Available online at www.sciencedirect.com
ScienceDirect
Progress in Natural Science 18 (2008) 337–344
Progress in Natural Science

Special report

An effective evolutionary algorithm for the multiple container packing problem

Sang-Moon Soak [a], Sang-Wook Lee [b], Gi-Tae Yeo [c,*], Moon-Gu Jeon [b]

[a] *Information Systems Examination Team, Korean Intellectual Property Office (KIPO), Gov. Complex Daejeon Building, 4 920 Dunsandong, Seogu, Daejeon, Republic of Korea*
[b] *School of Information and Mechatronics, Gwangju Institute of Science and Technology, 261 Cheomdan-gwagiro, Buk-gu, Gwanju, Republic of Korea*
[c] *College of Humanities and Social Sciences, Woosuk University 490 Hujung-Ri, Samrye-Eup, Wanju-Gun, Jeollabuk-Do, Republic of Korea*

Received 29 October 2007; received in revised form 13 November 2007; accepted 13 November 2007

Abstract

This paper focuses on a new optimization problem, which is called "The Multiple Container Packing Problem (MCPP)" and proposes a new evolutionary approach for it. The proposed evolutionary approach uses "Adaptive Link Adjustment Evolutionary Algorithm (ALA-EA)" as a basic framework and it incorporates a heuristic local improvement approach into ALA-EA. The first step of the local search algorithm is to raise empty space through the exchange among the packed items and then to improve the fitness value through packing unpacked items into the raised empty space. The second step is to exchange the packed items and the unpacked items one another toward improving the fitness value. The proposed algorithm is compared to the previous evolutionary approaches at the benchmark instances (with the same container capacity) and the modified benchmark instances (with different container capacity) and that the algorithm is proved to be superior to the previous evolutionary approaches in the solution quality.

Keywords: ALA-EA; Evolutionary algorithm; Heuristic; The multiple container packing problem

1. Introduction

In this paper, we deal with the problem which was firstly introduced by Raidl and Kodydek [1] as the multiple container packing problem (MCPP). Before Raidl and Kodydek, Martello and Toth [2] and Pisinger [3] had dealt with the same problem, but, they called it as the multiple knapsack problem (MKP).

The difference between these previous researches is that Raidl and Kodydek [1] dealt with the problem that the capacities of containers are all the same, while Martello and Toth [2] and Pisinger [3] dealt with the problem that the capacities are all different. Actually, the problems with all different container capacities seem to be much more difficult than the problem with all the same container capacity. Thus, here, we deal with both types of problems.

As Raidl [4] mentioned, MCPP comprises similarities to the knapsack problem and the bin packing problem. But, MCPP has much more difficulty in the context that it must solve two dependent parts simultaneously: (a) select items for packing, and (b) distribute chosen items over available containers. Thus, MCPP also belongs to the family of NP-hard problems, meaning that it is ever unlikely that we ever can devise polynomial algorithms to solve it exactly.

Therefore, to use an exact algorithm like a branch and bound method for solving this problem may require much computation time. Nevertheless, the branch and bound methods, which were devised by Martello and Toth [2] and Pisinger [3], had shown very good performance. Especially, the MULKNAP algorithm derived by Pisinger out-

* Corresponding author. Tel.: +82 63 290 1420; fax: +82 63 290 9312.
E-mail address: ktyeo@woosuk.ac.kr (G.-T. Yeo).

1002-0071/$ - see front matter © 2007 National Natural Science Foundation of China and Chinese Academy of Sciences. Published by Elsevier Limited and Science in China Press. All rights reserved.
doi:10.1016/j.pnsc.2007.11.007

performed the MTM algorithm derived by Martello and Toth. But, in both researches, they had dealt with only the instances that many items and small containers exist. And in Pisinger's research, the MULKNAP found optimum solutions in almost all instances. That is, the MULKNAP algorithm found the optimum solutions in the cases with relatively high n/C ratio (where n is the number of items and C is the number of containers) but at the case that the ratio was lower, it never found optimum solutions. More precisely, Pisinger mentioned in his papers [3] as follows: "A maximum amount of 1 h was given to each algorithm for solving the 10 instances in each class.... Small data instances are tested with $m = 5$ knapsacks, while large instances have $m = 10$, as none of the algorithms are able to solve problems with small values of n/m." Even though his branch-and-bound algorithm is able to find globally optimal solutions for various problems, the performance is very limited to problems involving many items but only few containers.

