THE MULTIPLE CONTAINER PACKING PROBLEM: A GENETIC ALGORITHM APPROACH WITH WEIGHTED CODINGS

Günther R. Raidl
Institute of Computer Graphics,
Vienna University of Technology
Karlsplatz 13/1861, 1040 Vienna, Austria
raidl@apm.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 problem of choosing C disjoint subsets of n items to be packed into distinct containers, such that the total value of the selected items is maximized, without exceeding the capacity of each of the containers. This so-called multiple container packing problem (MCPP) has applications in naval as well as financial management. It is a hard 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 the 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. Two different techniques for biasing the original problem and four decoding heuristics are discussed. They were tested in a weight-coded steady-state GA on a variety of MCPP instances. One biasing technique and one decoding heuristic turned out to be 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

We consider the problem where n given items should be partly packed into C containers of equal capacity $S_{\max}$. Each item has an associated value v_j $(j = 1, \ldots, n)$ and a (scalar) size s_j. The problem is to select C disjoint subsets of items, such that each subset fits into a container and the total value of the selected items is maximized. There are several applications for this problem in fields such as ship or truck loading, air baggage handling, or deciding how to fill C liquids into n tanks, when the liquids may not be mixed.

Formally, we can define the *multiple container packing problem* (MCPP) by

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

$$\text{subject to} \qquad \sum_{j=1}^{n} s_j x_{i,j} \leq S_{\max}, \quad i = 1, \ldots, C, \tag{2}$$

$$\sum_{i=1}^{C} x_{i,j} \leq 1, \quad j = 1, \ldots, n, \tag{3}$$

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

$$\text{with} \qquad s_j > 0, \; v_j > 0, \; S_{\max} > 0,$$

where $x_{i,j} = 1$ if item j is assigned to container i and $x_{i,j} = 0$ otherwise. The C constraints in (2) guarantee that the total size of all items packed into each container does not exceed $S_{\max}$, the n constraints in (3) ensure that each item is packed into a single container only.

To avoid trivial cases we further assume

$$s_j \leq S_{\max}, \quad j = 1, \ldots, n, \tag{4}$$

$$\sum_{j=1}^{n} s_j \geq S_{\max}\,. \tag{5}$$

The first assumption (4) ensures that each item j fits into a container as otherwise it may be removed from the problem. Constraint (5) avoids the trivial case where all items fit into one of the containers.

The MCPP comprises similarities to the well known *knapsack problem* (KP) and the *bin packing problem* (BPP). The KP is equivalent to the special case of the MCPP with only a single container ($C = 1$). It belongs to the family of $\mathcal{NP}$-hard problems, meaning that it is very unlikely that we ever can devise polynomial algorithms to solve it exactly [6]. But despite the exponential worst-case solution times of all exact KP-algorithms, many large scaled problem instances of practical relevance are pseudo-polynomially solvable, i.e. the complexity is

bounded by the number of variables and the magnitude of the largest coefficient in the instance. Such KP instances may be solved to optimality in very reasonable times using branch-and-bound techniques [20, 25]. Furthermore, efficient approximation algorithms have been developed for obtaining near optimal solutions [19].

There is a more general variant of the KP, called the *multiconstraint knapsack problem* [25], which involves more than one limited resources leading to multiple resource constraints. E.g. additionally to the size, the weight could be a second constrained resource. Exact algorithms to this problem can be found in [19, 25], but their application is limited to relatively small problem instances because this KP-variant is generally $\mathcal{NP}$-hard in the strong sense.

Note that in the literature, the MCPP is sometimes also refered to as *multiple knapsack problem* [25], but we do not adopt this name here since it is also used for the multiconstraint knapsack problem [17, 18] and can therefore be misleading.

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, values of items do not play a role. The BPP in its general form is $\mathcal{NP}$-hard in the strong sense [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 [27] for a more detailed comparison of the MCPP to other combinatorial optimization problems.

The MCPP is $\mathcal{NP}$-hard in the strong sense, and thus any dynamic programming approach would result in strictly exponential time bounds [6]. Recently, Pisinger [24, 25] devised a branch-and-bound algorithm which is able to find globally optimal solutions for various problem instances involving many items but only few containers. Nevertheless, large instances with smaller n/C ratios remain usually intractable. Because of its hardness, the MCPP is addressed by heuristic methods, including *genetic algorithms* (GAs) [1, 8, 21].

This paper presents a GA that encodes candidate solutions via a technique called *weight-coding*: A solution to the 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 several other combinatorial optimization problems such as the optimum communications spanning tree problem [23], the rectilinear Steiner tree problem [13], the shortest common supersequence problem [2], the 3-satisfiability problem [4], and the traveling salesman problem [15, 16]. An overview of various weight-coded GAs is given by Julstrom [14].

