

## METAHEURISTICS OPTIMIZATION VIA MEMORY TO SOLVE THE PROFITABLE ARC TOUR PROBLEM

Jalel EUCHI, Habib CHABCHOUB

GIAD Laboratory / FSEGS, Route de l'aéroport km 4.5  
– B.P. 1088 - 3018 Sfax – Tunisia.  
[jalel.euchi@fsegs.rnu.tn](mailto:jalel.euchi@fsegs.rnu.tn),  
[habib.chabchoub@fsegs.rnu.tn](mailto:habib.chabchoub@fsegs.rnu.tn)

Adnan YASSINE

Laboratoire LMAH, Université du Havre, ISEL - Quai  
Frissard, B.P. 1137- 76063 Le Havre CEDEX–  
FRANCE.  
[adnan.yassine@univ-lehavre.fr](mailto:adnan.yassine@univ-lehavre.fr)

**ABSTRACT:** *In this paper we propose a metaheuristic optimization via memory to solve the Profitable Arc Tour Problem (PATP). The PATP is a variant of the well-known Vehicle Routing Problem in which a set of vehicle tours are constructed. The objective is to find a set of cycles in the vehicle tours that maximize the collection of profits minus travel costs, subject to constraints limiting the length of cycles that profit is available on arcs. Computational experiments show that our algorithm provides good results in terms of quality of solution and completion processing times.*

**KEYWORDS:** *Profitable Arc Tour Problem; Vehicle Routing; Adaptive Memory.*

### 1 INTRODUCTION

The Vehicle Routing Problem (VRP) is a complex combinatorial optimization problem which can be seen as a merge of many problems. The vehicle routing and its generalizations have been widely studied in recent years. The first paper dealing with the Vehicle Routing Problem was published in the late 1950's by Danzig and Ramser (1959). This problem draws a large number of researchers because it is theoretically very interesting. Moreover, the applications of the VRP are numerous. Thus, most companies who must deliver a product to several customers are faced with this problem. The literature of the vehicle routing problem is therefore very large. It includes those of Bodin et al. (1983) and Laporte (1992a, 1992b).

More specifically, several variants of problems appeared in the literature (Corberan et al. (2006), Jozefowicz et al. (2009), Archetti et al. (2005)). A recent paper proposed by Feillet et al. (2005) studied the Profitable Arc Tour Problem (PATP) which is a generalization of the vehicle routing problem where it is not necessary to visit all vertices of the given graph. The profit is associated with each customer and it is known a priori. The PATP can be formulated as a discrete bi-criteria optimization problem where the two goals are maximizing the profit and minimizing the travelling cost. It is also possible to define one of the goals as the objective function and the other one as a satisfiability constraint.

Several extensions are present in the literature: the Selective Travelling Salesman Problem (STSP) (Gendreau et al. (1998)), the Orienteering Problem (OP) (Chao et al. (1996a)) and the Maximum Collection Problem (MCP) (Butt and Ryan (1999)). The objective is the maximization of the collected profit so that the total travelling cost (distance) does not exceed an upper bound.

The other variants, the Prize Collecting Travelling Salesman Problem (PCTSP) (Balas (1989)) is concerned with determining a tour with minimum total travelling cost where the collected profit is greater than lower bound. Feillet et al. (1996) provided an excellent survey of the existing literature on Travelling Salesman Problem with Profits (TSPP). Their survey presents various modelling approaches to TSPP and exact as well as heuristic solution techniques.

The multi-vehicles version of the OP is called the Team Orienteering Problem (TOP) and it is studied by Chao et al. (1996b). Butt and Cavalier (1994) address the Multiple Tour Maximum Collection Problem (MTMCP) in the context of recruiting football players from high schools. They propose a greedy tour construction heuristic to solve this problem. Later, Butt and Ryan (1999) developed an exact algorithm for the MTMCP based on the branch and price solution procedure. Gueguen et al. (2004) also propose an exact algorithm for the elementary shortest path problem with resource constraint.

Due to the complexity of the VRP and PATP, diverse approaches were proposed to solve this problem including Prins (2004), Bérubé et al. (2009), Rochat and Taillard (1995), Rego and Roucairol (1996) and Taillard (1999). Specifically, Feillet et al. (2005) present an exact method with Branch and Cut and with Column Generation to solve the Profitable Arc Tour Problem.

Tabu Search (TS) (Glover (2007)) is a kind of metaheuristic that has been widely used to solve complex combinatorial optimization problems. The success of TS is mainly due to its ability to steer the search process from getting stuck in a local optimum. This is achieved by allowing a move to a neighbouring solution that may result in deterioration in the objective value but simultaneously avoids cycling back through previous moves. The second algorithm is a Variable Neighbourhood

Search (VNS) (Hansen et al. (2008)) which uses a Tabu Search as a local search.

In order to improve the solution, both TS and VNS are embedded in the Adaptive Memory procedure. The metaheuristic proposed here for the PATP can be roughly described into four steps: Initialization, generation of an initial solution, solution improvements and updating the Adaptive Memory (AM).

The organization of the paper is presented as follows: In section 2 we introduce the Profitable Arc Tour Problem. The main paradigm of TS and VNS metaheuristics is stated in section 3. In section 4 we discuss our proposed approach. In section 5 an adaptive memory procedure to solve the PATP is described. In section 6 the results of both the TS and VNS are presented and compared. And finally the conclusion is in section 7.