Therefore, at present, to use a heuristic method must be a more suitable choice in relatively low n/m ratio instances, and for the problem with many containers because the problem with small items and many containers can be solved very easily.

Nevertheless, only three heuristic methods have been proposed for MCPP [1,4,5]. These algorithms have been applied to the problems (low n/C ratio) that the previous branch and bound cannot solve.

The main goal of this paper is to propose a new evolutionary algorithm and to show the performance of the proposed algorithm through the comparison to the previous heuristic algorithms at the problems with low n/C ratio.

2. The multiple container packing problem

The multiple container packing problem (MCPP) is a kind of a combinatorial optimization problem which is able to be applied to several real problems like a truck loading problem, an airplane luggage packing problem and a liquid packing problem.

MCPP can be formulated as follows:

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

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

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

$$x_{i,j} \in \{0, 1\}, \tag{4}$$

where $w_j > 0$, $v_j > 0$, $c_j > 0$

Here, w_j and v_j indicate, respectively, weight and value of items and c_i indicates the maximum weight (the container capacity) of items to be loaded into container i. If an item j is to be packed into a 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 and the C constraints in (3) guarantee that the total size of all items packed into each container does not exceed c_i.

3. Previous evolutionary approaches

Until now, just three evolutionary algorithms have been proposed for solving the MCPP: the evolutionary algorithms using the order-based encoding and the direct encoding derived by Raidl and Kodydek [1] and the evolutionary algorithm using the weight encoding derived by Raidl [4,5].

In case of EA using the order-based encoding, each solution is represented as a permutation of items and according to the order of a permutation, all items are packed into containers. If an item can be packed in a container, it is assigned to the container and the next item is considered. If an item cannot be packed in a container, the item is stored separately for a local search mechanism and then the other item is considered. This process is performed until all items are considered. Once all items are considered, a packing plan is completed. Thus, using this encoding, EA can always generate a feasible solution. But, because the encoding has to keep a form of a permutation, EA cannot use general evolutionary operators like the one-point or multi-point crossover and the uniform crossover.

In case of EA using the direct encoding, each solution is represented as a kind of an array and the index of an array (locus) corresponds to each item. The gene values indicate each container numbers. That is, if the value of gene located in the second index is 1, it can be interpreted as that item 2 is packed into the container 1. But, this decoding method can generate infeasible solutions, which violate the constraint on the maximum capacity c_i of each container. Thus, the EA requires a sort of repair process. But, the repair process is very simple: if an item violates the maximum capacity of a container, the item is stored for a local search like the order-based encoding.

After the above two steps the decoding processes are finished, which improve the fitness value using a local improvement algorithm. All previously unassigned items j are processed in a random order, and each container is checked in random order if enough space is available to pack item j. If possible, the item is assigned to the container and the algorithm keeps proceeding with the next unassigned item in the same way. For more details, see Ref. [1].

The EA uses the weight encoding which was derived by Raidl [4], and he also used weight values (wt_j) generated at random in a specific range [5]. The index of array corresponds to each item and the value indicates the weight values generated at random. To decode such a chromosome, a modified problem is generated by adding these weights to the relative item values $r_j = v_j/w_j$ (v_j: absolute item value, w_j: item size).

$$r'_j = r_j + wt_j. \tag{5}$$

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

$$v'_j = r'_j w_j = v_j + w_j w t_j. \quad (6)$$

And, Raidl proposed two decoding heuristics: heuristic A and heuristic B. First, all items are sorted in ascending order according to the above two values. And then, the heuristic A assigns all items to a container according to the sorting order. If an item cannot be packed into a container, then the other container is considered. One container after the other is filled by going through all unpacked items and packing all items not violating the capacity constraint into the current container. On the other hand, in case of heuristic B, 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. These algorithms have four variants according to the heuristic methods (A or B) and the sorting measures (the relative item values or the new absolute item values). For more details, refer to [4], [5].

4. New evolutionary algorithm for MCPP

4.1. Adaptive link adjustment-EA: ALA-EA

ALA-EA was derived by Soak [6,7] and it showed very good performance at the fixed charge transportation problem and the quadratic minimum spanning-tree problem.