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

Note that this article is based on [28], but extends it by the introduction of a new, more efficient biasing technique involving log-normally distributed weights and an empirical study regarding the selection of the biasing strength parameter γ.

2. Prior Evolutionary Approaches

Besides other heuristic approaches, several researchers have developed successful GAs for the KP and especially the more difficult multiconstraint KP, including Chu and Beasley [3], Hinterding [11, 12], Khuri et. al. [17], Olsen [22], Raidl [26], and Thiel and Voss [32]. A GA employing weight-coding is described in [29]. Leguizamon and Michalewicz [18] applied an ant system to the multiconstraint KP. A concise overview of various evolutionary approaches to multiconstraint KPs is given by Gottlieb [9]. He also points out that focusing the search to the border of the feasible region is a key-criterion for high effectivity. The feasible region contains all feasible solutions which cannot be simply improved by packing additional items without removing others. Falkenauer [5], Gent [7], and Reeves [31] presented hybrid GAs for the BPP.

In [27], it was observed that prior GA approaches to the KP and BPP can be roughly 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, 21]) 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 [27] 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 is possibly improved by trying to pack its unpacked items into a container that has not reached its maximum total size yet. In this way, only solutions lying on the boundary of the feasible region (according to [9]) are produced. The improvement is performed in a Lamarckian way, therefore, the genotype is changed accordingly. The experimental comparison in [27] indicates better performance for the OBE approach in case of fewer items and for the DE approach in case of larger problems. The local improvement leads in many cases not only to better results, but also to shorter running times because of faster convergence.

3. A Weighted Coding for the MCPP

In the weight-coded GA, a candidate solution to the multiple container packing problem is 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 in a first step in one of the following two ways.

3.1. Uniform Additive Biasing

As in most previous weight-coded approaches for other problems, we add weights to problem data, in detail to the relative item values $r_j = v_j/s_j$ (v_j: absolute item value, s_j: item size):

$$r'_j = r_j + w_j \,. \tag{6}$$

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

$$v'_j = r'_j s_j = v_j + s_j w_j \,. \tag{7}$$

During the generation of a GA's initial population and during mutation, weights w_j get uniformly distributed random values in a certain range $[-\gamma\overline{r}, \gamma\overline{r}]$ with $\overline{r} = 1/n \sum_{j=1}^{n} r_j$ being the average relative item value. The parameter γ controls how strong item weights can be biased and is therefore called *biasing strength*. Suitable values are theoretically discussed in section 3.3 and empirically investigated in section 5.1. Due to the symmetry of the range of values, the median biased problem corresponds to the original problem in a randomly generated initial GA population. Care must be taken to guarantee always positive biased item values, which means

$$\gamma \leq \frac{\min_{j=1,\ldots,n} r_j}{\overline{r}} \,. \tag{8}$$

However, our latest investigations on a weight-coded GA for the multiconstraint knapsack problem [29] revealed that the following different biasing approach might give some advantages.

```
procedure Heuristic A;
for all containers i do
  p ← 0; /* occupied size of container */
  for all unpacked items j sorted according to
        decreasing v'_j (or r'_j) do
    if p + s_j ≤ S_max then
      pack item j into container i;
      p ← p + s_j;
done;
```

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

3.2. Log-Normal Multiplicative Biasing

In this case, initial and mutated weights are randomly chosen from the distribution $(1+\gamma)^{\mathcal{N}(0,1)}$, where $\mathcal{N}(0,1)$ is the normal distribution with mean 0 and variance 1, and γ is again the biasing strength. Relative item values are now biased in a multiplicative way by

$$r'_j = r_j w_j \,, \tag{9}$$

which means for the absolute item values simply

$$v'_j = v_j w_j \,. \tag{10}$$

Because of the log-normal distribution of the weights, small changes of item values are made with higher probabilities, but arbitrarily large changes are theoretically possible. Biased item values will always be greater than 0 for any $\gamma > 0$. Again, median biased item values correspond to original item values.

The second step in decoding a chromosome is to apply a heuristic for the MCPP to the modified problem to obtain the phenotypic solution. Finally, an objective value is determined for this phenotype according to equation (1) using the original, unmodified problem data (item values v_j).

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 [10]. The pseudocode of two well suited heuristics is shown in Figs. 1 and 2 and discussed in the following sections.

3.3. 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 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 theoretically ensure that the GA is in principle capable of generating the globally optimal solution to an MCPP instance, all possible item permutations must be representable by a weight vector.