## 2 THE PROFITABLE ARC TOUR PROBLEM

In this paper, we address the Profitable Arc Tour Problem (PATP). The main reason behind this great attention is the abundance of its real life application in logistic distribution and transportation (Archetti et al. (2008)). This problem is defined over a graph where profits and travel costs are associated with the arcs. The objective is to find a set of cycles in the graph that maximizes the collection of profit minus travel costs, subject to constraints limiting the length of cycles.

Let  $G=(V, A)$  be a complete directed graph where  $V=\{1, \dots, n\}$  and  $A=\{(i, j), 1 \leq i, j \leq n\}$  are the sets of vertices and arcs respectively. Let  $c_{ij} \geq 0$  and  $l_{ij} \geq 0$  (asymmetric) be the cost and the length associated with arc  $(i, j)$  respectively. The cost and the length matrices are assumed to satisfy the triangle inequality. Let  $Q \subset A$  be the subset of the arcs of the graph where profit can be collected. More explicitly, a profit value  $g_{ij} \geq 0$  and a maximal integer number of times ( $q_{ij} \in \mathbb{N}$ ) that can be collected are associated with each arc  $(i, j) \in Q$ .

A collection cycle  $r$  is feasible if its length does not exceed the limit  $l_{\max}$ ; i.e.  $r_k$  is feasible if  $\sum_{(i, j) \in A} \delta_{ij}^k l_{ij} \leq l_{\max}$

where  $\delta_{ij}^k = \begin{cases} 1 & \text{if } (i, j) \in \text{cycle } r_k \\ 0 & \text{otherwise} \end{cases}$ . Let  $\Omega$  be the set of

feasible collection cycles having a length not greater than  $l_{\max}$  in graph  $G$ . With each collection cycle  $r_k \in \Omega$ , associate a cost  $c_k$  equal to the sum of costs of its arcs and a profit  $p_k$  equal to the profits collected on its arcs.

More specifically,

$$c_k = \sum_{(i, j) \in A} \delta_{ij}^k c_{ij} \text{ and } p_k = \sum_{(i, j) \in Q} \delta_{ij}^k a_{ij}^k g_{ij} \text{ where}$$

$$a_{ij}^k = \begin{cases} 1 & \text{if collection cycle } k \text{ collects the gain available on arc } (i, j) \in Q \\ 0 & \text{else} \end{cases}$$

It follows that the net profit associated with a feasible collection cycle is equal to  $(p_k - c_k)$ .

Note that in a feasible collection cycle some vertices and arcs can possibly appear several times. Figure 1 displays an instance of the PATP.

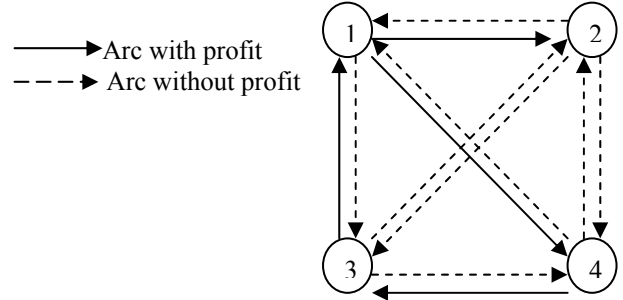


Figure 1. Instance of PATP

The objective of the PATP is to find a set of feasible collection cycles in the graph that maximizes the sum of the net profits collected:

$$\begin{aligned} \text{Maximize} \quad & \sum_{r_k \in \Omega} (p_k - c_k) x_k \\ \text{Subject to} \quad & \sum_{r_k \in \Omega} a_{ij}^k x_k \leq q_{ij} \quad ((i, j) \in Q) \quad (1) \\ & x_k \geq 0, \text{ integer} \quad (r_k \in \Omega) \end{aligned}$$

In this formulation, the variable  $x_k$  associated with the feasible collection cycle  $r_k \in \Omega$  corresponds to the number of times that  $r_k$  is used. Furthermore each constraint (1) limits to  $q_{ij}$  the number of times that the profit can be collected on arc  $(i, j) \in Q$ .

## 3 METAHEURISTICS OPTIMIZATION

Tabu Search is a metaheuristic that has a huge number of applications (Glover (2006)). It explores different kinds of memories in the search such as recency based (short-term), frequency based, long-term memories etc. Usually it uses one neighbourhood structure and, with respect to that structure, performs descent and ascent moves building a trajectory.

The second algorithm is a Variable Neighborhood Search (Hansen et al. (2008)) which uses TS as a local search. Hansen and Mladenovic (1997) proposed a new solution technique called VNS. The main idea of this new method is to use various neighbourhood structures during the search.

## 4 PROPOSED APPROACH

Our proposed solution algorithm is based on the Adaptive Memory Procedure (AMP) (Golden et al. (1997)) to solve the Profitable Arc Tour Problem. The AMP was

first proposed by Rochat and Taillard (1995) as an enhancement of TS to solve the Vehicle Routing Problem (VRP). It was motivated by the work of Glover regarding surrogate constraints (Glover (1977)). An important principle behind AMP is that good solutions may be constructed by combining different components of other good solutions. A memory containing components of visited solutions is kept. Periodically, a new solution is constructed using the data in the memory and improved by a local search procedure. The improved solution is then used to update the memory.

A pseudo code of the AMP is given below:

- 1) Initialize the memory  $M$ .
- 2) While a stopping criterion is not met, do:
  - a) Construct a new solution  $S$  combining components of  $M$ .
  - b) Apply a local search procedure to  $S$  (let  $S^*$  be the improved solution) (TS and VNS).
  - c) Update  $M$  using components of  $S^*$ .

