TRANSPORT
2010
25(2): 138–147

DEVELOPMENT OF A DECISION SUPPORT SYSTEM FOR SOLVING CONTAINER LOADING PROBLEMS

Türkay Dereli[1], **Gülesin Sena Daş**[2]

[1]Dept of Industrial Engineering, University of Gaziantep, Gaziantep, 27310, Turkey
[2]TUBITAK, EU Framework Programmes, National Coordination Office, Ankara, Turkey
E-mails: [1]dereli@gantep.edu.tr; [2]sena.das@tubitak.gov.tr

Received 24 November 2009; accepted 20 May 2010

Abstract. The globalization of supply chains and rising fuel costs are forcing container carriers both to minimize the number of trips and to maximize available container space. This makes container loading (CL) a critical process especially in real-life applications. Container loading (CL), which is a difficult problem to be solved, has many applications in container transportation and distribution industries. This article presents a container loading support system (CLSS). The proposed CLSS composes of three main components, including a hybrid Bees Algorithm as the main computational algorithm, the graphical user interface (GUI) and a simulation program. The aim of the designed system is to make the packing pattern more visible to the user in order to simplify the loading process. An illustrative example – a CL problem from literature – is also provided to introduce the operation of the system and to prove its efficiency.

Keywords: decision support system, container loading problem, Bees Algorithm, graphical user interface, swarm intelligence.

1. Introduction

The globalization of supply chains has significantly increased container shipment all around the world (Su and Wang 2009; Miao *et al.* 2009; Chen and Zeng 2010; Simongáti 2010; Rohács and Simongáti 2007; Vasilis Vasiliauskas and Jakubauskas 2007; Paulauskas and Bentzen 2008). 90% of all cargo moves in containers and approximately 250 million are shipped annually (van de Voort *et al.* 2003). At the moment, rising fuel costs provide strong incentive for container carriers to maximize available container space, thereby minimizing the number of required trips across the global container transportation system (Wang *et al.* 2008). Thus, it can be concluded that the loading phase is one of the critical parts of the container shipment process.

CL problem is NP-Hard problem (Pisinger 2002) which means that the problem is too complex to be solved in polynomial time. The problem can be described as follows: Given a set of n items with width (w_i), depth (d_i) and height (h_i) and a single container with known dimensions (W, D, H) where $w_i \le W$, $h_i \le H$ and $d_i \le D$. The problem is to pack the items into a container without overlapping while maximizing the utilization rate of the container. Numerous papers have been published on CL problem and related problems. The overview of literature reveals that most of the published works on the subject utilizes different types of data structures (Morabito and Arenales 1994; Eley 2002; Lim *et al.* 2005), heuristic algorithms (George and Robinson 1980; Bischoff and Marriott 1990; Gehring *et al.* 1990; Haessler and Talbot 1990; Ngoi *et al.* 1994; Pisinger 2002; Bischoff 2003; Moura and Oliveira 2005), meta-heuristic algorithms (Gehring and Bortfeldt 1997, 2002; Yeung and Tang 2005; Faina 2000; Bortfeldt and Gehring 2001) and some parallel approaches (Gehring and Bortfeldt 2002; Bortfeldt *et al.* 2003; Mack *et al.* 2004). In most of these approaches, the aim was to fill the container with better volume utilization than the others. An interested reader can refer to Wäscher *et al.* (2007) topology to reach corresponding literature on the categories of a similar problem. This article presents a container loading support system (CLSS) that can be rarely constructed.

Even if CL problem is solved to its optimum, packing shipment into a container is a complex process which often takes several days to allocate the pooled goods into a number of containers and then packing the allocated goods into them. Occasionally, workers must unload some containers and then reload them in a different pattern to pack more goods in containers (Chien and Deng 2004). Chien and Deng (2004) were the first researchers that realized this difficulty. They proposed a decision support system (DSS) based on a heuristic packing procedure. Another DSS for a similar problem namely Air-Cargo Loading Problem was put forward by Chan *et al.* (2006). The two-phased system is suggested to load air cargo pallets efficiently using Linear Programming and heuristics.

Differently from the previously introduced DSS using heuristic algorithms, the designed CLSS uses a swarm intelligence algorithm which is the core of the system. Other than the computational algorithm, the CLSS has the graphical user interface (GUI) and a simulation program that visualize the actual packing process in a 3D manner.

In the next section, a general overview of the CLSS is supplied. Following this introduction of the overall system, each component of the system is described in detail. Finally, the functioning of the system is explained through a step by step solution of CL problem from literature.

2. Container Loading Support System

The CLSS composes of a computational algorithm based on a recently new swarm intelligence algorithm called the Bees Algorithm, the GUI and a simulation program that visualizes the actual packing process. The data flow of the system is schematized in Fig. 1.

In order to operate the CLSS, the user should select the parameters of the computational algorithm and problem data from the GUI. Later, this data is sent to the computational algorithm where actual processing is done. By using the corresponding problem data (obtained from the problem database) and the parameters defined previously by the user, the computational algorithm starts working. When the pre-determined number of iterations is reached, the data about the volume utilization of the solution is sent to the GUI and the data about the position of each box is sent to a file called the visual file containing the *x*, *y* and *z* coordinates of each box of the final solution computed by the computational algorithm. Finally, the simulation program visualizes the final packing pattern using the visual file. In the next section, the details about the computational algorithm are presented.

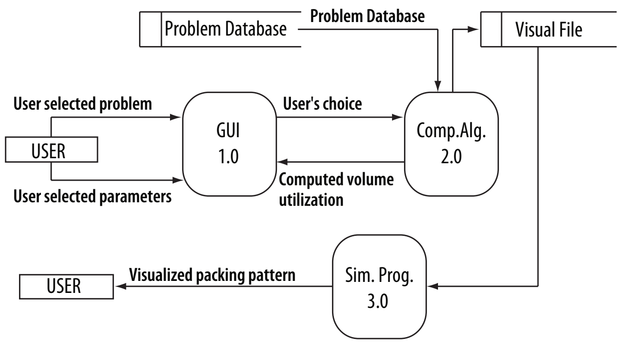


Fig. 1. The data flow diagram of the CLSS system

2.1. Computational Algorithm

The Bees Algorithm is used for the computational algorithm of the system. It is a recently new swarm intelligence algorithm. Numerous researchers have recently been inspired from the interesting features of honey bee colonies and conducted research on the subjects like foraging, learning, mating and dancing. Most of works in this field of research have been mainly influenced from (or based on) the pioneering works of Von Frisch (1967), Seeley (1955) and Dyer (2002). The utilization of the proposed models in this field and a number of algorithms based on the behaviours of honey bee colonies have been designed.