For uniform additive biasing, this means that the range $[-\gamma\overline{r}, \gamma\overline{r}]$ 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 genotypic search space of the GA remains as small as possible. For a GA using relative value (r'_j) ordering in heuristic $\mathcal{A}$, the smallest biasing strength which guarantees the reachability of all item permutations is therefore

$$\gamma^* = \frac{\max_{j=1,\ldots,n} r_j - \min_{k=1,\ldots,n} r_k}{2\overline{r}}\,. \qquad (11)$$

But note that condition (8) might not always be fulfilled in this case. To ensure always positive item values, we therefore should set

$$\gamma = \min\left(\gamma^*, \frac{\min_{j=1,\ldots,n} r_j}{\overline{r}}\right) \qquad (12)$$

and have to abandon the goal to make all possible item permutations representable in all cases.

To guarantee the representability of all item permutations when using absolute value (v'_j) ordering with uniform additive biasing, we would have to set the biasing strength to at least

$$\gamma^* = \frac{\max_{j=1,\ldots,n} \left((v_{\max} - v_j)/s_j, (v_j - v_{\min})/s_j\right)}{\overline{r}} \qquad (13)$$

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

Usually, this equation will result in a much larger biasing strength than in case of relative value ordering, and condition (8) is not fulfilled. As before, γ must then be reduced to the maximum value ensuring positive biased item values (equation (12)). Once again, reachability of all possible phenotypic solutions can then not be guaranteed.

For log-normal multiplicative biasing, γ must only be larger than 0 for theoretically being able to generate all possible item permutations. Furthermore, condition (8) can never be violated. But obviously, the choice of γ is in practice crucial to avoid too small probabilities of generating some weight vectors leading to good solutions. Therefore, section 5.1 presents empirical investigations to the selection of γ.

```
procedure Heuristic B;
p⃗ ← 0⃗; /* 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 ≤ S_max) ∧ (S_max − p_i − s_j < cs) then
      c ← i;
      cs ← S_max − p_i − s_j;
  if c > 0 then
    pack item j into container c;
    p_c ← p_c + s_j;
done;
```

Figure 2: *A more sophisticated decoding heuristic.*

3.4. 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 crucial 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. Regarding the biasing strength γ, the same theoretical considerations are therefore valid as for heuristic $\mathcal{A}$.

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

Table 1: *Average results of weight-coded GAs using uniform additive biasing and decoding heuristics $\mathcal{A}$ and $\mathcal{B}$ with absolute value ordering ($\gamma = 0.8$).*

Problem	WE$\mathcal{A}v$			WE$\mathcal{B}v$		
$n/C/S_{\max}$	*gap*	*evals*	t[sec]	*gap*	*evals*	t[sec]
30/3/100	2.74	5680	0.6	2.74	8860	1.2
30/6/100	2.32	11280	1.7	2.73	9520	1.5
30/9/100	2.90	2220	0.3	2.90	1440	0.2
30/12/100	1.05	27740	4.4	1.15	98900	16.9
50/5/100	3.89	34940	7.4	3.89	40260	9.0
50/10/100	3.38	41420	8.5	3.38	14000	3.4
50/15/100	2.16	44560	10.3	2.16	62660	16.2
50/20/100	1.05	41080	9.3	0.99	28260	8.0
200/20/100	2.45	25260	28.6	2.44	43180	49.7
200/40/100	2.00	123280	144.6	1.91	79900	110.6
200/60/100	1.67	326520	492.2	1.58	280500	459.7
200/80/100	2.14	160220	263.9	2.01	109640	213.7
30/3/200	1.67	21980	3.0	1.67	17480	2.5
30/3/300	1.45	78780	9.8	1.47	69540	10.3
30/3/400	0.33	8940	1.1	0.30	83680	12.4
50/5/200	1.16	28620	5.7	1.09	71420	15.4
50/5/300	0.80	168540	34.3	0.83	99480	21.5
50/5/400	1.34	216520	49.5	1.25	114320	25.4
200/20/200	1.42	338160	320.1	1.36	300660	353.2
200/20/300	1.91	586920	551.5	2.00	581160	670.5
200/20/400	1.37	559760	501.4	1.36	446600	505.4
Average	1.87	135830	116.6	1.87	121974	119.4

Table 2: *Average results of weight-coded GAs using uniform additive biasing and decoding heuristics $\mathcal{A}$ and $\mathcal{B}$ with relative value ordering ($\gamma = 0.2$).*