We develop two algorithms. The first one is the TS applied in the local search phase of the adaptive memory and the second one is the VNS applied in the same phase.

## 5 ADAPTIVE MEMORY PROCEDURE TO SOLVE THE PATP

In this section, details of the Tabu Search (TSAM) and Variable Neighbourhood Search embedded in Adaptive Memory (VNSAM) procedure are provided including details of the initialization, construction of initial solution, solution improvements and updating the AMP steps.

### 5.1 Initialization: (Step 1 in the Adaptive Memory)

To begin with, a certain amount of storage space (Adaptive Memory) is allocated within the AMP. A set of cycles is generated by the nearest neighbour method and is stored in the adaptive memory. We are going to skew the selection in favour of the more profitable arcs. For each arc  $(i, j)$  in  $Q$ , let's define an advantage to insert the arc  $(i, j)$

$$av_{ij} = \begin{cases} \frac{(g_{ij} - c_{ij})q_{ij}}{l_{ij}} & \text{if } q_{ij} > 0 \\ \frac{1}{l_{ij}} & \text{otherwise} \end{cases}$$

To choose an arc of a set  $\bar{Q} \subseteq Q$ , let define  $av_{\bar{Q}} = \sum_{(i,j) \in \bar{Q}} av_{ij}$  and consider that the arcs in  $\bar{Q}$  are in any order:  $(i_1, j_1), (i_2, j_2), \dots, (i_{|\bar{Q}|}, j_{|\bar{Q}|})$ .

To choose an arc  $(i, j)$  in  $\bar{Q}$ , we proceed as follows:

- i. Choose randomly a number  $\alpha \in [0, av_{\bar{Q}}]$

- ii. Let  $\tau$  be the smallest indication as  $\sum_{t=1}^{\tau} av_{i_t, j_t} \geq \alpha$
- iii. Then an arc selected is  $(i_{\tau}, j_{\tau})$ .

The procedure to generate  $N$  circuits of the adaptive memory is summed up as follows:

**Step1:** create  $m$  vehicle tours containing only an arc  $(0, 0)$ . The

set  $M = \{(i, j) : l_{0i} + l_{ij} + l_{j0} \leq l_{\max}\}$  is the set of constructed solutions. Let  $k=1$  and move to step 2.

**Step2:** if  $\tau_k$  contains only an arc  $(0, 0)$ , select with roulette wheel an arc  $(i, j) \in M$  and insert it

$$\text{in } r_k. \text{ Let } \begin{cases} \bar{Q} = Q - (i, j) \\ l_k = l_{0i} + l_{ij} + l_{j0} \end{cases}$$

Continue the construction of the circuit  $r_k$  as follows:

1. Choose with roulette wheel an arc  $(i, j) \in \bar{Q}$

2. Select or insert an arc  $(i, j)$  and two vertices  $p$  and  $q$  in  $r_k$  so that the evaluation function

$$\{(av_{pi} + av_{ij} + av_{jq} - av_{pq}) : l_k + l_{pi} + l_{ij} + l_{jq} - l_{pq} \leq l_{\max}\}$$

will be maximal. If such  $(i, j)$  is not found (because  $l_k + l_{pi} + l_{ij} + l_{jq} - l_{pq} \geq l_{\max}$ ), go to step 3.

3. Insert  $(i, j)$  between  $p$  and  $q$  and adjust the length  $l_k$  of circuit:  $r_k : l_k := l_k + l_{pi} + l_{ij} + l_{jq} - l_{pq}$ .

4. Let  $\bar{Q} = Q - (i, j)$

5. if  $\bar{Q} \neq \phi$ , repeat step 1

**Step3:** Let  $k = k + 1$ . If  $k \leq N$ , go to step 2.

### 5.2 Construction of initial solution: (Step 2 in the Adaptive Memory)

The metaheuristic algorithm starts from an initial solution  $s$  constructed in the AMP. Generate an order  $\{r_{k_1}, r_{k_2}, \dots, r_{k_N}\}$  in which the circuits of the AM are considered. More the profit of a circuit is raised more it is likely to be among the first circuits in the order. Let's consider the circuits successively in this order. Let's denote by  $\varphi_k$  the number of profits  $g_{ij}$  that have been collected by already selected circuits. Then, if the residual profit  $\bar{\pi}_k = \sum_{(i,j) \in Q} \delta_{ij}^k a_{ij}^k g_{ij} - \sum_{(i,j) \in A} \delta_{ij}^k c_{ij} > 0$  of the circuit  $r_k$  appears a number of times  $x_k > 0$  in the solution  $s$ .  $x_k = \min_{(i,j) \in r_k} \{(q_{ij} - \varphi_k) : \varphi_k < x_k\}$ .

- If  $x_k > 0$ , then it is necessary to adjust the number of times where the profit  $g_{ij}$  is already collected

on the arcs  $(i, j)$  as follows:  $\forall (i, j) \in r_k$  as  $\varphi_k < x_k$  then  $\varphi_k$  is adjusted as follows:  $\varphi_k := \varphi_k + x_k$ .

After that consider all the circuits in the AM:

- If  $\varphi_k = q_{ij} \forall (i, j)$ , then we have an initial solution and have to pass to the following step to improve this solution.
- If  $\varphi_k < q_{ij}$  for at least one arc  $(i, j)$ , then we continue the generation of the initial solution as follows:

1. Generate new circuits from  $r_k \in K$  not being in AMP which uses some arcs  $(i, j)$  having  $(\varphi_k < q_{ij})$  with the residual profit  $\bar{\pi}_k = \sum_{(i,j) \in Q} \delta_{ij}^k a_{ij}^k g_{ij} - \sum_{(i,j) \in A} \delta_{ij}^k c_{ij}$   $> 0$ . To generate a circuit we use step 2 of the procedure of initialization.

