heuristics directly (without any GA) to the original problems. In general, the heuristics using rel-

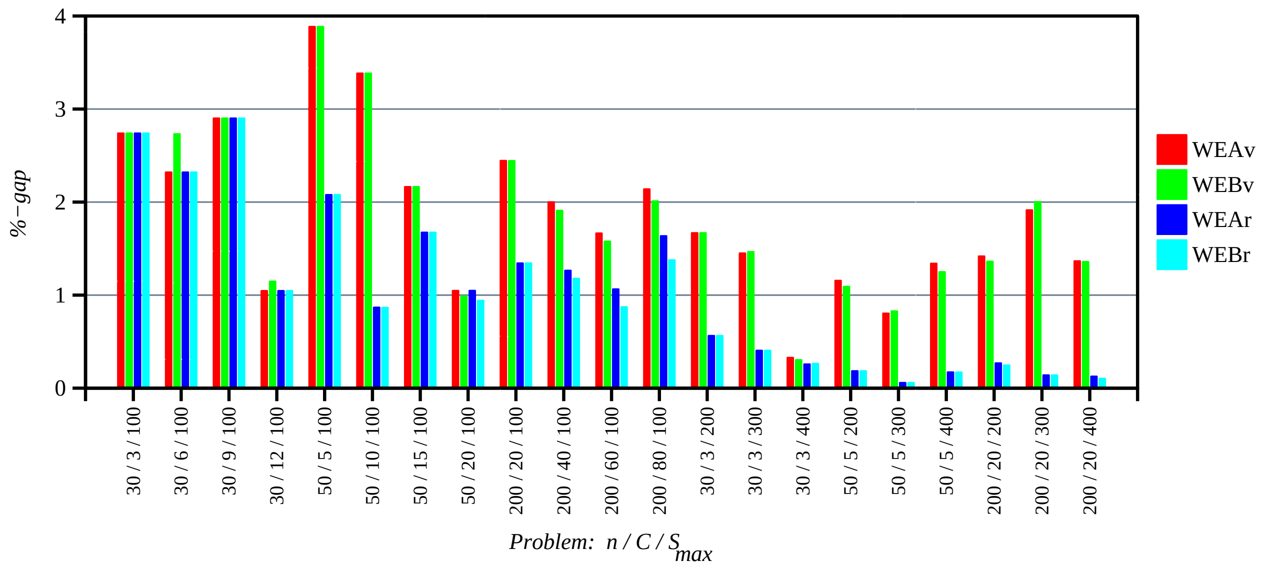

Figure 3: *Average* gaps *of final solutions obtained by the four variants of weight-coded GAs.*

Table 2: *Gaps of solutions obtained by applying heuristics only.*

Problem	$\mathcal{A}v$	$\mathcal{B}v$	$\mathcal{A}r$	$\mathcal{B}r$
n / C / $S_{\max}$	*gap*	*gap*	*gap*	*gap*
30 / 3 / 100	5.64	5.64	5.97	5.97
30 / 6 / 100	13.91	13.91	4.95	4.95
30 / 9 / 100	6.02	6.02	8.26	4.85
30 / 12 / 100	3.81	3.67	8.86	8.86
50 / 5 / 100	5.76	5.76	3.27	3.27
50 / 10 / 100	7.12	7.12	3.14	1.97
50 / 15 / 100	6.60	6.60	8.35	4.24
50 / 20 / 100	4.36	3.81	7.79	6.37
200 / 20 / 100	4.44	4.41	2.84	1.80
200 / 40 / 100	8.29	8.22	4.43	2.71
200 / 60 / 100	6.64	6.51	3.34	2.65
200 / 80 / 100	6.60	6.60	4.76	3.01
30 / 3 / 200	5.37	5.37	2.47	2.47
30 / 3 / 300	5.90	5.90	1.47	1.47
30 / 3 / 400	3.09	3.29	0.75	0.75
50 / 5 / 200	8.98	8.98	2.32	2.32
50 / 5 / 300	3.87	3.87	0.99	0.99
50 / 5 / 400	3.42	3.42	2.03	1.14
200 / 20 / 200	5.01	5.01	0.92	0.85
200 / 20 / 300	4.93	4.82	0.59	0.61
200 / 20 / 400	2.77	2.77	0.59	0.43
Average	5.83	5.79	3.72	2.94