Problem	WE$\mathcal{A}r$			WE$\mathcal{B}r$		
$n/C/S_{\max}$	*gap*	*evals*	t[sec]	*gap*	*evals*	t[sec]
30/3/100	2.74	380	0.1	2.74	540	0.1
30/6/100	2.32	4280	0.7	2.32	3040	0.5
30/9/100	2.90	5620	0.9	2.90	3160	0.6
30/12/100	1.05	12280	1.9	1.05	6520	1.2
50/5/100	2.08	3760	0.9	2.08	3420	0.9
50/10/100	0.87	79840	20.7	0.87	104860	29.2
50/15/100	1.67	42740	10.4	1.67	17400	5.2
50/20/100	1.05	144860	41.5	0.94	74460	24.1
200/20/100	1.34	119980	135.6	1.34	52080	65.7
200/40/100	1.26	404800	522.5	1.18	251540	382.4
200/60/100	1.06	819100	1237.0	0.87	446520	804.8
200/80/100	1.64	810300	1346.9	1.38	267920	574.5
30/3/200	0.56	4280	0.7	0.56	5920	1.0
30/3/300	0.40	79280	11.1	0.40	71200	11.8
30/3/400	0.26	55400	9.1	0.26	67060	11.0
50/5/200	0.18	46360	11.6	0.18	55160	14.3
50/5/300	0.06	93980	24.4	0.06	111340	28.9
50/5/400	0.17	112920	28.4	0.17	91880	23.8
200/20/200	0.27	316380	342.4	0.25	274480	339.5
200/20/300	0.14	594560	583.9	0.14	371940	450.5
200/20/400	0.13	402040	467.0	0.10	338400	416.4
Average	1.06	197769	228.5	1.02	124707	151.7

directly stored as real values. Initial solutions are generated by assigning each weight a random value (either uniformly distributed or log-normally distributed). 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 weight 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.

As observed in several previous steady-state GAs for combinatorial optimization problems and closer investigated in [30], it proved to be essential to disallow phenotypic duplicates in the population in order to prevent premature convergence. 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 objective value.

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 had been evaluated without finding a new best 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 Comparisons

The publicly available MCPP test problem set of [27][1] was used to evaluate the new weight-coded GA variants. This set consists of 21 problem instances 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 $\overline{s} = 50$. 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 $[0.8, 1.2]$. Therefore, item values v_j are strongly correlated to item sizes giving more realistic but also substantially more difficult problem instances [24, 25].

In case of uniform additive biasing and relative value ordering within the heuristic, γ should be about 0.2 for the used problem instances according to equations (11) and (12). For absolute value ordering, equation (13) results in a biasing strength $\gamma^* \approx 22$ which is larger than $\min_{j=1,\ldots,n} r_j/\overline{r} \approx 0.8$ and not feasible according to condition (8). Therefore, we have to choose a significantly smaller biasing strength $\gamma = 0.8$, and reachability of all

[1] http://www.apm.tuwien.ac.at/pub/TestProblems/mcpp

Table 3: *Average results of weight-coded GAs using log-normal multiplicative biasing and decoding heuristics $\mathcal{A}$ and $\mathcal{B}$ with relative value ordering ($\gamma = 0.2$).*

Problem	WE$\mathcal{A}_\mathcal{N}$			WE$\mathcal{B}_\mathcal{N}$		
$n/C/S_{\max}$	*gap*	*evals*	t[sec]	*gap*	*evals*	t[sec]
30/3/100	2.74	300	0.1	2.74	250	0.0
30/6/100	2.32	3340	0.7	2.32	1910	0.4
30/9/100	2.90	5000	1.0	2.90	3800	0.8
30/12/100	1.05	10140	2.2	1.05	5360	1.2
50/5/100	2.08	3050	0.9	2.08	3180	1.0
50/10/100	0.87	64410	20.5	0.85	71240	22.8
50/15/100	1.67	39610	13.1	1.67	8170	2.8
50/20/100	0.96	104020	34.7	0.94	26160	9.4
200/20/100	1.34	48410	64.7	1.31	15170	20.7
200/40/100	1.24	343400	509.0	1.14	161150	260.2
200/60/100	0.96	387540	614.2	0.83	224120	411.8
200/80/100	1.57	249130	434.8	1.31	252810	542.3
30/3/200	0.56	4260	0.9	0.56	5450	1.1
30/3/300	0.40	76280	15.3	0.40	74200	14.8
30/3/400	0.25	70410	14.0	0.28	11790	2.3
50/5/200	0.18	42260	13.1	0.18	30470	9.3
50/5/300	0.06	105190	32.3	0.05	93670	29.0
50/5/400	0.18	113950	34.9	0.18	145780	45.1
200/20/200	0.22	291830	367.2	0.20	231940	304.8
200/20/300	0.11	540170	648.5	0.09	299880	390.7
200/20/400	0.09	534540	621.0	0.09	441530	573.9
Average	1.04	144630	163.9	1.01	100382	125.9

Table 4: *Gaps of solutions obtained by applying heuristics $\mathcal{A}$ and $\mathcal{B}$ with absolute/relative value ordering directly to the original, unbiased problem instances.*

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

possible phenotypic solutions cannot be guaranteed in this case.