2. It is necessary to redefine  $av_{ij}$  as follows:

$$\bar{av}_{ij} = \begin{cases} \frac{(g_{ij} - c_{ij})(q_{ij} - \varphi_k)}{l_{ij}} & \text{if } 0 < q_{ij} < \varphi_k \\ \frac{1}{l_{ij}} & \text{otherwise} \end{cases}$$

Also, to initialize the arc  $r_k$  we have to choose from within this set:  $\bar{M} = \{(i, j) : l_{0i} + l_{ij} + l_{j0} \leq l_{\max} ; 0 < q_{ij} < \varphi_k\}$ . The circuit  $r_k \in K$  appears to a number of times  $x_k > 0$  in the solution where  $x_k = \min_{(i,j) \in r_k} \{(q_{ij} - \varphi_k) : \varphi_k < x_k\}$ .

When  $\varphi_k = q_{ij} \forall (i, j)$  becomes true or when there does not exist any circuits able to use some arcs as  $\varphi_k < q_{ij}$  and have a positive residual profit, the process does not pass to the following step to improve this solution  $s$ .

### 5.3 Solution improvements: (Step 3 in the Adaptive Memory)

The quality of feasible PATP solution is dependent only on the profit of its constituent vehicle tours. However, improving arcs sequences on solution tours in a way that the cost is minimum is crucial for achieving high quality solutions. A shorter solution tour is more likely to have more arcs included in successive iterations, and therefore, a higher potential profit at further iteration (i.e.; which arcs have been included in the tours?). Thus, for each iteration, a series of tour improvement procedures that consider tour duration are selected. These procedures include exchanging arcs between tours and randomly removing and re-inserting arcs within a tour. In this phase we have developed two algorithms to solve the Profitable Arc Tour Problem.

#### 5.3.1. Tabu search algorithm

TS have become the focus of numerous comparative studies and practical applications in recent years (Cor-

deau et al. (2003)). Tabu search procedures exploit the short term memory, i.e. the tabu list which keeps track of the recently visited solution or its attributes. A move to a neighbouring solution is allowed if the neighbouring solution is neither contained in the tabu list nor possesses an identical attribute (e.g. objective value) to a solution in such list. However, a move to a neighbouring solution could be basically selected on some aspiration criterion even if it is prohibited by the tabu list. For example, in most tabu search applications, a particular move may be permitted even if it (or its attribute) is contained in the tabu list as long as such a move will result in a solution that is superior to the best solution obtained so far (Figure 2).

#### Initial solution

The TS starts from an initial solution  $s$  constructed with the nearest neighbourhood method where customers are placed in an array sorted in the increasing order of profit. In this method, the customer with the biggest profit is appended to a route. When the next to-be inserted customer's distance exceeds the length of cycles on the current route, a new route is initiated.

#### Neighbourhood's structures

The TS algorithm that we have implemented uses two structures of neighbourhoods:

- Permutation-neighbourhood:

Let  $c$  and  $c'$  be two nodes on two different routes. A permutation-move consists in replacing  $r_c(s)$  and  $r_{c'}(s)$  by  $(r_c(s) - c) + c'$  and  $(r_{c'}(s) - c') + c$ , respectively.

- 2-move: In a 2-move, node  $c$  is moved from its route to a route  $r \neq r_c(s)$ . Route  $r$  can be an empty route. Hence,  $r_c(s)$  and  $r$  are replaced by  $r_c(s) - c$  and  $r + c$ , respectively.

A conventional Tabu List (TL) contains pairs  $(c, r)$  with the condition that it is forbidden to move customer  $c$  to circuit  $r$ . A move  $(c, r)$  is considered as tabu if  $(c, r) \in TL$ . The TS is stopped where  $\theta_{\max}$  iterations have been performed without improving  $s^*$ .

#### 5.3.2. Variable Neighbourhood Search

The VNS is a recent metaheuristic for solving combinatorial and global optimization problems. It is a simple yet very effective metaheuristic that has shown to be very robust on variety practical NP-hard problems. Its basic idea is a systematic change of neighborhood within a local search.

Here, several neighborhoods' structures are used instead of a simple one as it generally changes the case in many local search implementations. Furthermore, the systematic change of neighbourhoods is applied during

both a descent phase and an exploration phase, allowing getting out of local optima.

In the initialization phase, a set of  $k_{\max}$  (a parameter) neighborhood is reselected ( $N_k, k = 1, \dots, k_{\max}$ ), a stopping condition is determined and an initial local solution is found. Then the main loop of the method has the steps described in figure 3. Below, the implementation of each part of the VNS to solve the PATP is described. The VNS algorithm starts with the initial solution obtained with the nearest neighbourhood, and then shaking is performed. Shaking flips  $k$  different randomly selected variables of the currently best solution. The swap neighbourhood local search heuristic is used to improve the solution. We use a simple swap, let  $v_i$  and  $v_j$  be two nodes on two different routes. A permutation-move consists of replacing all possible swaps of pairs of different positions. If the swap moves improve the solution, we apply the insertion local search. At each iteration the heuristic computes for each couple  $[v_i, v_j]$  not yet selecting the best position where to insert it into the tour. It then inserts the couple which insertion minimizes the length of the current tour and maximizes the profit. We repeat this procedure until no improvement in the solution is performed.