These algorithms, namely bee or bee colony algorithms, are mainly inspired from two features – mating (Abbass 2001; Afshar *et al.* 2007; Teo and Abbass 2003; Koudil *et al.* 2007) and food foraging (Lucic 2002; Yang 2005; Pham *et al.* 2006a, 2006b, 2006c).

Despite its relatively short history, the approaches inspired from the behaviours of honey bees have been applied to job shop scheduling (Chong *et al.* 2006), transportation problems (Lucic 2002), partitioning and scheduling problems (Koudil *et al.* 2007), training of multi-layered perception networks (Pham *et al.* 2006a), recognizing patterns in control charts (Pham *et al.* 2006b), optimization of continuous functions (Karaboğa and Baştürk 2007; Pham *et al.* 2006c), water resources management problems (Bozorg and Afshar 2004), data mining problems (Fathian *et al.* 2007; Benatchba *et al.* 2005) and generalized assignment problems (Baykasoğlu *et al.* 2007). Up to date, this algorithm has not been utilized to solve CL problems.

The BA algorithm is hybridized with a heuristic filling procedure to solve CL problem. This hybrid algorithm will follow the introduction of the heuristic filling procedure.

Heuristic Filling Procedure

The proposed heuristic filling procedure is a '*wall-building*' approach that loads the container *layer-by-layer* in a recursive manner. Before filling each layer, its dimensions are determined which is explained in the next Section.

Layers are filled one at a time. In case it is not possible to fill a layer with the boxes in the set of available boxes, the current layer is closed and a new one is started. The procedure is repeated until it is not possible to locate a new layer to the remaining container width or when the set of available boxes is empty. As a result of this packing process, the container is filled with the isolated vertical layers where spanning boxes between layers is avoided.

Determination of layer dimensions

Before starting the filling process, it is important to determine the dimensions of a layer since the width of the layer must be carefully selected to obtain a good performance (Pisinger 2002).

In this study, the width of each layer w_L is set equal to the width of the Layer Determining Box (LDB) (Gehring *et al.* 1990). In order to determine the LDB, first, the boxes among the set of available boxes are sorted by width dimension in a non-increasing order. Thus, the box with the greatest width dimension is given the highest priority. In case of a tie among the boxes with the same width dimension, the box with the smallest depth dimension d_i is given a higher priority. Finally, the highest priority box in the set of available boxes is chosen as the LDB.

After determining the LDB, the layer having width w_L equal to the width w_i of the LDB, height h_L and depth d_L equal to those of the container is filled. As a result, the dimensions of the layer are determined as seen in Equation (1):

$$\begin{aligned} & w_L = \max(w_i), \text{ where } i = 1,\ldots,n, \\ & h_L = H; \\ & d_L = D. \end{aligned} \tag{1}$$

Filling the layer

Following the determination of layer dimensions, it is possible to fill the layers that are filled in a recursive manner. The main advantage of using *recursive algorithms* is that they reduce the solution to a problem with a particular set of input to the solution of the same problem with smaller input values (Rosen 2006). Besides, recursive algorithms are simple and easy to implement.

When the first box (LDB) is allocated into the layer, three empty new-spaces namely '*beside*', '*in front*' and '*above*' of the packed box are produced. In the given situation (see Fig. 2 and Fig. 3), only two of these spaces '*in front*' and '*above*' occur as a result of the allocation of the first box into the layer. Thus, only these spaces are shown in Fig. 2. However, when a box having a width smaller than the LDB is packed into the layer, empty space beside the packed box occurs as seen in Fig. 3. In the proposed filling procedure, empty spaces are filled in the following order: first, the empty space '*in front*' of the packed box is filled, then the empty spaces '*beside*' and '*above*' of the packed box are filled, respectively.

Now, suppose that there is a packed box in the layer as shown in Fig. 2 and it is desired to pack the next highest priority box in the set of available boxes into the container. First, the space '*in front*' of the packed box is checked. If it is possible to allocate this box into this space, then it is packed there and removed from the set of available boxes since there is not empty space '*beside*' the packed box in the current layer and space '*above*' the packed box is checked. If it is possible to allocate this box into this space, then the box is packed to this space and the packed box is removed from the set of available boxes. Otherwise, the suitability of the next highest priority box in the set of available boxes to the available empty spaces in the layer is investigated. The process is repeated in a recursive manner for each box in the set of available boxes and for all empty spaces available in the current layer until it is not possible to fill empty spaces with a box in the set of available boxes.

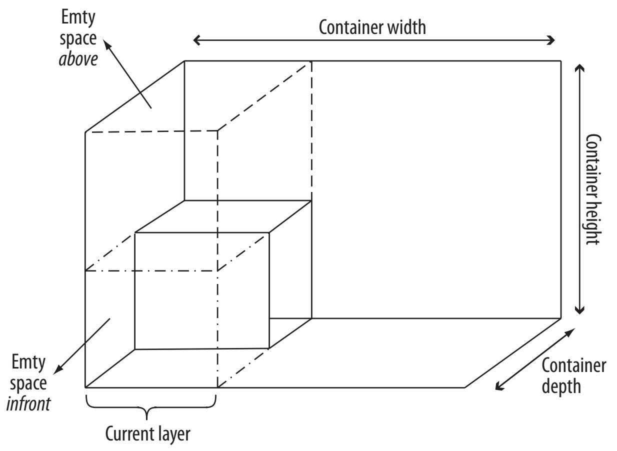


Fig. 2. Empty space in front of and above the LDB

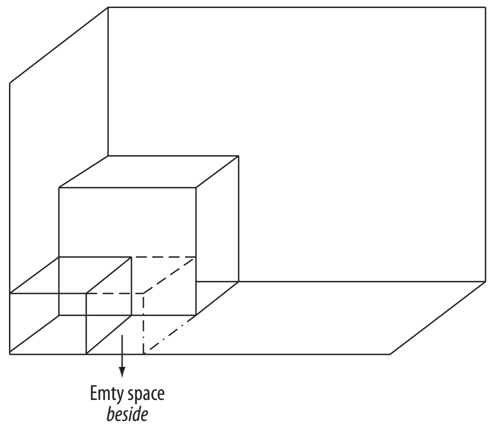


Fig. 3. Empty space beside a packed box in the layer

Having introduced the basic features of the heuristic filling procedure in detail, the main steps of the procedure are summarized in Table 1.

Table 1. The main steps of the heuristic filling procedure

Step 1. Input data on a problem and calculate the rank of each box according to the determined ranking criteria.
Step 2. Is there enough container width for a new layer? If yes, Then go to Step 3, Else go to Step 10.
Step 3. Select LDB as the box with the highest rank among the set of available boxes.
Step 4. Select the box with the highest rank in the set of available boxes.
Step 5. Is it possible to pack the selected box to the current layer? If yes, Then go to Step 6, Else go to Step 4.
Step 6. Pack the selected box and remove the packed box from the set of available boxes.
Step 7. Update the available space data in the layer.
Step 8. Is there enough space for a new box in the layer? If yes, Then go to Step 9, Else go to Step 2.
Step 9. Are there any boxes in the set of available boxes? If yes, Then go to Step 2, Else go to Step 10.
Step 10. Calculate the utilization rate of the container and Stop.