In general, at ALA-EA, all possible edges in network problems correspond to each gene's locus and all gene values are initially set to '0' (see Fig. 1). The gene values indicate a measure to distinguish between good edges or bad edges for the problem considered during the entire evolutionary process. And, it first sorts all genes according to their gene value in ascending order and then each edge is considered one by one from the gene (edge) with the lowest gene value to the gene (edge) with the highest gene value. Because all gene values are all the same at generation 1, a random sorting mechanism is applied. After all edges are considered according to the sorted order, ALA-EA generates a network (phenotype) related to the problem considered. And then, ALA-EA applies α and β processes to the solutions according to the fitness value. If the fitness value is better than the current best solution, then ALA-EA applies α process to genes (edges) included in the phenotype. Otherwise, β process is applied to all genes in each solution. The β process is controlled by the θ value, which controls how many genes (edges) are selected for β process. While α process is a kind of 'penalty', and β process is a kind of 'reward'. The reason why 'penalty' is applied to the best current solution is that if ALA-EA gives 'reward' to the best current solution, it takes no effect to the best current solution and moreover the population will be filled by the best current solution in a few generations. Finally, the evolutionary process will fall into a local optima. On the other hand, if ALA-EA gives 'penalty' to the best current solution, it results in the competition between the edges, which are included in the phenotype, and the edges which are not included in the phenotype.

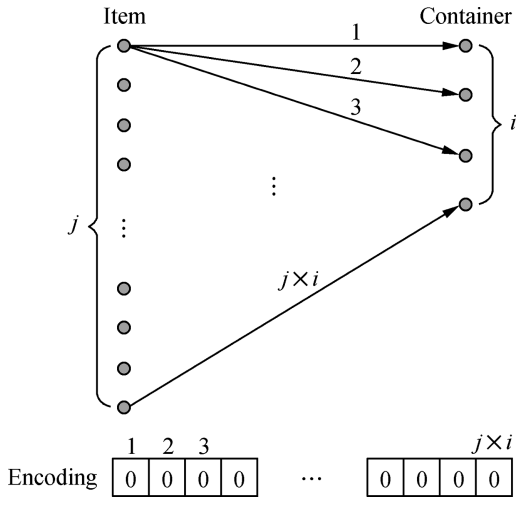

Fig. 1. Encoding method of ALA-EA. Here, each possible edge is corresponding to each locus of encoding.

4.2. Evolutionary operators

4.2.1. Selection strategy

For sampling process during evolution, we use the real world tournament (RWTS) selection [8]. RWTS had been tested for several optimization problems and found to be slightly superior to others overall. Recently, Lee et al. [9] showed mathematically and empirically that RWTS has very outstanding features in the points of the sampling accuracy and the population diversity preservation.

We here briefly describe RWTS as follows: To generate an intermediate population, the candidate solutions in population t are paired according to their population index number, and the fittest in each pair (breaking ties randomly) is placed into the intermediate population. These "winning" candidate solutions are called "level 1" winners. Next, the level 1 winners are paired (if the number of winners is an odd number, one candidate among the other solutions in the population is randomly selected), and the fittest in each of these pairs (breaking ties randomly) are termed the "level 2" winners, and copied into the intermediate population (i.e. additional copies are included in these candidates). This process continues until the intermediate population has the same size as the original population. When, at some level k and beyond, there is only 'winner' to use, this is always paired with the index order of the population. When the intermediate population is full, genetic operators are applied to it to produce the next generation.

4.2.2. Crossover and mutation operator

Actually, ALA-EA does not have any limitation in using evolutionary operators. But, for finding more efficient operators for MCPP, we had tested several operators.

In preliminary experiments, several crossovers (1-point, 2-point and uniform crossover) and mutation operators (inversion, swap and random perturbation mutation) were tested. Among them, the uniform crossover with the probability of 100% and the swap mutation with the probability of 100% gave better results. So we use them for the new algorithm in this paper.

