Ant Algorithms for a Truck Loading Problem with Multiple Destinations

Jean Respen[1] and Nicolas Zufferey[1]

GSEM, University of Geneva, Switzerland, jean.respen@unige.ch, n.zufferey@unige.ch

Keywords: truck loading, bin-packing, evolutionary algorithm.

1 Introduction

The French car manufacturer *Renault* is daily facing a NP-hard combinatorial optimization problem – called here the *Renault* truck loading problem ($RTLP$) – where items need to be placed in a truck while satisfying different constraints. More than a thousand trucks are daily considered, which have to deliver goods to several car factories. As a single truck can deliver goods to different delivery points, classes of items have been defined, where a class is associated with a delivery point. Each problem instance contains the size of the truck (in millimeters) and the various sizes of all the items that must fit in (in millimeters). The heights of the items can be ignored as they rely on complex factory constraints which are supposed to be already satisfied. At first sight, this problem seems related to a strip-packing 2D problem with rotation, which has been already covered by many research papers (Lodi *et. al.* 2002). New features are proposed by *Renault* in $RTLP$: different *classes* of items, and a significant *number* of items per truck in conjunction with a large *standard deviation* of the sizes of the items. Such elements make the considered problem more complex, but more relevant to modern and realistic issues. In this paper, we propose to tackle $RTLP$ with *Constructive Ant Systems* (CAS), which are evolutionary population-based metaheuristics. A good survey on ant algorithms can be found in (Dorigo *et. al.* 2006).
Based on (Respen and Zufferey 2013), the problem and the solution space structure are formally described in Section 2, where an efficient decoding greedy algorithm is also designed. In Section 3 are proposed different ant algorithms for $RTLP$. Numerical experiments are reported and discussed in Section 4.

2 Description of the problem and solution space structure

$RTLP$ can be formally described as follows. A number n of rectangular items have to be placed in a truck (of width W_t and length L_t), and $90°$ rotations of items are allowed. For each item i, we know its width w_i, its length l_i, its initial orientation, and its class C_j (where $j \in \{1, \ldots, m\}$ with $m \leq n$). In addition, the classes must be placed in an increasing fashion from the front to the rear of the truck. More precisely, the ordinate of the origin item which belongs to class C_i (label 1 on Figure 1) must be strictly smaller than the ordinate of the extremity of any item of class C_{i+1} (label 2 on Figure 1). The goal consists in minimizing the ordinate f of the extremity item (the closest one to the rear) of class C_m (label 3 on Figure 1).
To tackle bin-packing problems, one can work either with *direct coded* solutions or *indirect coded* solutions. A direct coded solution directly represents a real loading of the items in

the truck, which means that the position of each item has to be continuously known. Then, if an item is added to or removed from the truck, the new position of the loaded items is very hard to recompute, which is a major drawback. Working with indirect coded solutions require the use of a *decoding* algorithm to built the associated direct coded solution and to get its value. Such an approach has the main advantage of being very flexible when adding (resp. removing) an item to (resp. from) the solution. Therefore, a decision has been made to work with *indirect coded* solutions. More formally, an indirect coded solution s is a sequence of elements. To build a direct coded solution $\hat{s}$ and compute its value $f(s)$, a *decoding greedy algorithm* (DGA) is performed on the indirect coded solution s. Component i of the indirect coded solution s takes the form $s_i = (ID_i, \mathcal{C}_i)$, where $ID \in \{1, \ldots, n\}$, and $\mathcal{C} \in \{1, \ldots, m\}$. DGA decodes the vector s into a real solution $\hat{s}$ (i.e. a true loading) by inserting in $\hat{s}$ the items from s in a $FIFO$ order. At each step, DGA pops the next item i of s and orients it greedily (i.e. by minimizing the augmentation of the loading length). The complexity of DGA is $\mathcal{O}(n)$. For instances with $n = 60$, DGA requires less than 0.1 second (on the used computer) to decode a solution into a real loading. In summary, DGA is requested to built a direct coded solution (i.e. a loading of the truck) from an indirect coded solution (i.e. a loading sequence of items) and to compute the length of the obtained loading (i.e. the value of the solution).
Preliminary tests showed that the performance of DGA is rather poor. Indeed, at each iteration, DGA orients the involved item without having any visibility on the next items to insert. To reduce this drawback, the following *look-ahead* process is proposed. At each iteration, DGA evaluates the orientation of the involved unloaded item (say i) as follows. DGA examines the insertion of the next σ (parameter tuned to 3) insertions subsequent to the insertion of item i (i.e. a total number of $\sigma + 1$ items are tested). DGA tries each possible orientation (i.e. 90°-rotated or not-rotated) for each of the $\sigma + 1$ items in order to minimize the augmentation of the resulting loading length (in other words, $2^{(\sigma+1)}$ options are evaluated). The orientation of item i is then the one associated with the best option. For instances with $n = 60$, DGA requires less than 0.5 second (on the used computer) to decode a solution into a real loading.