The Proposed Bees Algorithm (BA) for Container Loading Problems

Since it is essential to find out the dimensions of a layer for a good container loading performance as discussed previously, BA is considered. As mentioned before, *the width of the layers* is set equal to *the width of the boxes,* which has the highest priority in the set of available boxes. If the priority of the boxes in the set of available boxes can be altered, alternative widths for the layers can be considered to reach a good container-loading performance. In our algorithm, the priorities of the boxes (in the set of available boxes) are changed by enabling or disabling the rotation of the boxes.

The BA algorithm makes an analogy to a honey bees colony that tries to find promising food sources in nature. The natural food foraging process of a honey bee colony starts with a number of scout bees from the colony searching for food sources. When scout bees find a rich food source, they begin the so called '*waggle dance*' in the hive (Dyer 2002).

This dance, which is a form of communication between the bees in the colony, includes information about the distance, direction and quality of the food source. Equipped with this important knowledge, the colony sends follower bees – more follower bees are sent to more promising food sources – to these food sources to collect food. While collecting it, bees evaluate the food level of the source and collect the needed information for the next waggle dance. The employed BA mimics the food foraging process of the honey bee colony. The main steps of BA for CL (*hybrid-BA*) problems hybridized with the heuristic filling procedure presented in the previous section are schematized in Fig. 4 (Daş and Dereli 2007).

In the process of implementing the *hybrid-BA* for CL, several parameters should be determined. These key parameters are the number of scout bees *n*, the number of selected sites *m,* the number of elite sites *e* chosen from *m* sites, the number of bees recruited to search *e* elite sites *nep*, the number of bees recruited to search *m-e* other sites *nsp* and termination criteria. These parameters of the algorithm are adjusted by trial and error since there is not a defined procedure to help the users to choose the most appropriate set of parameters.

As can be seen in Fig. 4, the algorithm starts with *n* scout bees being placed randomly in the search space. These *n* scout bees represent the initial population. Following the acquisition of random initial solutions, the latter found by scout bees are evaluated by the proposed heuristic filling procedure. Bees that have good fitness among this initial population are selected so that *m* sites are chosen for neighbourhood search which is primarily around the best sites among these *m* sites known as elite sites *e* and other selected sites *m-e.* Here, elite sites (*e*) represent more promising solutions searched with *nep* bees greater than *nsp* bees searching the other *(m-e)* sites.

To obtain the next bee population, the bees (evaluated with the heuristic filling procedure) with the highest fitness value are selected from each *m* site. In order to complete population to *n* bees, the remaining *n-m* bees are assigned randomly to the search space in order to find new solutions. These steps are repeated until the *stopping criterion* is met or the solution converges.

At this point, it is meaningful to explain why the '*hybridization*' of the algorithm is required. The hybridization of the BA algorithm with the heuristic filling-procedure is essential, since the *objective function values* corresponding to each solution is needed all through the algorithm. Without these values of an objective function, neighbourhood search is not started in the algorithm. In Fig. 4**,** the interactions between the BA algorithm and the heuristic procedure are also illustrated. Fig. 4 clearly indicates that the heuristic procedure is called upon whenever the objective value of a solution is needed by the BA algorithm.

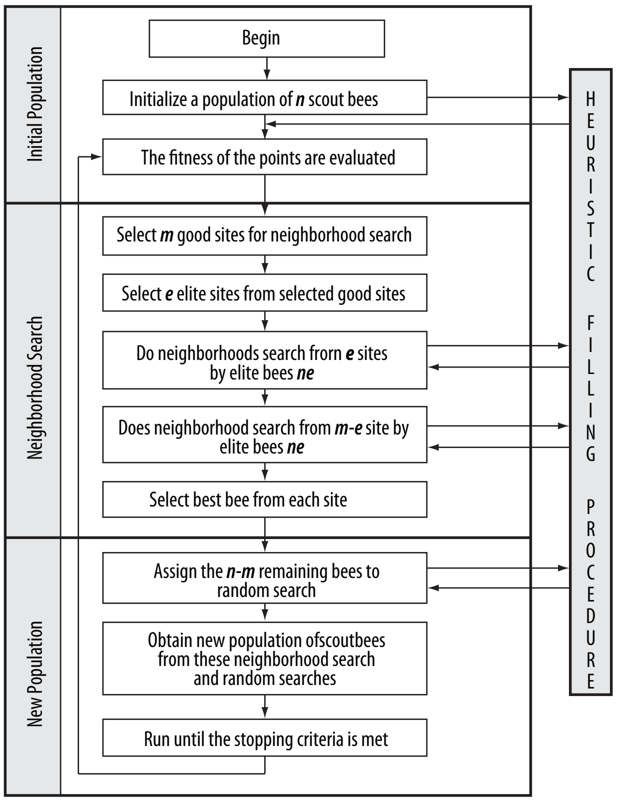

Initial Population
Neighborhood Search
New Population
Begin
Initialize a population of ***n*** scout bees
The fitness of the points are evaluated
Select ***m*** good sites for neighborhood search
Select ***e*** elite sites from selected good sites
Do neighborhoods search frorn ***e*** sites by elite bees ***ne***
Does neighborhood search from ***m-e*** site by elite bees ***ne***
Select best bee from each site
Assign the ***n-m*** remaining bees to random search
Obtain new population ofscoutbees from these neighborhood search and random searches
Run until the stopping criteria is met
HEURISTIC FILLING PROCEDURE

Fig. 4. The algorithm of the *hybrid-BA* (Daş and Dereli 2007)

Representation of a solution and neighbourhood search

Each bee in the population represents a bit string of length equal to the number of the box types of the given CL problem. Each bit in this string shows an alternative orientation of a box type (there can be different rotation orientations for different problems). Suppose that, CL problem having relevant data like one presented in Table 2 is being dealt with.

The problem contains a total of 320 boxes of 9 different box types (boxes of a similar type have the same dimensions). The bit string representation of this problem is shown in Fig. 5. Suppose that these boxes can only be base-rotated and that there is a '*this way up*' constraint for these boxes. Each bit in this string represents box type and the numbers '0' and '1' represent whether all the boxes of that type are rotated or not. The rotation of an ordinary box is shown in Fig. 6.