4.2.3. New evolutionary algorithm for the MCPP

Because ALA-EA uses all possible edges for an encoding at network problems, for applying it to MCPP we should be able to encode the problem to satisfy this condition. Fig. 1 shows the encoding method of ALA-EA for MCPP. In there, all items are represented as they can be assigned to each container and all gene values are set to '0' at the initialization process. Therefore, the edge numbers correspond to the decision variable $x_{i,j}$.

For the decoding, ALA-EA first selects the edge e with the lowest gene value and checks whether the edge e can be included in the assignment plan. If possible, then the decision variable $x_{i,j}$ of the edge e is set to '1', otherwise '0'. And then the edge considered is eliminated from set A, which indicates the set of all possible edges. This process is repeated until set A empties. Finally, a feasible solution is generated through this process.

4.2.4. Heuristic local search algorithms

In this subsection, we propose two heuristic local search methods for improving the solutions found by ALA-EA.

4.2.4.1. Empty space raising heuristic (ESRH). The main idea of ESRH is to raise the empty space of a specific container through the movement of packed items and to pack the current unpacked items into the raised empty space. First, we check whether empty space of a specific container can be increased by moving an item in a container a into a container b. Here, the important point is that if an item in the container a can be moved to the container b, other items in the container a are checked whether they can be moved into other containers. Once an item in the container b is moved to other containers, the container b is not considered for putting items during this process any more.

Through this process, the empty space of a specific container should be increased as much as possible. This results in higher possibility of that an unpacked item can be packed into the empty space. After that, all unpacked items are again checked whether they can pack into containers. If an unpacked item can be packed into the increased empty space of a container, it is loaded into the container. The ESRH algorithm is shown below.

```
for all containers C do
    S ← ∀C;
    if any item in C_i does not move then
        for all I_k in C_i do
            if I_k in C_i can be moved into C_t(C_t ∈ S) then
                C_i ← C_i − {I_k}, C_t ← C_t ∪ {I_k};
                S ← S − C_i;
for all unpacked items I_k do
    if I_k can be packed into C_t then
        C_t ← C_t ∪ {I_k};
```

I_k: Item k, C_i: The set of items in container i

4.2.4.2. Exchange heuristic (EH). We propose another heuristic algorithm, the exchange heuristic. This is very simple and straightforward. The main idea of EH is to check whether the packed items and the unpacked items are exchanged with improving of the fitness value. If the exchange can improve the fitness value, then two items are exchanged. This process is performed until all unpacked items are considered. The EH algorithm is shown below.

```
for all unpacked items I_k do
  for all I_t in C_i do
   if exchange (I_k, I_t) satisfies constraints (2) and (3),
        and improves fitness function (1) then
   C_i ← C_i − {I_t}, C_i ← C_i ∪ {I_k};
```

I_k: Item k, C_i: The set of items in container i

4.2.4.3. A 2-step heuristic local search algorithm. We introduce a 2-step heuristic local search algorithm for improving the feasible solutions generated by decoding process of ALA-EA. This heuristic algorithm combines the previous two local search methods: ESRH and EH.

Fig. 2 depicts the 2-step heuristic local search algorithm. Here, assume that after the decoding process of ALA-EA, the empty space of containers A and B remains 10 and 5, respectively, and an item of $w_j = 10$ is packed in the container B, and there is unpacked item with $w_k = 15$. Then, the item of $w_j = 10$ can be moved into the container A and finally the container A is fully packed and the container B has empty space 15. Now, we can pack the item of $w_k = 15$ into the container B. Next, the unpacked items and the packed items are exchanged at the second heuristic (EH).

The proposed ALA-EA incorporating with 2-step heuristic algorithm is shown below. Here, we used the same ALA-EA that was used for the fixed charge transportation problem and the constrained minimum spanning-tree problem [6,7]. That is, α and β are set to 1, respectively, and θ is set to 90%.