Since 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 a 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

$$gap = (V_{\max}^{\mathrm{LP}} - V)/V_{\max}^{\mathrm{LP}} \cdot 100\,\%\,. \tag{14}$$

For each problem instance and each GA variant, 10 independent runs were performed and averaged. Table 1 shows results for weight-coded GAs using uniform additive biasing and decoding heuristics $\mathcal{A}$ and $\mathcal{B}$ with absolute values v'_j as criteria for the order of processing items (algorithms WE$\mathcal{A}v$ and WE$\mathcal{B}v$). As discussed before, the biasing strength γ was set to 0.8. The table contains gaps of best-of-run solutions and the numbers of evaluations together with the CPU times for obtaining these solutions. Table 2 presents results for the same GAs with relative value ordering during decoding (algorithms WE$\mathcal{A}r$ and WE$\mathcal{B}r$), and Table 3 shows results for GAs using log-normal multiplicative biasing and heuristics $\mathcal{A}$ respectively $\mathcal{B}$ with relative value item ordering (algorithms WE$\mathcal{A}_\mathcal{N}$ and WE$\mathcal{B}_\mathcal{N}$). For WE$\mathcal{A}r$, WE$\mathcal{B}r$, WE$\mathcal{A}_\mathcal{N}$, and WE$\mathcal{B}_\mathcal{N}$, γ was set to 0.2. Average gaps of all six test series are also depicted graphically in Fig. 3. Figure 4 shows the needed numbers of evaluations to find the finally best solutions.

In nearly all test cases, the GAs with heuristics based on relative value ordering outperform the variants using absolute value ordering. Especially for instances with many items ($n = 50, 200$) and relatively few containers, relative value ordering causes substantially better solutions. No significant quality and CPU time differences could be observed between WE$\mathcal{A}v$ and WE$\mathcal{B}v$. Concerning CPU time, WE$\mathcal{A}r$, WE$\mathcal{B}r$, WE$\mathcal{A}_\mathcal{N}$, and WE$\mathcal{B}_\mathcal{N}$ 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 nearly all cases, WE$\mathcal{B}r$ and WE$\mathcal{B}_\mathcal{N}$ found slightly better solutions or—for the simpler problem instances—the same solutions as WE$\mathcal{A}r$ and WE$\mathcal{A}_\mathcal{N}$. On average, but especially for large problems ($n = 200$), WE$\mathcal{B}r$ and WE$\mathcal{B}_\mathcal{N}$ are significantly faster and need fewer evaluations than do WE$\mathcal{A}r$ and WE$\mathcal{A}_\mathcal{N}$. Differences in the obtained gaps between WE$\mathcal{A}r$ and WE$\mathcal{A}_\mathcal{N}$, respectively WE$\mathcal{B}r$ and WE$\mathcal{B}_\mathcal{N}$, are only small, but the variants using log-normal multiplicative biasing (WE$\mathcal{A}_\mathcal{N}$, WE$\mathcal{B}_\mathcal{N}$) produce in nearly all cases either equally good or slightly better solutions. Regarding the number of needed eval-