Table 3: *Average* gaps *of final solutions from direct-encoded and order-based GAs (from [20]).*

Problem	DE	DEI	OBE	OBEI
n / C / $S_{\max}$	*gap*	*gap*	*gap*	*gap*
30 / 3 / 100	2.74	2.74	3.16	2.74
30 / 6 / 100	2.69	2.45	2.82	2.32
30 / 9 / 100	3.31	3.01	3.25	2.90
30 / 12 / 100	2.48	1.58	1.42	1.05
50 / 5 / 100	2.60	2.58	2.96	2.28
50 / 10 / 100	1.43	1.10	1.73	1.58
50 / 15 / 100	2.53	1.99	2.35	2.00
50 / 20 / 100	2.58	1.50	2.03	1.67
200 / 20 / 100	1.91	1.65	2.84	2.57
200 / 40 / 100	1.94	1.64	2.52	2.49
200 / 60 / 100	1.99	1.48	2.10	2.28
200 / 80 / 100	2.89	2.09	2.36	2.62
30 / 3 / 200	0.68	0.66	0.96	0.68
30 / 3 / 300	0.50	0.47	0.53	0.44
30 / 3 / 400	0.39	0.37	0.45	0.33
50 / 5 / 200	0.42	0.35	0.88	0.60
50 / 5 / 300	0.51	0.30	0.48	0.70
50 / 5 / 400	0.40	0.39	0.53	0.53
200 / 20 / 200	0.71	0.51	1.26	2.49
200 / 20 / 300	0.45	0.27	0.76	1.64
200 / 20 / 400	0.32	0.25	0.41	0.64
Average	1.60	1.30	1.71	1.65

ative value ordering found significantly better solutions than those using absolute value ordering. The solutions directly obtained by any heuristic were poor compared to final solutions of the various GAs.

Table 3 contains results of the GAs Raidl and Kodydek presented in [20]. These are direct-encoded (DE) and order-based encoded (OBE) approaches with optional local improvement operators (DEI and OBEI). For nearly all test problems the new weight-coded GAs with relative value ordering (WE$\mathcal{A}r$ and WE$\mathcal{B}r$) found solutions with smaller *gaps* than those prior approaches. Furthermore, significantly fewer evaluations were usually needed by all the weight-coded GAs.

A reason for the better performance of the weight-coded GAs seems to be that the recombination and mutation operators are not as disruptive to the phenotypes as in case of the prior GAs. In DE(I) as well as in OBE(I), two genotypically very similar solutions might represent very different phenotypes. The genotype/phenotype mappings of the proposed weight-coded GAs, especially WE$\mathcal{B}r$, seem to have a much stronger locality. Similar genotypes usually map to similar phenotypes, and the recombination can therefore produce offsprings that inherit much of the parental phenotypical structure.

6. Conclusions and Future Work

This paper has presented a new coding of solutions to the MCPP using vectors of weights. For decoding solutions, two substantially different heuristics for the MCPP were described, and each of them can use either absolute or relative value ordering. The coding, the decoding algorithms, and various GA characteristics were tested using 21 different MCPP instances. The results indicate better performance for relative value ordering. The heuristic WE$\mathcal{B}r$, which fills containers in parallel, works in many cases slightly better than WE$\mathcal{A}r$. WE$\mathcal{A}r$ and WE$\mathcal{B}r$ decisively outperform the more traditional direct-encoded and order-based GAs from [20] regarding the quality of final solutions.

Future work will include the examination of other, more sophisticated decoding heuristics that also may involve some kind of local search. Currently, we are trying to understand the implications of using weight-coding with different decoding heuristics for the search space of the GA. Furthermore, similar weight-coded GAs may also be suited for approaching related problems as the (multi-constraint) knapsack problem or the bin packing problem.