1: Initialization

Initialize all gene values $g_{a,b} = 0$, $C_t \leftarrow \phi, T \leftarrow \phi$;

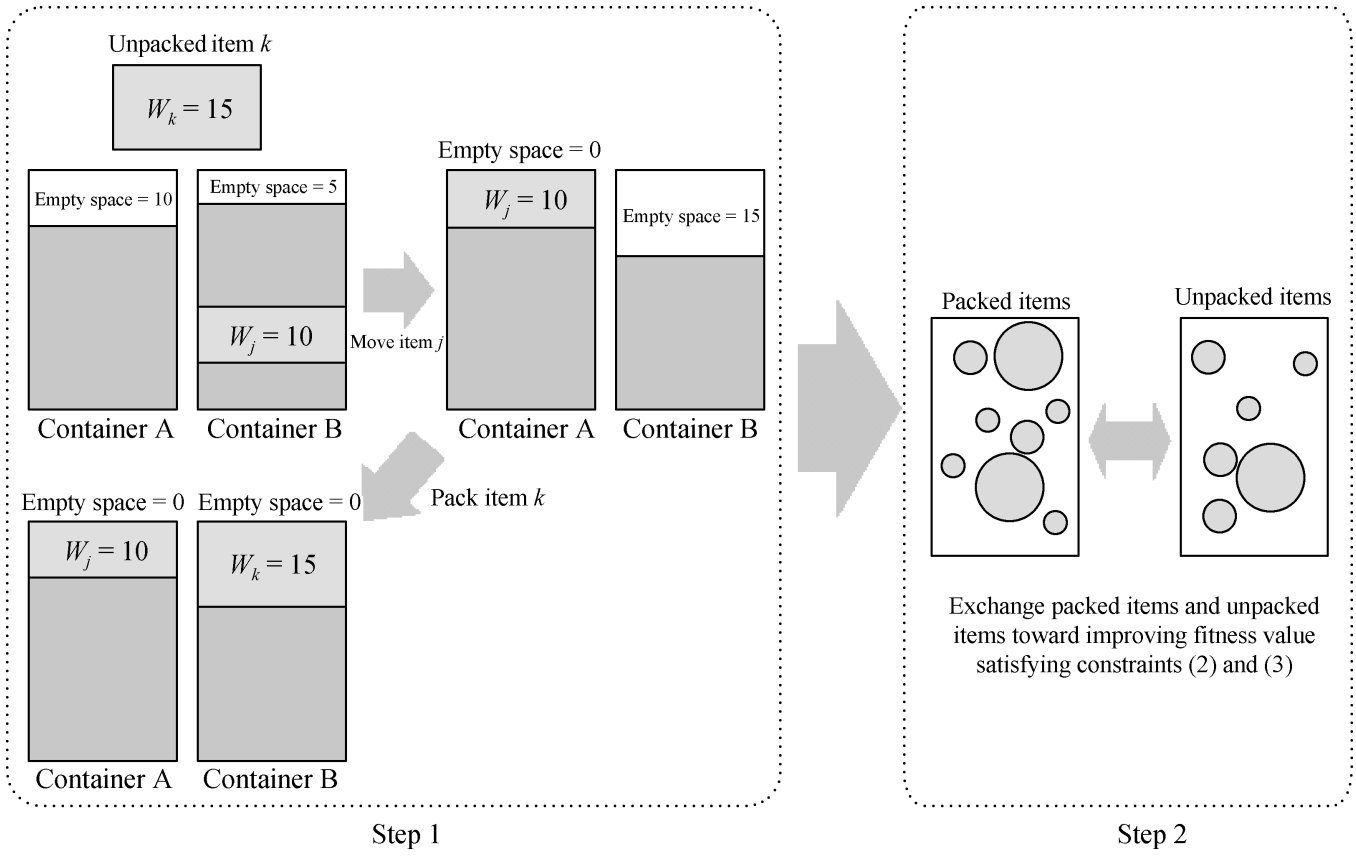

Fig. 2. The 2-step heuristic local search algorithm.

2: Packing plan (decoding) and evaluation
$T \leftarrow \phi$, $A \leftarrow E$;/*E: all possible edges's set */
for $a = 1$ to *pop_size* **do**
2.1: Packing plan (decoding)
while $A \neq \{\phi\}$ **do**
Select the gene (edge) b with the lowest gene value;
$A \leftarrow A - \{b\}$;
if the edge b satisfies constraints (2) and (3) **then**
$C_t \leftarrow C_t \cup \{I_a\}$,$T \leftarrow T \cup \{b\}$; /* edge $b = (I_a, C_t)$ */
2.2: 2-step heuristic local search
(Step 1): the empty space raising heuristic
(Step 2): the exchange heuristic
2.3: Evaluation
if *best* $> f_i$ **then**/* α process */
best $\leftarrow f_i$;
for $j = 1$ to L **do**
if $j \in T$ **then**
$g_{i,j} \leftarrow g_{i,j} + \alpha$;
else /* β process */
for $j = 1$ to L **do**
$r = random$ [0,100];
if $((r < \theta)$ && $(j \in T))$ **then**
$g_{i,j} \leftarrow g_{i,j} - \beta$;
3: Evolutionary operators
4: Termination condition check