Table 2. Data for a CL problem

Box Type	Width	Depth	Height	Total number
Type 1 (T1)	108	65	55	45
Type 2 (T2)	95	52	45	55
Type 3 (T3)	70	62	35	20
Type 4 (T4)	83	40	20	30
Type 5 (T5)	90	70	40	18
Type 6 (T6)	55	48	37	27
Type 7 (T7)	68	20	10	34
Type 8 (T8)	100	83	44	41
Type 9 (T9)	60	32	23	50

0	1	1	0	1	0	1	1	1
T1	T2	T3	T4	T5	T6	T7	T8	T9

Fig. 5. The bit string representation of a solution (only for a base-rotated box)

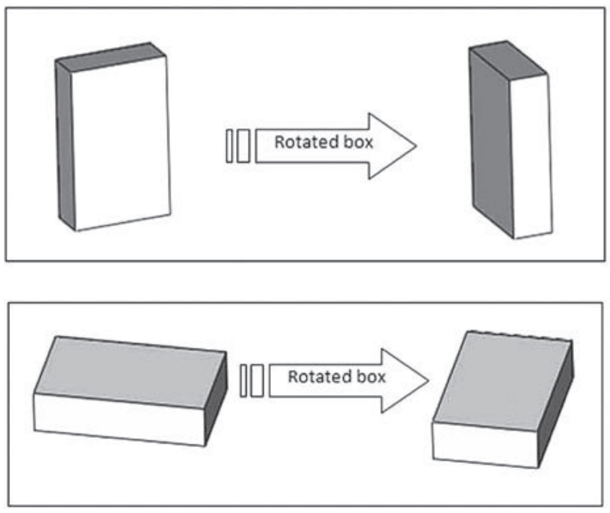
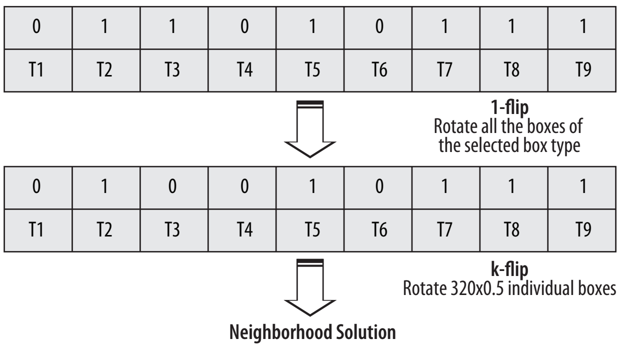


Fig. 6. The rotation of a box (only for a base-rotated box)

The solution provided in Fig. 5 suggests a program to rotate (to exchange width and depth dimensions in case only rotation on the base is allowed) all the boxes of *types* 2, 3, 5, 7, 8 and 9 allocating these boxes into the container following this new rotation-orientation. This structure is preferred to the structure where each bit in the string represents a box dealing with the problem. In this case, the resulting bit string will be of length 320 which will be a very inconvenient and time-consuming structure for large sized problems.

Two operators, namely '1-flip' and '*k*-flip', are defined in this work in order to reach neighbourhood solutions. In case of the '1-flip' type of the operator, the value of a randomly selected bit is flipped from '1 to 0' or from '0 to 1', i.e. all the boxes of a selected box type are either rotated or not.

The second operator is inspired from the work done by Kong *et al.* (2008). A simple random 4-flip method as local search was designed. This operator randomly selects four variables from the solution and flips their values from '1 to 0' or from '0 to 1'. If the newly generated solution is better, the original solution is replaced with the new one. This method was applied 1000 times for each solution. The number of flips and the number of execution through a set of experiments were selected. Similarly to this structure, an operator named '*k*-flip' is designed for this work. This operator rotates a corresponding number of boxes ('*k*' *times* 'total box number') that are randomly selected. It is possible to rotate the boxes of a *specific box-type* as well as the boxes of an *alternative box-type* using operator one after another. For example, for the sample problem provided in Table 2, the neighbourhood search process through the use of operators discussed above is illustrated and explained in Fig. 7. For the example provided above, first, '1-flip' operator is applied to the randomly selected bit. Accordingly, the randomly selected bit representing all the boxes of type 3 is rotated. Following '1-flip' operator, '*k*-flip' operator is also applied to the example of a bit string.



0	1	1	0	1	0	1	1	1
T1	T2	T3	T4	T5	T6	T7	T8	T9

1-flip
Rotate all the boxes of the selected box type

0	1	0	0	1	0	1	1	1
T1	T2	T3	T4	T5	T6	T7	T8	T9

k-flip
Rotate 320x0.5 individual boxes
Neighborhood Solution

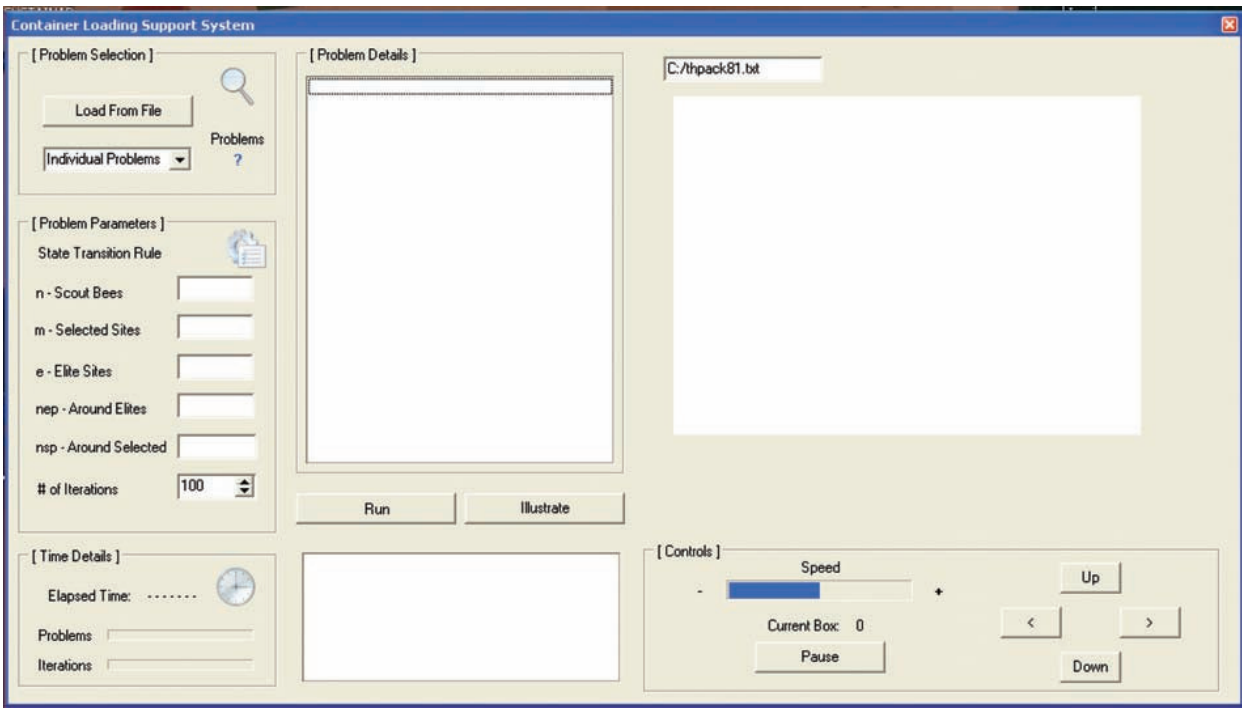
Fig. 7. The operation of defined operators

'*k*-flip' is applied to the number of a selected bit position equal to the '*k*' *times* of 'total box number' in that position (where $k < 1$). If k is selected as 0.5, then, a total of 320´0.5 randomly selected individual boxes should be rotated, for example, randomly selected boxes where box number 15 having type 1, box number 123 having type 3, etc.