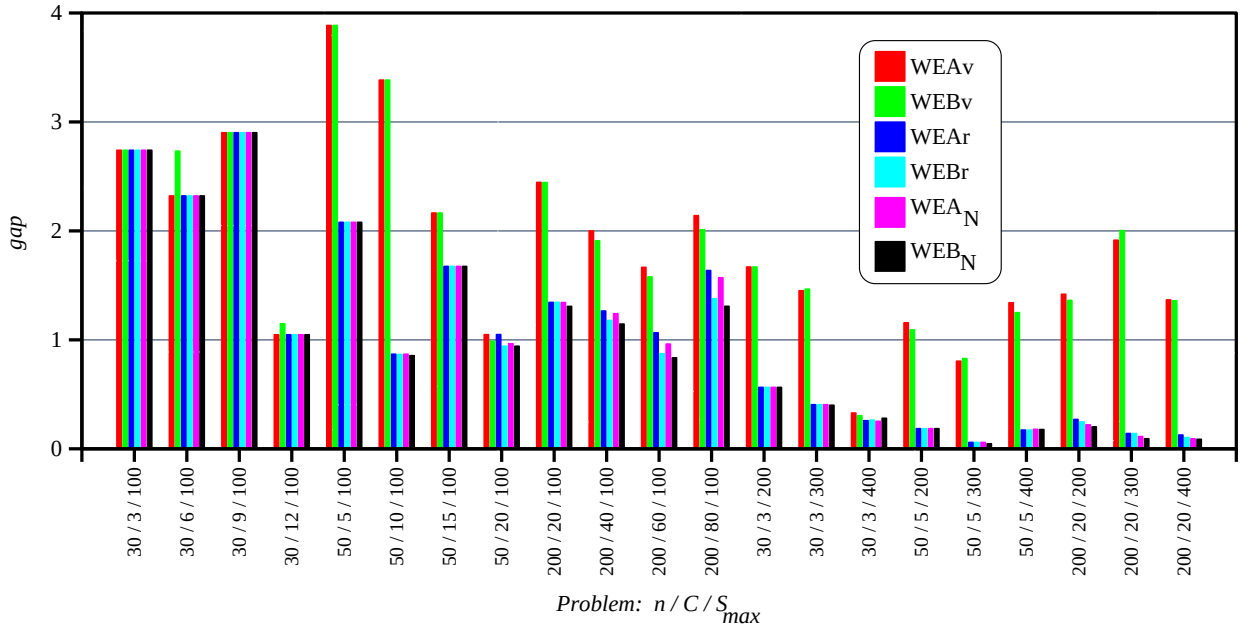

Figure 3: *Average gaps of final solutions obtained by the weight-coded GAs.*

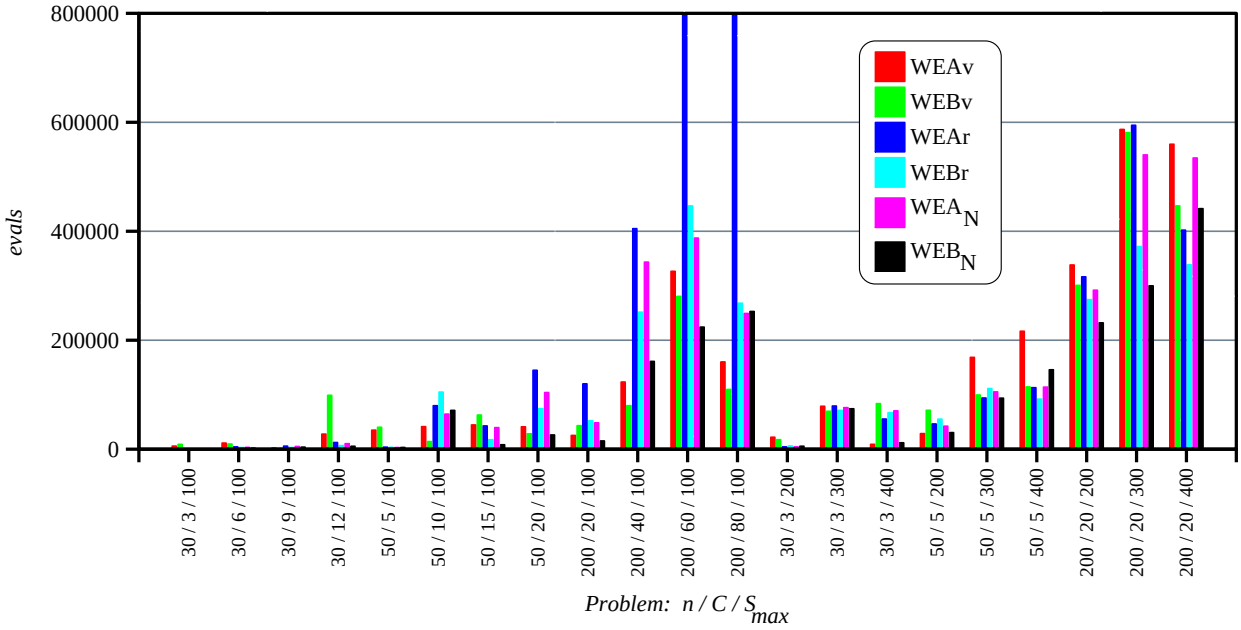

Figure 4: *Average numbers of evaluations until final solutions were found by the weight-coded GAs.*

uations and the corresponding CPU times, WE$\mathcal{A}_\mathcal{N}$ and WE$\mathcal{B}_\mathcal{N}$ are in average clearly faster than their counterparts WE$\mathcal{A}r$ and WE$\mathcal{B}r$.

Table 4 shows the gaps of solutions obtained by applying all four heuristics directly (without any GA) to the unbiased, original problems. In general, the heuristics using relative value ordering find significantly better solutions than those using absolute value ordering. But note that the solutions directly obtained by any heuristic are poor compared to final solutions of the various GAs.

Table 5 contains results of the GAs Raidl and Kodydek presented in [27]. 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$\mathcal{A}r$, WE$\mathcal{B}r$, WE$\mathcal{A}_\mathcal{N}$, WE$\mathcal{B}_\mathcal{N}$) find so-

Table 5: *Average gaps of final solutions from direct encoded and order-based GAs with optional local improvement (from [27]).*

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

lutions with smaller gaps than those prior approaches. Furthermore, significantly fewer evaluations are 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$\mathcal{B}r$ and WE$\mathcal{B}_{\mathcal{N}}$, 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 structures.