L: The total number of edges, I_k: Item k, C_i: Container i
$g_{i,j}$: jth gene in ith solution

5. Experimental comparison

5.1. The problem with the same container capacity

We compare the proposed algorithm to the previous evolutionary approaches, which have been known to give very good performance at MCPP. For fair comparison, we applied the proposed algorithm to the benchmark instances [10] and compared the found results with the results published in the previous literature [1,4]. In case of EA using the weight coding, WE*Br* showed better performance than the other variants. Thus, we used the results of WE*Br* for comparison. In this experiment, we adopted the same termination condition which was used in the previous researches: each run terminated when 200,000 solutions have been evaluated without finding a new best solution. The only difference is that we used the normal replacement strategy (offsprings replace their parents) instead of the phenotypic duplicate elimination replacement strategy (only offsprings different from all others in the population replace the worst solutions). Therefore, in our experiments, the proposed algorithm had been tested under much a tighter condition and we cannot expect an advantage of the duplicate elimination replacement strategy for the proposed algorithm.

All experiments are performed on a personal computer with Intel Core2 1.83 GHz CPU and 1 Gbyte RAM. Table 1 shows the experimental condition for the implemented algorithms.

Table 1
Experimental condition

	Weight coding EA	ALA-EA
Population size	100	
Initialization	Random	
Selection	Tournament selection with replacement	RWTS
Crossover	Uniform (60%)	Uniform (100%)
Mutation	Swap (a probability of $3/n$)	Swap (100%)

Since optimal solution values for these benchmark instances are not known, we use the percentage difference (*gap*) between the total value V of packed items in each solution and the optimal value V_{max}^{LP} of the LP-relaxed problem for measuring the quality of a final solution. 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 $C \times c_i$ is reached. The last item is counted proportionately. Knowing the LP optimum, the percentage difference is

$$gap = (V_{max}^{LP} - V)/V_{max}^{LP} \times 100\%. \quad (7)$$

For each problem instance and the proposed algorithm, 15 independent runs were performed and averaged.

Table 2 shows the experimental results. Here, low values indicate better results. Comparing the experimental results, ALA-EA with 2-step shows a better performance than DE, DEI, OBE and OBEI but worse performance than WE*Br*. But, the difference is very small. The maximum difference is 1.32 at 200/60/100 instance and the difference does not exceed 1 at most other instances. When comparing the standard deviation, the proposed algorithm seems to give very stable results at all instances. Here, we want to emphasize that ALA-EA with 2-step local improvement did not use 'the duplicate elimination replacement strategy'. Thus, when considering the performance gap between WE*Br* and ALA-EA with 2-step local improvement, the results should be interpreted very carefully.

5.2. The problem with different container capacities

In this experiment, we focused on the comparison between ALA-EA with 2-step and WE*Br*. We implemented WE*Br* following Raidl's work [4]. The only difference is not to incorporate with the duplicate elimination replacement strategy. The reason is that if it can be of help to the WE*Br*'s performance, we can expect similar results also at ALA-EA with 2-step. Thus, for reducing the computation time and comparing the basic algorithms' performance, we adopted the normal replacement strategy for two algorithms. We also used the same test data as the previous instances but each container's capacity was generated randomly in the range of $[c_i - 25, c_i + 25]$. Therefore,