If the obtained neighbourhood solution is better than the current one, then, the neighbourhood solution is saved. Otherwise, the current solution is saved. Using both operators, it is possible to evaluate a large number of solutions.

2.2. The Graphical User Interface and the Simulation Program

The GUI of the CLSS is designed in Borland Builder. This interface is used to select the problem type and parameters of the computational algorithm and to visualize the packing pattern. The snapshot of the interface is shown in Fig. 8. A simulation program is used to visualize the packing pattern. The simulated program is coded in OpenGL and integrated into the GUI for user friendly use.



Individual Problems

C:/thpack81.txt

# of Iterations 100

Current Box: 0

Fig. 8. The snapshot of the GUI

3. Illustrative Application

The following example is supplied to demonstrate how the CLSS system works in a step-by step manner. The problem is selected from the Loh and Nee (LN) (1992) test cases which are a set of well known test problems. The aim is to allocate a set of boxes with varying dimensions into a container without any overlap to maximize the volume utilization of the container dealing with each problem.

Step 1: The user selects a test case composed of a set of problems from the GUI and determines the individual problem from this set. The snapshot of the CLSS presented in Fig. 9 illustrates the Problem Selection window used for this operation. Suppose that the user selects the LN test cases and problem number 12. A container having the width of 3200, the depth of 2400 and the height of 1000 units and a total of 120 boxes of 6 different box types are considered in this example. The dimensions of the boxes corresponding to the selected problem are presented in Table 3.

Step 2: Following the selection of the problem from the *Problem Selection* window, the user enters directly the parameters of the computational algorithm to the cells in the *Problem Parameters* section that includes the parameters regarding the hybrid-BA.

These parameters comprise the number of scout bees *n*, the number of selected sites *m*, the number of elite sites *e* chosen from *m* sites, the number of bees recruited to search *e* elite sites *nep*, the number of bees recruited to search *m-e* other sites *nsp* and the maximum number of iterations as termination criteria.

Suppose that for the solution of the selected LN problem, the following set of parameters is used: the number of bees (population) *n* = 5, the number of selected sites *m* = 3, the number of elite sites *e* = 2, the number of bees send to elite points *nep* = 4, the number of bees sent to other selected points *nsp* = 3 and the maximum number of iterations = 1000. Although such a choice of parameters is used in this example, the user is given the opportunity to try different values for the parameters of the computational algorithm.

Table 3. Data on the user selected proble

Type	Width	Depth	Height	Number of boxes
1	900	275	200	10
2	400	350	275	33
3	1200	300	250	10
4	500	375	275	27
5	800	400	200	15
6	600	300	225	25

Suppose that for the solution of the selected LN problem, the following set of parameters is used: the number of bees (population) *n* = 5, the number of selected sites *m* = 3, the number of elite sites *e* = 2, the number of bees send to elite points *nep* = 4, the number of bees sent to other selected points *nsp* = 3 and the maximum number of iterations = 1000. Although such a choice of parameters is used in this example, the user is given the opportunity to try different values for the parameters of the computational algorithm.

Step 3: Having selected the problem and parameters, the user can run the computational algorithm by clicking the 'RUN' button.

Step 4: The computational algorithm works until the number of iterations is reached. Then, data about the selected problem and obtained solution is viewed in the *Problem Details* section of the GUI. Besides the value of the obtained solution, it is also possible to learn the elapsed time passed to solve the problem in the *Time Details* section.

Step 5: When the user clicks the 'ILLUSTRATE' button on the spreadsheet, the simulation program attached to the GUI executes and visualizes the packing pattern. The program reads the *x*, *y* and *z* coordinates of each box from the visual data file and shows how each box is loaded into the 3D container one by one. The resultant 3D graph of the packing pattern can be viewed from different angles using the arrow buttons located below the three dimensional graph where each loaded box is shown with different colours. It is also possible both to pause simulation using the 'PAUSE' button and to adjust the speed of simulation using the '–/+' buttons on both sides of the Speed bar.

Fig. 10 and Fig. 11 illustrate the final packing pattern for the selected problem together with the volume utilization of the problem in the *Problem Details* section. As can be seen from Fig. 10 and Fig. 11, the volume utilization of 78.5% (optimal value for this problem) is reached in 78 seconds.