After the shaking and the local search procedure, the solution thus obtained has to be compared to the incumbent solution to be able to decide whether or not to accept it. The evaluation function for solutions has to be specified, particularly if infeasible solutions are allowed. To improve the solution generated for each algorithm, we use two improvement procedures:

#### *Exchanging of arcs between two circuits*

For randomly selected circuit  $r_1$  and  $r_2$  from the current solution  $s$ , these improvement procedures make one-arc exchange between circuit  $r_1$  and  $r_2$  (Brandão (2009)). Starting with the first arc on  $r_1$ . We scan from the first to the last arc on  $r_2$  to examine if an exchange of the first arc on  $r_1$  with the current arc on  $r_2$  makes the total cost of the two circuits shorter. The exchange is made immediately when such arc on  $r_2$  is found. Then, the procedure is repeated with the second arc on  $r_1$  and scanning arc on  $r_2$ . This procedure stops when all one-arc exchanges between  $r_1$  and  $r_2$  leading to improve tour duration have been performed.

#### *A random arc-insertion procedure*

For selected vehicle cycle  $\tau$  of duration  $D(\tau)$ , this semi-greedy random insertion approach randomly removes a subset of arcs from the cycle and re-inserts them in the resulting partial cycle in a greedy way.

### 5.4 Updating the adaptive memory: (Step 4 in the adaptive memory)

Let's consider all circuits  $r_k \in K$  available in the Adaptive Memory or which make part of the solution. Let's choose the  $N$  circuits having the biggest profit  $\pi_k = (p_k - c_k)$  to form the new adaptive memory. Finally let's take the phase of construction of an initial solution of the problem with this new memory.

The steps of proposed metaheuristic are summarized as follows:

**Step1:** Generate  $m$  cycles derived from different solution nearest neighbor methods.

Start with  $AMP = \phi$  (adaptive memory procedure).

**Step2:** While a stopping criterion is not met, do:

$AMP' = AMP; s = \phi$

Repeat, while  $AMP' = \phi$

Choose randomly a cycle  $r \in AMP'$

Let  $s' = s' \cup \{r\}$

For each cycle  $r \in AMP'$ , where  $r \cap r' \neq \phi$

Let  $AMP = AMP \cup \{r\}$

**Step3:** Improve the new constructed solution.

For each cycle  $r$  in  $s$ , let  $AMP = AMP \cup \{r\}$

Find a solution  $s^*$  by considering the continuous cycles in  $AMP$

1- Apply Tabu search algorithm (only for the first algorithm).

2- Apply Variable Neighbourhood search algorithm (only for the second algorithm).

Exchanging of arcs between two tours

A random arc-insertion procedure

**Step4:** update the AMP by inserting the new constructed cycles and removing cycles (if necessary) which belong to the worst solutions.

## 6 EXPERIMENTAL RESULTS

In this section the hybrid metaheuristics implementation proposed above will be analyzed when applied on a wide set of instances taken from the literature.

### 6.1 Implementation and instances

To test our methods we used the same set of test cases as Feillet et al. (2005). These were obtained with the help of Mr. Feillet who sent us these instances. We report our results in the same framework he used to make comparisons easier.

We consider two sets of instances to evaluate the performance of hybrid metaheuristic. The first group is composed of 140 instances studied by Feillet et al. (2005) with different number of bonus and the second group is made up of the set of problems randomly generated by using the combinations of the instances used by

Feillet et al. (2005) and employed with a log-normal distribution).

In the table 1 we gives only the average results for different types of benchmarks for example for the Pb\_01 it contains a several types of instances with the same number of vehicles but different number of bonus.

The algorithm described here has been implemented in C++ using Visual Studio C++ 6.0. Experiments are performed on a PC Pentium 4, 3.2 GHz with 512MB of RAM.

### 6.2. Parameter settings

The proposed metaheuristic employs a set of parameters whose values need to be set before the algorithm is run. These parameters include the number of tabu iteration  $N_{max}$ , tour improvement frequency  $\chi$ , tour selection parameter T, neighborhood size  $\beta$  and maximum non-improvement iterations  $\theta$ . These parameters were determined as the basis of a number of preliminary runs. The values of these parameters are defined as follows:  $N_{max} = 200$ ;  $\chi = 6$ ;  $T = 30$ ;  $\beta = 200$ ,  $\theta = 100$ .

### 6.3 Evaluation method

In order to verify the effectiveness and efficiency of the proposed approach, we compare the results generated with TS Adaptive Memory (TSAM) with those generated with VNS Adaptive Memory (VNSAM).

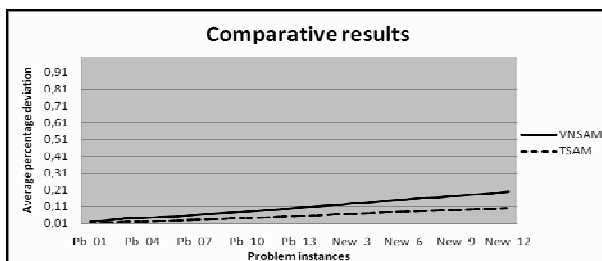


Figure 4. Relative gap

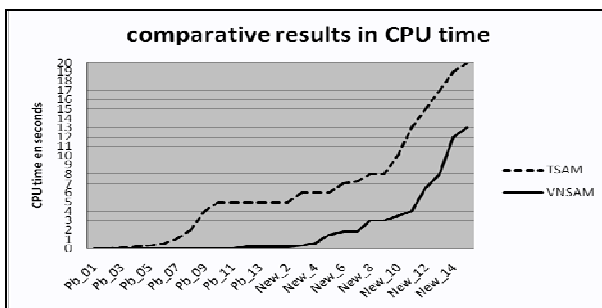


Figure 5. CPU time

Figure 4 present the comparison of results between the TSAM and VNSAM at the level of quality of solution. According to the figures we notice that the average percentage deviation tends towards zero between the two

algorithms. In fact, the VNSAM finds are slightly fitter solutions than the TSAM. In figure 5, we observed the comparison of the computation time of the proposed approach reveals that the VNSAM is computationally efficient than the TSAM.