Table 2
Results of the problem with the same container capacity

Problems	DE[1]	DEI[1]	OBE[1]	OBEI[1]	WEBr[4]	ALA-EA with 2-Step		
$n/C/c_i$	(Average)	(Average)	(Average)	(Average)	(Average)	(Average)	σ	CPU (s)
30/3/100	2.74	2.74	3.16	2.74	2.74	2.74	0.00	11.76
30/6/100	2.69	2.45	2.82	2.32	2.32	2.48	0.20	37.13
30/9/100	3.31	3.01	3.25	2.90	2.90	2.90	<0.00	43.92
30/12/100	2.48	1.58	1.42	1.05	1.05	1.22	0.18	73.68
50/5/100	2.6	2.58	2.96	2.28	2.08	2.24	0.15	36.47
50/10/100	1.43	1.10	1.73	1.58	0.87	1.12	0.12	106.54
50/15/100	2.53	1.99	2.35	2.00	1.67	1.87	0.16	148.94
50/20/100	2.58	1.5	2.03	1.67	0.94	1.46	0.22	291.80
200/20/100	1.91	1.65	2.84	2.57	1.34	1.64	0.08	1106.54
200/40/100	1.94	1.64	2.52	2.49	1.18	1.64	0.20	5744.78
200/60/100	1.99	1.48	2.10	2.28	0.87	2.19	0.30	12795.14
200/80/100	2.89	2.09	2.36	2.62	1.38	2.56	0.51	16495.39
30/3/200	0.68	0.66	0.96	0.68	0.56	0.58	0.04	13.97
30/3/300	0.50	0.47	0.53	0.44	0.4	0.44	0.01	16.53
30/3/400	0.39	0.37	0.45	0.33	0.26	0.31	0.05	13.75
50/5/200	0.42	0.35	0.88	0.60	0.18	0.4	0.10	53.77
50/5/300	0.51	0.30	0.48	0.70	0.06	0.34	0.07	53.70
50/5/400	0.40	0.39	0.53	0.53	0.17	0.21	0.04	59.85
200/20/200	0.71	0.51	1.26	2.49	0.25	0.50	0.07	2486.59
200/20/300	0.45	0.27	0.76	1.64	0.14	0.29	0.06	3019.12
200/20/400	0.32	0.25	0.41	0.64	0.1	0.27	0.08	2120.83
Total Average	1.59	1.30	1.70	1.64	1.02	1.30		

DE and DEI: Direct encoding with and without local improvement.
OBE and OBEI: Order-based encoding with and without local improvement.

Table 3
Results of the problem with different container capacity

Problems	WEBr					2-step ALA-EA				
$n/C/c_i$	Best	Average	Worst	σ	CPU (s)	Best	Average	Worst	σ	CPU (s)
30/3/diff.	1.72	1.72	1.72	0.00	50	1.72	1.72	1.72	0.00	50
30/6/diff.	0.97	1.19	1.22	0.06	50	0.89	1.21	1.77	0.24	50
30/9/diff.	0.86	1.47	1.73	0.26	50	0.86	1.77	2.80	0.45	50
30/12/diff.	1.40	1.68	1.92	0.17	50	0.95	1.49	2.02	0.41	50
50/5/diff.	1.44	1.52	1.61	0.05	100	1.49	1.81	2.16	0.23	100
50/10/diff.	1.17	1.50	1.83	0.19	100	0.81	1.20	1.68	0.24	100
50/15/diff.	1.42	2.03	2.31	0.24	100	1.43	1.96	2.84	0.36	100
50/20/diff.	1.49	1.65	1.79	0.10	100	0.94	1.23	1.79	0.27	100
200/20/diff.	2.73	2.91	3.04	0.09	600	1.19	1.64	2.10	0.25	600
200/40/diff.	2.58	2.83	2.98	0.12	600	1.21	1.51	1.98	0.25	600
200/60/diff.	2.27	2.42	2.58	0.09	600	1.34	1.73	1.95	0.18	600
200/80/diff.	1.76	1.89	1.98	0.05	600	1.56	1.99	2.53	0.26	600
30/3/diff.	0.31	0.33	0.98	0.03	50	0.31	0.35	0.42	0.03	50
30/3/diff.	0.46	0.48	0.50	0.02	50	0.46	0.47	0.58	0.03	50
30/3/diff.	0.23	0.23	0.23	0.00	50	0.23	0.24	0.28	0.02	50
50/5/diff.	0.48	0.78	0.95	0.16	100	0.18	0.43	0.61	0.15	100
50/5/diff.	0.23	0.44	0.72	0.14	100	0.23	0.28	0.55	0.10	100
50/5/diff.	0.35	0.39	0.50	0.05	100	0.18	0.30	0.35	0.05	100
200/20/diff.	1.50	1.91	2.02	0.14	600	0.44	0.65	0.89	0.12	600
200/20/diff.	1.17	1.32	1.45	0.08	600	0.24	0.32	0.42	0.05	600
200/20/diff.	0.54	0.65	0.72	0.72	600	0.13	0.20	0.30	0.05	600
Total Average		1.397					1.071			

because the V_{max}^{LP} value is different from that in the previous case, we calculated it in the same way that we have explained in the previous subsection.