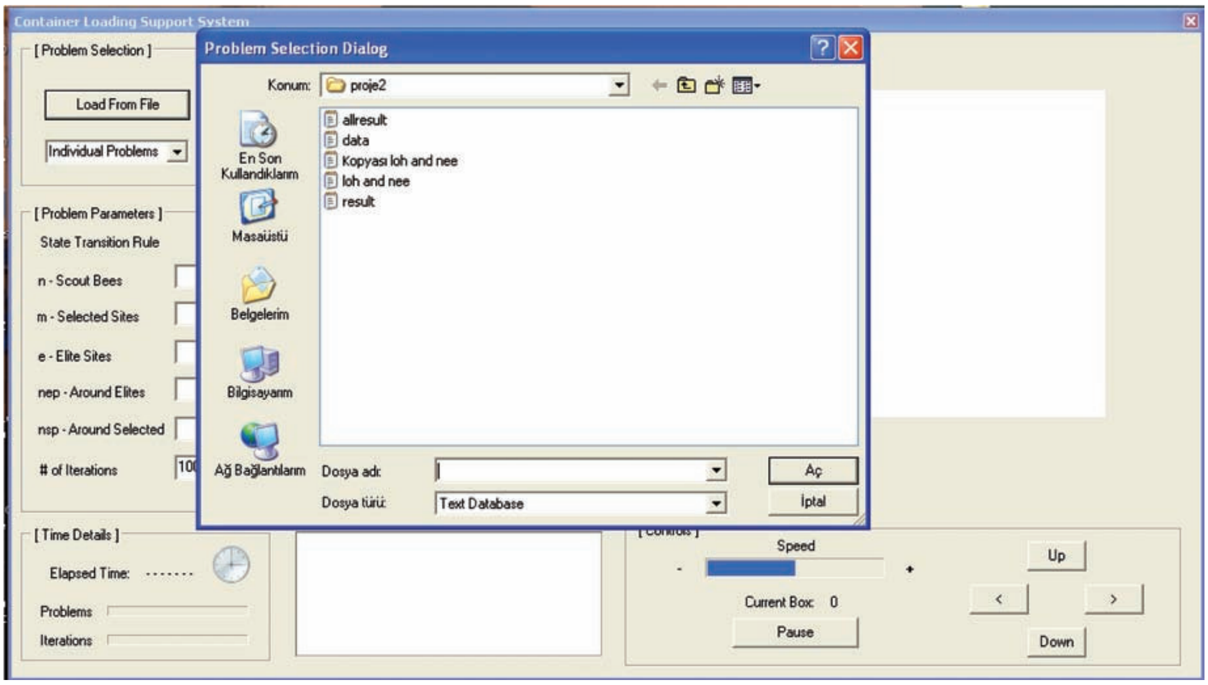
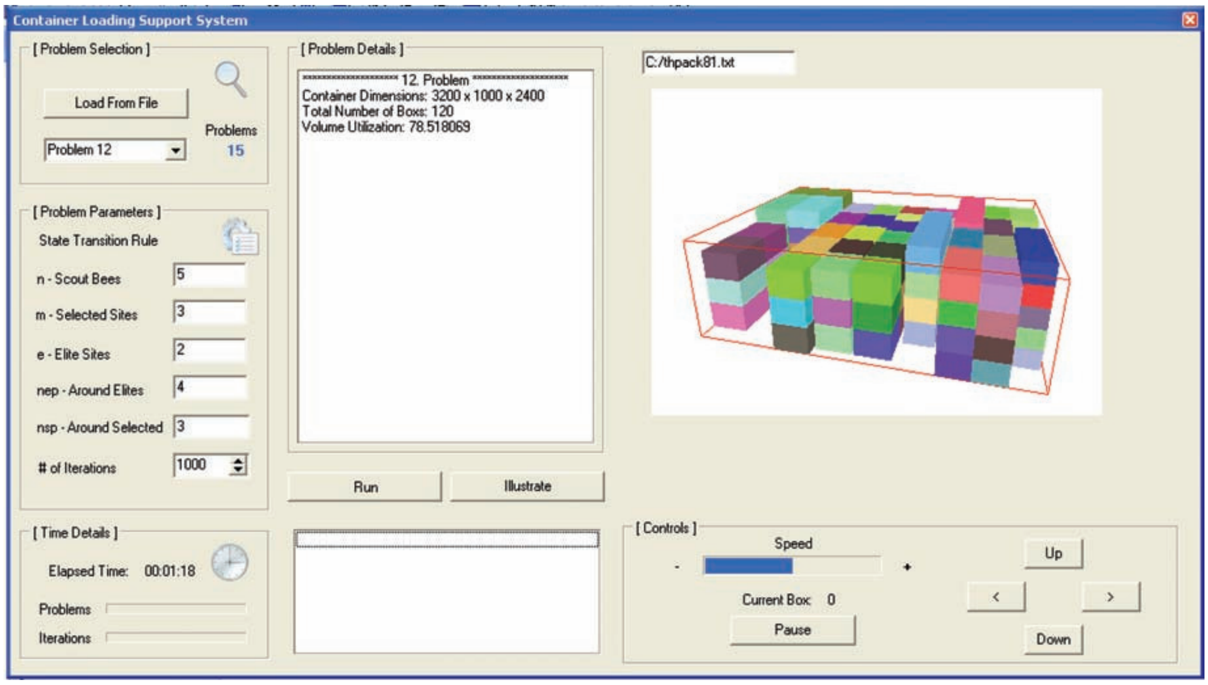


Fig. 9. The dialog window for Problem Selection



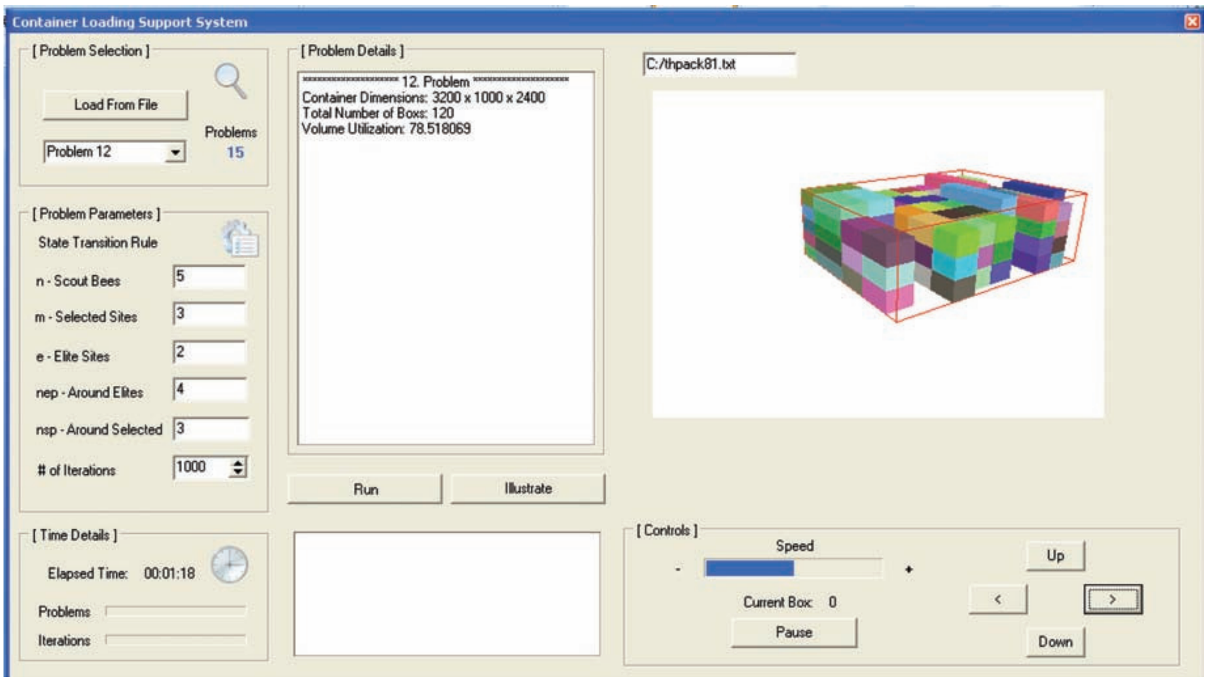
Problem 12
Problems 15

n - Scout Bees 5
m - Selected Sites 3
e - Elite Sites 2
nep - Around Elites 4
nsp - Around Selected 3
# of Iterations 1000

```
************** 12. Problem **************
Container Dimensions: 3200 x 1000 x 2400
Total Number of Boxs: 120
Volume Utilization: 78.518069
```