Table 1 summarizes the quality of solutions obtained by the two algorithms. As we can see from this table that the VNSAM is very robust and clearly outperforms the TSAM. As we can see in table 1, for all instances, a new best solution was found compared with solution produced by TSAM. We observe that for small sized problems, both TSAM and VNSAM could obtain an optimal solution. In fact, it gives the CPU time in seconds which is needed by TSAM and VNSAM to find the best solution. In all instances, we observe that the algorithm provide a good computational time.

Table 2 describes the quality of the results over 10 runs of the TSAM and VNSAM on each of instances with the parameters described above with the exact methods provided by Feillet et al. (2005) in the form of percentage deviation. It is shown, from this table, that the percentage deviation between the VNSAM and the best known solutions is very slighter.

Table 3 gives the comparison results of the TSAM, VNSAM, CAP and SARPP in the term of processing time. We can conclude that our proposed algorithm find an optimal solution very quickly.

From the experiments carried out here we can bring to a close that the VNSAM algorithm obtained good results although it was the best performer on the data sets. The results of this research show that the proposed metaheuristic is a very effective tool for finding good solutions for the PATP.

## 7 CONCLUSION

We have discussed a Profitable Arc Tour Problem (PATP) and we have proposed solutions techniques to solve it based on the TS and VNS method together with Adaptive Memory. The algorithm has produced encouraging results. The results of the metaheuristic are compared to the results of exact method provided by Feillet et al. (2005) on each benchmark problems. The results showed that the solution produced by our proposed approach was highly dependent on the choice of the initial solution.

The AMP and its mechanism for updating stored solutions allow a comparatively large pool of good and diversified solutions to be stored and used during the search process, alternating between small and large neighborhood stages during the metaheuristic course.

This paper has two main contributions. First, it presents the first applications of TS and VNS embedded in Adaptive Memory to solve the PATP. Second, the

## ACKNOWLEDGMENTS

We would like to thank Dr. Feillet for his helpful to make the comparison easier.

## REFERENCES

- C. Archetti, M.W.P. Savelsbergh, M.G. Speranza (2008). To split or not to split: That is the question. *Transportation Research Part E* 44: 114–123.
- C. Archetti, D. feillet, A. Hertz, MG. Speranza. The capacitated Arc Routing Problem with Profits. *Les cahiers du GERAD* ISSN:0711-2440 G-2009-01 january 2009.
- E. Balas (1989). The prize collecting travelling salesman problem. *Networks* 19: 621–636
- JF. Bérubé, M. Gendreau, JY. Potvin (2009). An exact  $\epsilon$ -constraint method for bi-objective combinatorial optimization problems: Application to the Travelling Salesman Problem with Profits. *European Journal of Operational Research* 194: 39–50
- LD Bodin, B.L. Golden, A.A. Assad, M.O. Ball (1983). Routing and scheduling of Vehicles and crews. The state of the Art. *Computers and Operations Research* 10: 69–211.
- J. Brandão (2009). A deterministic tabu search algorithm for the fleet size and mix vehicle routing problem. *European Journal of Operational Research* 195: 716–728
- SE. Butt, D.M. Ryan (1999). An optimal solution procedure for the multiple tour maximum collection problem using column generation. *Computers and Operations Research* 26 (4): 427–441
- SE. Butt, T.M. Cavalier (1994). A heuristic for the multiple tour maximum collection problem. *Computers and Operations Research* 21: 101–111.
- IM. Chao, B. Golden, E.A. Wasil (1996). A fast and effective heuristic for the orienteering problem. *European Journal of Operational Research* 88: 475–489
- IM Chao, B. Golden, E.A. Wasil (1996). The team orienteering problem. *European Journal of Operational Research* 88: 464–474
- A. Corberán, E. Mota, J.M. Sanchis (2006). A comparison of two different formulations for arc routing problems on mixed graphs. *Computers & Operations Research* 33: 3384–3402.
- J.F. Cordeau, G. Laporte (2003). A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B: Methodological* 37 (6): 579–594.
- D. Feillet, P. Dejax, M. Gendreau (2005). The Profitable Arc Tour Problem: Solution with a Branch-and-Price Algorithm. *Transportation Science* 39(4): 539–552.
- M. Gendreau, G. Laporte, F. Semet (1998). A tabu search heuristic for the undirected selective travelling salesman problem. *European Journal of Operational Research* 106 (2-3): 539–545
- F. Glover (2007). Tabu search—Uncharted domains. *Annals of Operations Research* 149: 89–98.
- F. Glover (2006). Parametric tabu-search for mixed integer programs. *Computers & Operations Research* 33: 2449–2494
- F. Glover (1977). Heuristics for Integer Programming Using Surrogate Constraints. *Decision Sciences* 8(1): 156–166.
- B.L. Golden, G. Laporte, E.D. Taillard (1997). An adaptive memory heuristic for a class of vehicle routing problems with minmax objective. *Computers & Operations Research* 24(5): 445–452.
- C. Gueguen, D. Feillet, P. Dejax, M. Gendreau (2004). An exact algorithm for the elementary shortest path problem with resource constraint: Application to some vehicle routing problems. *Network* 44: 216–219.
- P. Hansen, N. Mladenovic, J.A. Moreno-Pérez (2008). Variable neighborhood search. *European Journal of Operational Research* 191(3): 593–595.
- P. Hansen, N. Mladenovic (2001). Variable neighbourhood search: Principles and applications. *European Journal of Operational Research* 130: 449–467
- N. Jozefowicz, F. Semet, E.G. Talbi (2009). An evolutionary algorithm for the vehicle routing problem with route balancing. *European Journal of Operational Research* 195: 761–769
- G. Laporte (1992). The Vehicle Routing Problem: An overview of exact and approximate algorithms. *European Journal of Operational Research* 59 (3): 345–358.
- G. Laporte (1992). The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research* 59 (2): 291–247
- N. Mladenovic, P. Hansen (1997). Variable neighbourhood search. *Computers and Operations Research* 24: 1097–1100
- C. Prins (2004). A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers and Operations Research* 31: 1985–2002
- C. Rego, C. Roucairol (1996). A parallel tabu search algorithm using ejection chains for the vehicle routing problem, In: Osma, I.H., Kelly J. (Eds.): *Meta-Heuristics: Theory and Applications*.
- Y. Rochat, E.D. Taillard. 1995. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of heuristics* 1: 147–167
- E.D. Taillard (1999). A heuristic column generation method for the heterogeneous fleet VRP. *RAIRO* 33 (1): 1–14.

```

Begin Tabu Search
  x: initial solution
   $x_{best} = 0$ ; /* is the best solution obtained by Tabu search */
  TL := 0; iter := 0;  $\theta := 0$ ;
  /* apply nearest neighbour heuristic */
   $x' = \text{Nearest\_Neighbour}(x)$ ;
  While ( $\theta < 100$ ) do
    update TL;
    /* use the swap local search procedure */
     $x'' = \text{swap}(x')$ ;
    /* use the insertion local search when an improvement is obtained */
     $x' = \text{insert}(x'')$ ;
    update TL;
    if ( $f(x') > f(x)$ ) then
       $x' := x$ ;
      iter ++; noimp ++;
    end if
  until ( $\theta = 100$ )
  end While
  update TL;
   $x_{best} := x'$ ;
end Tabu Search