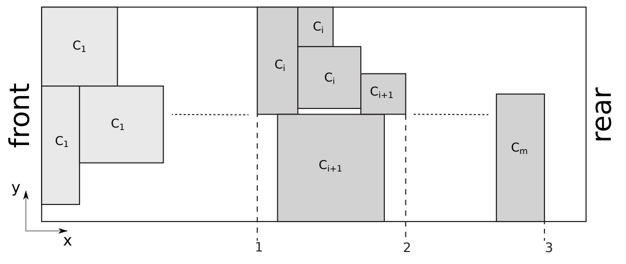

Fig. 1. A possible solution for $RTLP$

3 Ant algorithms for $RTLP$

We propose three different constructive ant algorithms to tackle $RTLP$ (denoted $CAS(1)$, $CAS(2)$ and $CAS(3)$), where each ant is a constructive heuristic able to build an indirect coded solution, and thus a loading. N (parameter tuned to 10) ants are used at each generation. For each ant, at iteration i of its construction process, let $s = (s_1, s_2, \ldots, s_{i-1})$

be the partial indirect coded solution containing $i-1$ items. As in classical ant algorithms, the selection of the next move (corresponding here as the next item to load) is based on the *greedy force* (defined as the short term profit) and the *trail system* (which is a central memory based on the history of the search, allowing ants to exchange information). In the proposed ant algorithms, the GF's and the TR's are normalized within interval [0,1] in order to better control these two types of information.
The greedy force $GF(i)$ of an item i is given by $GF(i) = UB - DGA(s + \{i\})$, where $UB = 4 \cdot L_t$ is an upper bound of the loading. We can see that the larger $GF(i)$ is, the shorter is the loading $s + \{i\}$, which is consistent with the notion of short term profit.
To define the trail $TR(i)$ of item i, we first define the *attractiveness* $Attract(j,k)$ between two items j and k in an indirect coded solution s as $n - dist(j,k)$, where $dist(j,k)$ is defined as the number of items between j and k in s (but $dist(j,k) = 0$ if k appears before j in s). At the end of each generation (i.e. when each ant of the population has provided a solution), each trail $TR(j,k)$ is updated as follows: $TR(j,k) = \rho \cdot TR(j,k) + (1-\rho) \cdot \Delta TR(j,k)$, where $\rho \in]0,1[$ is an *evaporation* coefficient (fixed to 0.9, as in most of the ant algorithms) and $\Delta TR(j,k)$ is a *reinforcement* term. $\Delta TR(j,k)$ is defined as the average attractiveness between j and k in the $b\%$ (parameter tuned to 50%) best solutions of the current generation. The trail value $TR(i)$ of item i can now be defined as $TR(i) = \sum_{j \in s} w_{ji} \cdot TR(j,i)$, where w_{ji} is a weight defined as $index(j)/index(i)$, where $index(j)$ is for example three is item j is the 3rd component of s. These weights allows to give more importance to pairs of items which are closer in s (according to the distance function $dist$).
In $CAS(1)$, when constructing an indirect coded solution, each ant selects the next item i to load – in the set Ω of non already inserted items – using the standard probability function *prob* defined as in Equation (1), where α and β are parameters. Interestingly, preliminary experiments showed that better results are obtained with parameter α (tuned to 0.5) smaller than parameter β (tuned to 2). It means that more importance should be given to the greedy force rather than to the trail.

$$prob(i) = \frac{GF(i)^{\alpha} \cdot TR(i)^{\beta}}{\sum\limits_{j \in \Omega} GF(j)^{\alpha} \cdot TR(j)^{\beta}} \tag{1}$$

In $CAS(2)$, an ant chooses the next item i using Equation (1) with probability p (parameter tuned to 0.35), but the item which maximizes $GF(j) \cdot TR(j)$ otherwise (i.e. with probability $(1-p)$). In other words, $CAS(2)$ is more aggressive than $CAS(1)$ as the usual tradeoff between the greedy forces and the trails only occurs with probability p.
In $CAS(3)$, each ant selects the next item i as follows. First, it generates the set A with the $q\%$ (parameter tuned to 0.75) largest TR values. Then among the set A, it selects the item with the largest GF value (ties are broken randomly). In contrast with $CAS(1)$ and $CAS(2)$, the greedy forces and the trails are *successively* used in order to select the next item (instead of *jointly*).