5.1. Influence of the biasing strength γ

Table 6 and Fig. 5 show how the biasing strength γ affects the quality of the weight-coded GAs' final solutions. Each value was determined by averaging the results of 10 runs for the problem instance with $n = 200$ items, $C = 40$ containers, and a container size of $S_{\max} = 100$. Note that for the GAs using uniform additive biasing, γ cannot be larger than 0.8 to ensure always positive biased item values v'_j (condition (8)).

The influence of γ is basically very similar in all GA variants. If γ is smaller than a certain lower bound, final

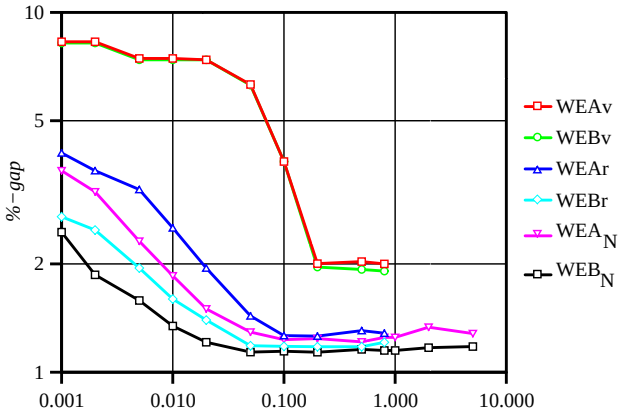

Figure 5: *Influence of different values for γ on the final gaps (problem instance $n = 200$ / $C = 40$ / $S_{\max} = 100$).*

solutions are significantly worse. This bound is about 0.2 for WE$\mathcal{A}v$ and WE$\mathcal{B}v$, 0.1 for WE$\mathcal{A}r$ and WE$\mathcal{A}_{\mathcal{N}}$, and 0.05 for WE$\mathcal{B}r$ and WE$\mathcal{B}_{\mathcal{N}}$. Obviously, for values of γ less than these limits, the weighted coding is in practice not able to represent solutions significantly different from those the decoding heuristics would find on their own with enough probability. With increasing biasing strength, the GAs can examine more of the problem's search space. As γ increases above 0.2 (respectively 0.1 or 0.05), the solutions' qualities do not change much. We can observe only a slight increase of the gaps for WE$\mathcal{A}_{\mathcal{N}}$ and WE$\mathcal{B}_{\mathcal{N}}$ above $\gamma = 1$.

The number of evaluations, which is also shown in Table 6, depends strongly on γ. In general, more evaluations are needed with increasing γ, since more different solutions are produced causing larger diversity.

6. Conclusions and Future Work

This paper has presented a new coding of solutions to the multiple container packing problem using vectors of weights. Two techniques for biasing the original problem with the weights were proposed. For obtaining the actual solution out of a biased problem, two substantially different heuristics were described, and each of them can use either absolute or relative value ordering. Weight-coding eliminates the necessity of an explicit repair algorithm, a penalization of infeasible solutions, or special recombination and mutation operators. The different variants of weight-coded GAs were tested using 21 problem instances. Results primarily indicate better performance for relative value ordering. Heuristic $\mathcal{B}$, which fills containers in parallel, works in many cases slightly better than heuristic $\mathcal{A}$. Furthermore, log-normal multiplicative biasing leads consequently to slightly better solutions than uniform additive biasing and usually causes shorter running times because less evaluations are needed for convergence. The selection of a suitable value for the biasing strength γ turned out to be not so critical as long as γ is large enough to enable good solutions to be representable with enough probability.

Table 6: *Influence of different values for the biasing strength γ (problem instance: $n = 200$ / $C = 40$ / $S_{\max} = 100$).*

γ	WE$\mathcal{A}v$		WE$\mathcal{B}v$		WE$\mathcal{A}r$		WE$\mathcal{B}r$		WE$\mathcal{A}_{\mathcal{N}}$		WE$\mathcal{B}_{\mathcal{N}}$	
	gap	*evals*	*gap*	*evals*	*gap*	*evals*	*gap*	*evals*	*gap*	*evals*	*gap*	*evals*
0.001	8.29	100	8.22	100	4.07	5160	2.71	100	3.63	57540	2.45	22700
0.002	8.29	100	8.22	100	3.63	70660	2.48	400	3.17	146960	1.87	157540
0.005	7.45	100	7.38	100	3.22	86460	1.95	53420	2.31	127760	1.58	117710
0.01	7.45	100	7.38	100	2.52	84740	1.60	146120	1.85	141460	1.34	302800
0.02	7.38	360	7.38	100	1.95	150600	1.40	246020	1.50	276220	1.21	194520
0.05	6.30	17680	6.28	11420	1.43	167060	1.18	148000	1.29	233220	1.14	131190
0.1	3.85	26940	3.85	22340	1.26	320880	1.18	155580	1.23	344170	1.14	160930
0.2	2.00	155640	1.96	129280	1.26	404800	1.18	251540	1.24	343400	1.14	161150
0.5	2.03	264980	1.93	149520	1.30	467400	1.18	286560	1.21	215620	1.16	194190
0.8	2.00	123280	1.91	79900	1.28	475620	1.21	284520	1.25	377400	1.15	324050
1	–	–	–	–	–	–	–	–	1.26	350130	1.16	298340
2	–	–	–	–	–	–	–	–	1.33	337710	1.17	330770
5	–	–	–	–	–	–	–	–	1.28	345900	1.18	378520
10	–	–	–	–	–	–	–	–	1.33	390470	1.16	329170