```

Figure 2. Pseudo code of Tabu Search algorithm

```

procedure VNS(x)
  x: initial solution
  k: Neighborhood order
  N: number of iteration
   $N_{max}$ : number of iteration without improvement
  begin
    /* select with nearest neighbour a solution in the  $k^{th}$  neighborhood structure  $N_k(x)$  */
     $x' = \text{Nearest\_Neighbour}(x)$ ;
    k = 1;
  do
    do
      /* use the swap local search procedure */
       $x'' = \text{swap local search}(x')$ ;
      /* use the insertion local search when an improvement is obtained */
       $x' = \text{insertion local search}(x'')$ ;
    until (no possible improvement)
    if ( $f(x') > f(x)$ ) then // check the improvement of best solution
       $x = x'$ ;
      k = 1;
    else
      k = k + 1;
    endif
  until ( $N = N_{max}$ )
  return x;
end VNS

```

Figure 3. Pseudo code of VNS algorithm

**Table1** Computational results for TSAM and VNSAM algorithm

instances	n	TSAM			VNSAM		
		nb	Average	CPU time	nb	Average	CPU time
Pb_01	5	5	47,048.0	0.00	5	47,048.0	0.00
Pb_02	10	8	236,952.9	0.00	8	236,952.9	0.00
Pb_03	15	12	573,179.3	0.12	12	573,179.3	0.02
Pb_04	20	16	1,123,103.4	0.22	16	1,123,103.4	0.02
Pb_05	25	22	1,848,166.4	0.35	22	1,848,166.4	0.02
Pb_06	30	28	2,635,932.5	0.5	28	2,635,932.5	0.03
Pb_07	35	29	3,727,594.2	1.03	29	3,727,594.2	0.03
Pb_08	40	35	4,972,050.5	2.0	35	4,972,050.5	0.03
Pb_09	45	39	5,989,998.9	4.04	39	5,989,998.9	0.04
Pb_10	50	42	7,626,723.5	5.01	42	7,629,276.1	0.08
Pb_11	55	47	9,422,559.7	5.15	50	9,427,041.2	0.09
Pb_12	60	51	10,945,175.1	5.20	51	10,955,146.4	0.17
Pb_13	65	59	12,801,622.0	5.20	60	12,818,514.0	0.22
New_1	70	63	15,822,174.2	5.23	63	15,839,301.0	0.22
New_2	75	68	18,875,121.0	5.28	70	18,886,881.3	0.23
New_3	80	72	25,904,370.1	6.03	73	25,915,291.4	0.31
New_4	85	65	31,874,290.9	6.30	70	31,915,820.1	0.60
New_5	90	77	33,627,478.7	6.42	81	33,683,881.2	1.32
New_6	95	84	36,745,257.1	6.92	88	36,780,237.6	1.83
New_7	100	84	39,565,635.3	7.19	90	39,629,257.6	1.86
New_8	105	92	42,785,977.2	8.28	92	42,832,748.0	3.00
New_9	110	94	45,951,751.0	8.32	94	45,978,677.0	3.05
New_10	115	94	47,972,567.9	10.03	94	48,002,129.7	3.65
New_11	120	104	54,915,842.1	13.04	106	54,983,221.5	4
New_12	125	110	66,097,470.2	15.79	112	66,112,237.1	6.69
New_13	130	112	74,183,188.4	17.16	121	74,233,591.9	8.05
New_14	135	115	77,374,291.8	19.20	131	77,395,274.1	12.1
New_15	140	122	80,867,164.1	20.05	133	80,910,737.6	13.12