References

[1] Bäck T., Fogel D. B., Michalewicz Z.: *Handbook of Evolutionary Computation*, Oxford University Press, 1997.

[2] Capp K., Julstrom B.: A Weight-Coded Genetic Algorithm for the Minimum Weight Triangulation Problem, in *Proc. of the 1998 ACM Symposium on Applied Computing*, ACM Press, 1998, pp. 327–331.

[3] Chu P. C.: *A Genetic Algorithm Approach for Combinatorial Optimization Problems*, Ph.D. thesis at The Management School, Imperial College of Science, London, 1997.

[4] Chu P. C., Beasley J. E.: *A Genetic Algorithm for the Multidimensional Knapsack Problem*, working paper at The Management School, Imperial College of Science, London, 1997.

[5] Eiben A. E., Van der Hauw J. K.: Solving 3-SAT by GAs adapting constraint weights, in *Proc. of the 1997 IEEE Int. Conference on Evolutionary Computation*, Indianapolis, IN, 1997, pp. 81–86.

[6] Falkenauer E.: *A Hybrid Grouping Genetic Algorithm for Bin Packing*, working paper at CRIF Industrial Management and Automation, CP 106-P4, 50 av. F. D. Roosevelt, Brussels, Belgium, 1994.

[7] Garey M. D., Johnson D. S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

[8] Goldberg D. E.: *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison–Wesley, 1989.

[9] Hinterding R.: Mapping, Order-independent Genes and the Knapsack Problem, in *Proc. of the 1st IEEE Int. Conference on Evolutionary Computation 1994*, Orlando, FL, 1994, pp. 13–17.

[10] Julstrom B.: Representing Rectilinear Steiner Trees in Genetic Algorithms, in *Proc. of the 1996 ACM Symposium on Applied Computing*, ACM Press, 1996, pp. 245–250.

[11] Julstrom B.: Strings of Weights as Chromosomes in Genetic Algorithms for Combinatorial Problems, in *Proc. of the 3rd Nordic Workshop on Genetic Algorithms and their Applications*, Vaasa, Finland, 1997, pp. 33–48.

[12] Julstrom B.: Comparing Decoding Algorithms in a Weight-Coded GA for TSP, in *Proc. of the 1998 ACM Symposium on Applied Computing*, ACM Press, 1998, pp. 313–317.

[13] Julstrom B.: Insertion Decoding Algorithms and Initial Tours in a Weight-Coded GA for TSP, in *Proc. of the 3rd Genetic Programming Conference*, Madison, 1998, pp. 528–534.

[14] Khuri S., Bäck T., Heitkötter J.: The Zero/One Multiple Knapsack Problem and Genetic Algorithms, in *Proc. of the 1994 ACM Symposium on Applied Computing*, ACM Press, 1994, pp. 188–193.

[15] Martello S., Toth P.: *Knapsack Problems: Algorithms and Computer Implementations*, J. Wiley & Sons, 1990.

[16] Michalewicz Z.: *Genetic Algorithms + Data Structures = Evolution Programs*, Springer, Berlin, 1992.

[17] Olsen A. L.: Penalty Functions and the Knapsack Problem, in *Proc. of the 1st Int. Conference on Evolutionary Computation 1994*, Orlando, FL, 1994, pp. 559–564.

[18] Palmer C. C., Kershenbaum A.: Representing Trees in Genetic Algorithms, in *Proc. of the 1st Int. Conference on Evolutionary Computation 1994*, Orlando, FL, 1994, pp. 379–384.

[19] Raidl G. R.: An Improved Genetic Algorithm for the Multiconstrained 0–1 Knapsack Problem, in *Proc. of the 1998 IEEE International Conference on Evolutionary Computation*, Anchorage, Alaska, 1998, pp. 207–211.

[20] Raidl G. R., Kodydek G.: Genetic Algorithms for the Multiple Container Packing Problem, in *Proc. of the 5th Int. Conference on Parallel Problem Solving from Nature*, Amsterdam, The Netherlands, 1998, pp. 875–884.