All GAs using relative value ordering decisively outperform the more traditional direct-encoded and order-based GAs with optional local improvement from [27] regarding the quality of final solutions and average numbers of evaluations needed for convergence.

Future work should 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 combinatorial optimization problems.

References

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

[2] Branke J., Middendorf M.: Searching for Shortest Common Supersequences by Means of a Heuristic-Based Genetic Algorithm, in *Proc. of the 2nd Nordic Workshop on Genetic Algorithms and their Applications*, University of Vaasa, Vaasa, Finland, 1996, pp. 105–113.

[3] Chu P. C., Beasley J. E.: *A Genetic Algorithm for the Multidimensional Knapsack Problem*, Journal of Heuristics 4, 1998, pp. 63–86.

[4] 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.

[5] Falkenauer E.: A Hybrid Grouping Genetic Algorithm for Bin Packing, *Journal of Heuristics* 2(1), 1996, pp. 5–30.

[6] Garey M., Johnson D.: *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness*, Freeman, San Francisco, 1979.

[7] Gent I. P.: Heuristic Solution of Open Bin Packing Problems, to appear in *Journal of Heuristics*, 1999.

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

[9] Gottlieb J.: On the Effectivity of Evolutionary Algorithms for the Multidimensional Knapsack Problem, in *Proc. of Artificial Evolution 1999*, Dunkerque, France, to appear in Springer LNCS, 1999.

[10] Gottlieb J., Raidl G. R.: Characterizing Locality in Decoder-Based EAs for the Multidimensional Knapsack Problem, in *Proc. of Artificial Evolution 1999*, Dunkerque, France, to appear in Springer LNCS, 1999.

[11] 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.

[12] Hinterding R.: Representation, Constraint Satisfaction and the Knapsack Problem, in *Proc. of the 1999 IEEE Congress on Evolutionary Computation*, Washington D.C., 1999, pp. 1286–1292.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] Leguizamon G., Michalewicz Z.: A New Version of Ant System for Subset Problems, in *Proc. of the 1999 IEEE Congress on Evolutionary Computation*, Washington DC, 1999, pp. 596-603.

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

[20] Martello S., Pisinger D., Toth P.: New Trends in Exact Algorithms for the 0–1 Knapsack Problem, to appear in *European Journal of Operational Research*, 1999.

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

[22] 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.

[23] 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.

[24] Pisinger D.: The Multiple Loading Problem, in *Proc. of NOAS'95*, University of Reykjavik, Iceland, 1995, pp. 18–19.

[25] Pisinger D.: *Algorithms for Knapsack Problems*, PhD thesis, University of Copenhagen, Dept. of Computer Science, Denmark, 1995.

[26] 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.

[27] 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.

[28] Raidl G. R.: A Weight-Coded Genetic Algorithm for the Multiple Container Packing Problem, in *Proc. of the 14th ACM Symposium on Applied Computing*, San Antonio, TX, 1999, pp. 291–296.

[29] Raidl G. R.: Weight-Codings in a Genetic Algorithm for the Multiconstraint Knapsack Problem, in *Proc. of the 1999 IEEE Congress on Evolutionary Computation*, Washington DC, 1999, pp. 596-603.

[30] Raidl G. R., Gottlieb J.: On the Importance of Phenotypic Duplicate Elimination in Decoder-Based Evolutionary Algorithms, in *Late-Breaking Papers Proc. of the 1999 Genetic and Evolutionary Computation Conference*, Orlando, FL, 1999, pp. 204–211.

[31] Reeves C. R.: Hybrid Genetic Algorithms for Bin-Packing and Related Problems, *Annals of OR* 63, 1996, 371–396.

[32] Thiel J., Voss S.: Some Experiences on Solving Multiconstraint Zero-One Knapsack Problems with Genetic Algorithms, *INFOR* 32, 1994, pp. 226–242.