Elapsed Time: 00:01:18

Fig. 10. The final packing pattern obtained for the problem of 120 boxes



Problem 12
Problems 15

```
****************** 12. Problem ******************
Container Dimensions: 3200 x 1000 x 2400
Total Number of Boxs: 120
Volume Utilization: 78.518069
```

C:/thpack81.txt

State Transition Rule
n - Scout Bees: 5
m - Selected Sites: 3
e - Elite Sites: 2
nep - Around Elites: 4
nsp - Around Selected: 3
# of Iterations: 1000

Elapsed Time: 00:01:18
Problems
Iterations

Current Box: 0

Fig. 11. The final packing pattern obtained for the problem of 120 boxes

4. Conclusions

In this paper, a CLSS to determine and visualize the container packing pattern of the CL process is presented. The system consists of three main components that include the computational algorithm based on the Bee Algorithm hybridized with a heuristic filling procedure, the GUI and the Simulation Program. The Computational Algorithm solves the selected problem using the input from the GUI coded in Borland Builder, enables the determination of problem related choices by the user (input to the Computational Algorithm) and the Simulation Program coded via OpenGL, illustrates the final packing pattern in a 3D manner. The DSS proposed in this study has been also used in a medium-sized company with positive results.

The presented simulation program, which is a component of the proposed DSS, can be used independently of the CLSS to visualize a packing pattern computed with a different algorithm. Thus, this component of the system can also be used for academic work on CL problems where visualizing the packing pattern is necessary.

References

Abbass, H. A. 2001. MBO: Marriage in honey bees optimization a haplometrosis polygynous swarming approach, in *Proceedings of the 2001 Congress on Evolutionary Computation, Vols 1 and 2*, 207–214.

Afshar, A.; Haddad, O. B.; Marino, M. A, Adams, B. J. 2007. Honey-bee mating optimization (HBMO) algorithm for optimal reservoir operation, *Journal of the Franklin Institute–Engineering and Applied Mathematics* 344(5): 452–462. doi:10.1016/j.jfranklin.2006.06.001

Baykasoğlu, A.; Özbakır, L.; Tapkan, P. 2007. Artificial bee colony algorithm and its application to generalized assignment problem, in *Swarm Intelligence: Focus on Ant and Particle Swarm Optimization* by Felix T. S. Chan & Manoj Kumar Tiwari (Eds), 113–144.

Benatchba, K.; Admane, L.; Koudil, M. 2005. Using bees to solve a data mining problem expressed as a max-sat one, *Lecture Notes in Computer Science* 3562: 212–220. doi:10.1007/11499305_22

Bischoff, E. E. 2003. *Dealing with Load Bearing Strength Considerations in Container Loading Problems*. EBMS Working Paper EBMS/2003/3, European Business Management School University of Wales, Swansea.

Bischoff, E. E.; Marriott, M. D. 1990. A comparative evaluation of heuristics for container loading, *European Journal of Operational Research* 44(2): 267–276. doi:10.1016/0377-2217(90)90362-F

Bortfeldt, A.; Gehring, H. 2001. A hybrid genetic algorithm for the container loading problem, *European Journal of Operational Research* 131(1): 143–161. doi:10.1016/S0377-2217(00)00055-2

Bortfeldt, A.; Gehring, H.; Mack, D. 2003. A parallel tabu search algorithm for solving the container loading problem, *Parallel Computing* 29(5): 641–662. doi:10.1016/S0167-8191(03)00047-4

Bozorg, H. O.; Afshar, A. 2004. MBO algorithm: a new heuristic approach in hydrosystems design and operation, in *Proceedings of the 1st International Conference on Managing Rivers in the 21st Century*, 499–504.

Chan, F. T. S.; Bhagwat, A.; Kumar, N.; Tiwari, M. K.; Lam, P. 2006. Development of a decision support system for air-cargo pallets loading problem: A case study, *Expert Systems with Applications* 31(3): 472–485. doi:10.1016/j.eswa.2005.09.057

Chen, C.; Zeng, Q. 2010. Designing container shipping network under changing demand and freight rates, *Transport* 25(1): 46–57. doi:10.3846/transport.2010.07

Chien, C. F.; Deng, J. F. 2004. A container packing support system for determining and visualizing container packing patterns, *Decision Support Systems* 37(1): 23–34. doi:10.1016/S0167-9236(02)00192-6

Chong, C. S.; Low, M. Y. H.; Sivakumar, A. I.; Gay, K. L. 2006. A bee colony optimization algorithm to job shop scheduling, in *Proceedings of the 2006 Winter Simulation Conference*, Vols 1–5, 1954–1961.

Daş, G. S.; Dereli, T. 2007. Container loading using hybrid bees algorithm, in *Proceedings of The EU/ME 2007 – Metaheuristics in the Service Industry, 8th Workshop of the EURO Working Group, EU/ME, The European Chapter on Metaheuristics,* University of Hohenheim, October 04–05 2007, Hohenheim, Stuttgart, Germany, 52–59.

Dyer, F. C. 2002. The biology of the dance language, *Annual Review of Entomology* 47: 917–949. doi:10.1146/annurev.ento.47.091201.145306

Eley, M. 2002. Solving container loading problems by block arrangement, *European Journal of Operational Research* 141(2): 393–409. doi:10.1016/S0377-2217(02)00133-9

Faina, L. 2000. A global optimization algorithm for the three-dimensional packing problem, *European Journal of Operational Research* 126(2): 340–354. doi:10.1016/S0377-2217(99)00292-1

Fathian, M.; Amiri, B.; Maroosi, A. 2007. Application of honey-bee mating optimization algorithm on clustering, *Applied Mathematics and Computation* 190(2): 1502–1513. doi:10.1016/j.amc.2007.02.029

Gehring, H.; Bortfeldt, A. 1997. A genetic algorithm for solving the container loading problem, *International Transactions in Operational Research* 4(5–6): 401–418. doi:10.1111/j.1475-3995.1997.tb00095.x

Gehring, H.; Bortfeldt, A. 2002. A parallel genetic algorithm for solving the container loading problem, *International Transactions on Operational Research* 9(4): 497–511. doi:10.1111/1475-3995.00369

Gehring, H.; Menschner, K.; Meyer, M. 1990. Computer-based heuristic for packing pooled shipment containers, *European Journal of Operational Research* 44(2): 277–288. doi:10.1016/0377-2217(90)90363-G

George, J. A.; Robinson, D. F. 1980. A heuristic for packing boxes into a container, *Computers & Operations Research* 7(3): 147–156. doi:10.1016/0305-0548(80)90001-5