4 Results

In order to better benchmark the results, we compare the ant algorithms with an exhaustive greedy method EG. EG builds an indirect coded solution s from scratch, and at each step greedily inserts the next item in s (with the use of the *look-ahead* process). We consider a set of 30 real benchmark instances provided by *Renault*. Tests were performed on an Intel Quad-core i7 @ 3.4 GHz with 8 GB DDR3 of RAM memory, with a time limit T of 900 seconds. In order to have fair comparisons, EG is restarted as long as T is not reached, whereas the CAS's results are averaged over 10 runs. Table 1 presents the results of the different algorithms. For each instance ID are first given the number n of items and the number m of classes. Column $f^\star$ indicates the best objective function value returned by any of the considered algorithms. The second column EG shows the percentage gap between the best solution value returned by EG and $f^\star$. The remaining columns show the same information for the ant algorithms. The last row indicates the average gap of each method. We remark that all the CAS methods outperform EG, which means that the trail system is relevant. $CAS(2)$ outperforms $CAS(1)$, which shows that an aggressive selection process seems to be more interesting than the use of Equation (1). As the best methods is $CAS(3)$, it seems that a sequential use of the trails and the greedy forces in the selection process of an ant seems to be more efficient than the joint use of these quantities (as in $CAS(1)$, $CAS(2)$, and most of the state-of-the-art ant algorithms). This confirms the observations of (Zufferey 2012) for the famous graph coloring problem.

Table 1. Obtained results

ID	n	m	$f^\star$	EG	$CAS(\mathbf{1})$	$CAS(\mathbf{2})$	$CAS(\mathbf{3})$
1	23	1	12970	1.62%	0.80%	1.63%	0.13%
2	25	1	13226	2.07%	0.70%	0.64%	1.48%
3	24	1	12950	2.59%	1.06%	1.38%	0.70%
4	25	1	13170	2.75%	1.32%	1.02%	2.37%
5	26	1	13470	0.07%	1.02%	0.80%	1.02%
6	20	2	14000	2.07%	2.07%	1.73%	1.77%
7	23	1	12980	4.21%	2.74%	2.24%	1.26%
8	25	1	14288	3.26%	1.95%	2.31%	2.37%
9	18	4	13369	0.44%	3.55%	3.40%	4.50%
10	23	3	13068	3.48%	3.92%	3.46%	4.02%
11	20	2	13560	2.06%	1.79%	1.92%	1.89%
12	17	3	12992	2.42%	0.81%	0.88%	0.80%
13	25	1	13470	1.97%	0.93%	1.23%	0.54%
14	20	2	13240	3.55%	3.50%	3.74%	3.44%
15	20	4	13685	2.26%	2.83%	3.27%	2.99%
16	24	1	13070	3.21%	1.71%	2.10%	0.88%
17	23	4	13078	2.03%	1.43%	1.05%	2.09%
18	24	1	13380	4.56%	1.45%	2.27%	0.68%
19	24	1	13380	4.56%	1.45%	2.27%	0.68%
20	23	1	13070	5.05%	2.41%	2.28%	1.29%
21	25	1	13146	2.84%	1.64%	1.44%	2.03%
22	25	1	13470	1.60%	1.04%	1.00%	0.09%
23	24	1	13380	4.04%	1.34%	1.75%	0.97%
24	18	2	11640	2.41%	2.72%	1.67%	1.86%
25	23	1	12550	1.20%	1.87%	1.16%	1.79%
26	19	2	12220	1.06%	1.91%	1.48%	1.69%
27	23	1	13250	5.86%	1.35%	2.08%	0.77%
28	25	1	13416	1.97%	1.66%	1.14%	0.64%
29	20	1	13500	10.74%	5.13%	4.02%	4.79%
30	25	1	13196	3.80%	1.93%	1.07%	2.10%
AVG				**2.99%**	**1.93%**	**1.88%**	**1.72%**

References

Dorigo, M., Birattari, M., and Stuetzle, T., 2006, "Ant colony optimization - artificial ants as a computational intelligence technique", *IEEE Computational Intelligence Magazine*, Vol. 1, pp. 28-39.

Lodi, A., Martello, S., and Monaci, M., 2002, "Two-dimensional packing problems: A survey", *European Journal of Operational Research*, Vol. 141, pp. 241-252.

Respen, J., Zufferey, N., 2013, "A Renault truck loading problem: from benchmarking to improvements", *Proceedings of the 14th EU/ME Workshop, Hamburg, Germany. February 28 to March 1.*, pp. 79-84.

Zufferey, N., "Optimization by ant algorithms: Possible roles for an individual ant", *Optimization Letters*, Vol. 6 (5), pp. 963-973.