Because ALA-EA with 2-step and EA using the weight encoding have different computation complexity ($O(|n|\log|n|)$ for WE*Br* and $O(|Cn|\log|Cn|)$ for ALA-EA), it is unfair to use the generation limit for termination as the previous experiments. So, we adopted a time limit for the termination condition for much more fair comparison. And, for both algorithms, 15 independent runs were performed and averaged.

Table 3 shows the experimental results. Comparing the best results found by two algorithms, ALA-EA with 2-step shows a better performance than WE*Br* in most cases. Especially, except 50/5/100 and 50/15/100 instances, the proposed algorithm found superior results to WE*Br*. In case of the average results, the proposed algorithm also shows better performance than WE*Br* except small size instances with 30 items. Taking the results into account, ALA-EA with 2-step seems to be much effective at relatively bigger size instances with many items and containers. In case of standard deviation, two algorithms do not exceed 0.5 and show very stable, but WE*Br* seems to give more predictable results.

We simply report statistical result analyzed by *t*-test. WE*Br* was never statistically better than the new algorithm in these 20 cases except 30/3/diff. Especially except small instances (item = 30), the new algorithm was found to outperform WE*Br* with confidence at least 99.9% (90% at 200/80/diff.) in all cases.

6. Conclusion

This paper dealt with the MCPP that relates to how multiple items can be packed into multiple containers with the capacity constraint and we proposed a new evolutionary algorithm combining ALA-EA and 2-step local improvement algorithm. For showing the proposed algorithm's performance, we have tested it at two different types of problems.

Even though the proposed algorithm has higher computation complexity than the previous EA using weight encoding, we could confirm that it can give better performance than EA using weight encoding especially for the problem with different container capacities. Actually, at real world, the packing process is dealt with enough time because it is dependent on the shipping plan of a ship. Thus, the computation time may not be a critical factor for algorithm development. Anyway, in future work, we will investigate an algorithm that can reduce the computation time without the change of performance.

Acknowledgements

This research was financially supported by the UCN Project, the MIC 21C Frontier R&D Program in Korea, and the Center for Distributed Sensor Networks at GIST.

References

[1] Raidl GR, Kodydek G. Genetic algorithms for the multiple container packing problem. In: Proceedings of the 5th international conference on parallel problem solving from nature (PPSN V), LNCS, 1998;1498: 875–884.
[2] Martello S, Toth P. Solution of the zero-one multiple knapsack problem. Eur J Oper Res 1980;4:276–83.
[3] Pisinger D. An exact algorithm for large multiple knapsack problems. Eur J Oper Res 1999;114:528–41.
[4] Raidl GR. A weight-coded genetic algorithm for the multiple container packing problem. In: Proceedings of the 1999 ACM symposium on applied computing, 1999, p. 291–296.
[5] Raidl GR. The multiple container packing problem: a genetic algorithm approach with weighted codings. ACM Appl Comp Rev 1999;7(2):22–31.
[6] Soak SM, Corne DW, Ahn BH. A new evolutionary algorithm for spanning-tree based communication network design. IEICE Trans Commun 2005;E88-B(10):4090–3.
[7] Soak SM. 'Adaptive link adjustment' applied to the fixed charge transportation problem. IEICE Trans Fundamentals 2007;E90-A(12):2863–76.
[8] Soak SM, Corne DW, Ahn BH. The edge-window-decoder representation for tree-based problems. IEEE T Evolut Comput 2006;10(2): 124–44.
[9] Lee S, Soak SM, Park H, et al. Statistical properties analysis of real world tournament selection in gas. Appl Intell, in press.
[10] Available from: <http://www.apm.tuwien.ac.at/pub/TestProblems/mcp>.