Haessler, R. W.; Talbot, F. B. 1990. Load planning for shipments of low density products, *European Journal of Operational Research* 44(2): 289–299. doi:10.1016/0377-2217(90)90364-H

Karaboğa, D.; Baştürk, B. 2007. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm, *Journal of Global Optimization* 39(3): 459–471. doi:10.1007/s10898-007-9149-x

Kong, M.; Tian, P.; Kao, Y. C. 2008. A new ant colony optimization algorithm for the multidimensional knapsack problem, *Computers and Operations Research* 35(8): 2672–2683. doi:10.1016/j.cor.2006.12.029

Koudil, M.; Benatchba, K.; Tarabet, A.; Sahraoui, E. B. 2007. Using artificial bees to solve partitioning and scheduling problems in codesign, *Applied Mathematics and Computation* 186(2): 1710–1722. doi:10.1016/j.amc.2006.08.166

Lim, A.; Rodrigues, B.; Yang, Y. 2005. 3-D Container packing heuristic, *Applied Intelligence* 22(2): 125–134. doi:10.1007/s10489-005-5601-0

Loh, H. T.; Nee, A. Y. C. 1992. A packing algorithm for hexahedral boxes, in *Proceedings of Industrial Automation Conference,* Singapore, May 1992, 2: 115–126.

Lucic, P. 2002. *Modeling Transportation Problems Using Concepts of Swarm Intelligence and Soft Computing*, Dissertation submitted to the Faculty of Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in Civil Engineering. 147 p. Available from Internet: <http://scholar.lib.vt.edu/theses/available/etd-03092002-231724/unrestricted/Panta_Lucic_ED.pdf>.

Mack, D.; Bortfeldt, A.; Gehring, H. 2004. A parallel hybrid local search algorithm for the container loading problem, *International Transactions in Operational Research* 11(5): 511–533. doi:10.1111/j.1475-3995.2004.00474.x

Miao, X.; Yu, B.; Xi, B. 2009. The uncertainty evaluation method of supply chain reliability, *Transport* 24(4): 296–300. doi:10.3846/1648-4142.2009.24.296-300

Morabito, R.; Arenales, M. 1994. An and/or-graph approach to the container loading problem, *International Transactions in Operational Research* 1(1): 59–73. doi:10.1016/0969-6016(94)90046-9

Moura, A.; Oliveira, J. F. 2005. A grasp approach to the container-loading problem, *IEEE Intelligent Systems* 20(4): 50–57. doi:10.1109/MIS.2005.57

Ngoi, B. K. A.; Tay, M. L.; Chua, E. S. 1994. Applying spatial representation techniques to the container packing problem, *International Journal of Production Research* 32(1): 111–123. doi:10.1080/00207549408956919

Paulauskas, V.; Bentzen, K. 2008. Sea motorways as a part of the logistics chain, *Transport* 23(3): 202–207. doi:10.3846/1648-4142.2008.23.202-207

Pham, D. T.; Ghanbarzadeh, A.; Koç, E.; Otri, S.; Rahim, S.; Zaidi, M. 2006a. The bees algorithm – a novel tool for complex optimisation problems, in *Proceedings of the 2nd International Virtual Conference on Intelligent Production Machines and Systems (IPROMS'2006)*. Available from Internet: <http://conference.iproms.org/sites/conference.iproms.org/files/PID217625.pdf>.

Pham, D. T.; Koç, E, Ghanbarzadeh A, Otri, S. 2006b. Optimization of the weights of multi-layered perceptrons using the bees algorithm, in *Proceedings of the 5th International Symposium on Intelligent Manufacturing Systems*, Sakarya, Turkey, 38–46.

Pham, D. T.; Otri, S.; Ghanbarzadeh, A.; Koc, E. 2006c. Application of the bees algorithm to the training of learning vector quantization networks for control chart pattern recognition, in *Proceedings of International Conference on Information and Communication Technologies*, 24–28 April 2006, Damascus, Syria, 1624–1629.

Pisinger, D. 2002. Heuristics for the container loading problem, *European Journal of Operational Research* 141(2): 382–392. doi:10.1016/S0377-2217(02)00132-7

Rohács, J.; Simongáti, G. 2007. Th e role of inland waterway navigation in a sustainable transport system, *Transport* 22(3): 148–153.

Rosen, K. H. 2006. *Discrete Mathematics and its Applications*. 6th edition, McGraw-Hill Science/Engineering/Math. 1008 p.

Seeley, T. D. 1955. *The Wisdom of the Hive*. Cambridge: Harvard University Press.

Simongáti, G. 2010. Multi-criteria decision making support tool for freight integrators: selecting the most sustainable alternative, *Transport* 25(1): 89–97. doi:10.3846/transport.2010.12

Su, T.-J.; Wang, P. 2009. Carrier's liability under international maritime conventions and the UNCITRAL draft convention on contracts for the international carriage of goods wholly or partly by sea, *Transport* 24(4): 345–351. doi:10.3846/1648-4142.2009.24.345-351

Teo, J, Abbass, H. A. 2003. A true annealing approach to the marriage in honey-bees optimization algorithm, *International Journal of Computational Intelligence and Applications (IJCIA)* 3(2): 199–211. doi:10.1142/S146902680300094X

Van de Voort, M.; O'Brien, K. A.; Rahman, A.; Valeri, L. 2003. *'Seacurity': Improving the Security of the Global Sea-Container Shipping System*, Rand Corporation. 19 p. Available from Internet: <www.rand.org/pubs/monograph_reports/2005/MR1695.pdf>.

Vasilis Vasiliauskas, A.; Jakubauskas, G. 2007. Principle and benefits of third party logistics approach when managing logistics supply chain, *Transport* 22(2): 68–72.

Von Frisch, K. 1967. *The Dance Language and Orientation of Bees*. Cambridge MA: Harvard University Press.

Wang, Z.; Li, K. W.; Levy, J. K. 2008. A heuristic for the container loading problem: A tertiary-tree-based dynamic space decomposition approach, *European Journal of Operational Research* 191(1): 86–99. doi:10.1016/j.ejor.2007.08.017

Wäscher, G.; Haussner, H.; Schumann, H. 2007. An improved typology of cutting and packing problems, *European Journal of Operational Research* 183(3): 1109–1130. doi:10.1016/j.ejor.2005.12.047

Yang, X. S. 2005. Engineering optimizations via nature-inspired virtual bee algorithms, *Lecture Notes in Computer Science* 3562: 317–323. doi:10.1007/11499305_33

Yeung, L. H. W.; Tang, W. K. S. 2005. A hybrid genetic approach for container loading in logistics industry, *IEEE Transactions on Industrial Engineering* 52(2): 617–627. doi:10.1109/TIE.2005.844224