**n:** Number of vertices.

**nb:** Total number of inserted vertices.

**Cost:** cost of the solution

**CPU time:** Total CPU time in second

**Table 2** Percent deviation results for the TSAM, VNSAM on the large test problems

instances	TSAM	VNSAM	Best known solution	% deviation	
	Average	Average	Average	$\Delta_{TSAM}$	$\Delta_{VNSAM}$
Pb_01	47,048.0	47,048.0	47,048.0	0.00	0.00
Pb_02	236,952.9	236,952.9	236,952.9	0.00	0.00
Pb_03	573,179.3	573,179.3	573,179.3	0.00	0.00
Pb_04	1,123,103.4	1,123,103.4	1,123,103.4	0.00	0.00
Pb_05	1,848,166.4	1,848,166.4	1,848,166.4	0.00	0.00
Pb_06	2,635,932.5	2,635,932.5	2,635,932.5	0.00	0.00
Pb_07	3,727,594.2	3,727,594.2	3,727,594.2	0.00	0.00
Pb_08	4,972,050.5	4,972,050.5	4,972,050.5	0.00	0.00
Pb_09	5,989,998.9	5,989,998.9	5,989,998.9	0.00	0.00
Pb_10	7,626,723.5	7,629,276.1	7,638,764.7	0.15	0.12
Pb_11	9,422,559.7	9,427,041.2	9,438,438.0	0.16	0.12
Pb_12	10,945,175.1	10,955,146.4	10,970,388.7	0.23	0.14
Pb_13	12,801,622.0	12,818,514.0	12,823,662.9	0.17	0.04
New_1	15,822,174.2	15,839,301.0	15,839,301.0	0.10	0.00
New_2	18,875,121.0	18,886,881.3	18,886,881.3	0.06	0.00
New_3	25,904,370.1	25,915,291.4	25,915,291.4	0.04	0.00
New_4	31,874,290.9	31,915,820.1	31,915,820.1	0.13	0.00
New_5	33,627,478.7	33,683,881.2	33,683,881.2	0.16	0.00
New_6	36,745,257.1	36,780,237.6	36,780,237.6	0.09	0.00
New_7	39,565,635.3	39,629,257.6	39,629,257.6	0.16	0.00
New_8	42,785,977.2	42,832,748.0	42,832,748.0	0.10	0.00
New_9	45,951,751.0	45,978,677.0	45,978,677.0	0.05	0.00
New_10	47,972,567.9	48,002,129.7	48,002,129.7	0.06	0.00
New_11	54,915,842.1	54,983,221.5	54,983,221.5	0.04	0.00
New_12	66,097,470.2	66,112,237.1	66,112,237.1	0.02	0.00
New_13	74,183,188.4	74,233,591.9	74,233,591.9	0.06	0.00
New_14	77,374,291.8	77,395,274.1	77,395,274.1	0.02	0.00
New_15	80,867,164.1	80,910,737.6	80,910,737.6	0.05	0.00
<b>Average</b>				0.103	0.015

**TSAM:** Tabu Search Adaptive Memory

**VNSAM:** Variable Neighbourhood Search Adaptive Memory.

**Table 3** Comparison processing time results

	<b>SARPP</b>	<b>CAP</b>	<b>TSAM</b>	<b>VNSAM</b>
<b>Instances</b>	<b>CPU</b>	<b>CPU</b>	<b>CPU</b>	<b>CPU</b>
Pb_01	0.24	0.22	0.00	0.00
Pb_02	0.87	0.84	0.00	0.00
Pb_03	3.79	2.17	0.12	0.02
Pb_04	17.15	5.96	0.22	0.02
Pb_05	1256.43	14.91	0.35	0.02
Pb_06	-	34.73	0.5	0.03
Pb_07	-	71.89	1.03	0.03
Pb_08	-	122.30	2.0	0.03
Pb_09	-	231.94	4.04	0.04
Pb_10	-	446.54	5.01	0.08
Pb_11	-	876.80	5.15	0.09
Pb_12	-	1914.16	5.20	0.17
Pb_13	-	3609.24	5.20	0.22
New_1	-	-	5.23	0.22
New_2	-	-	5.28	0.23
New_3	-	-	6.03	0.31
New_4	-	-	6.30	0.60
New_5	-	-	6.42	1.32
New_6	-	-	6.92	1.83
New_7	-	-	7.19	1.86
New_8	-	-	8.28	3.00
New_9	-	-	8.32	3.05
New_10	-	-	10.03	3.65
New_11	-	-	13.04	4
New_12	-	-	15.79	6.69
New_13	-	-	17.16	8.05
New_14	-	-	19.20	12.1
New_15	-	-	20.05	13.12

**SARPP:** Solution of an Arc Routing with Profit (Branch and Price)

**CAP:** Column Generation

**TSAM:** Tabu Search Adaptive Memory

**VNSAM:** Variable Neighbourhood Search Adaptive Memory.