This article can be cited as I. P. Solos, I. X. Tassopoulos and G. N. Beligiannis, Optimizing Shift Scheduling for Tank Trucks Using an Effective Stochastic Variable Neighbourhood Approach, International Journal of Artificial Intelligence, vol. 14, no. 1, pp. 1-26, 2016.
Copyright©2016 by CESER Publications

Optimizing Shift Scheduling for Tank Trucks Using an Effective Stochastic Variable Neighbourhood Approach

Ioannis P. Solos, Ioannis X. Tassopoulos and **Grigorios N. Beligiannis**[1]

[1]Department of Business Administration of Food and Agricultural Enterprises,
University of Patras, G. Seferi 2, GR-30100 Agrinio, Greece;
Email: gbeligia@upatras.gr

ABSTRACT

In this contribution we introduce the application of a stochastic variable neighbourhood algorithm in order to solve a problem taken from a real world situation faced by a small oil company. The problem at hand is an optimization problem and relates to shift scheduling of tank trucks. Input data consists of a set of tank trucks with different properties along with a set of drivers having different driving skills. The objective is to establish a one-to-one assignment between the set of available drivers and the set of tank trucks shifts so as all related hard and soft constraints are satisfied and the tank truck shifts are optimized. This specific problem has been already addressed at the literature and some encouraging results have been published (Knust and Schumacher, 2011). Nevertheless, the optimization algorithm proposed in current contribution manages to achieve better solutions for all but one instance among thirty of them in the same reasonable amount of computational time.

Keywords: Heuristic search, stochastic variable neighbourhood, shift scheduling, tank trucks, swap mechanisms.

Mathematics Subject Classification: 68T20, 90C59, 90B36, 90B90, 90B06

Computing Classification System: I.2.8, G.1.6

1. INTRODUCTION

Personnel scheduling problems have been widely studied during the last decades (Van den Bergh et al., 2013). The main reason for this increase in academic attention is that businesses have become more and more service oriented. So, assigning employees to shifts in certain time periods in order to maximize the level of services and minimize respective costs, taking into account their qualifications and preferences under specific regulations, can be very beneficial for many companies and public organizations.

A popular classification of personnel scheduling problems is per application area (Van den Bergh et al., 2013). The main categories reported in the respective literature are:

- Services (general, nurse, other health care, protection/emergency, call center, etc.) (Maenhout and Vanhoucke, 2013; Dietz, 2011; Rasmussen et al., 2012; Tsai and Li, 2009)

- Transportation (general, airline, railway, bus, etc.) (Knust and Schumacher, 2011; Dück et al., 2012; Elizondo et al., 2010; De Matta and Peters, 2009)
- Manufacturing (Kyngäs et al., 2012; Corominas et al., 2012)
- Retail (Zolfaghari et al., 2010; Nissen and Gunther, 2010)
- Military (Safaei et al., 2011; Li and Womer, 2009)
- General (Valls et al., 2009; Naudin et al., 2012)

From the above categories, the transportation personnel scheduling problems maybe constitute the most important category for modern companies. Supply change management in modern companies comprises mainly of their operational activities (Uzar and Catay, 2012). Transportation and inventory management have a decisive influence on the companies' costs and profits (Relvas et al, 2013). So, minimizing the total cost of driver and vehicle scheduling while using all stuff in an efficient way, covering all duties and trying to take into account as many as possible of their preferences is a major concern for any modern company and public organization.

A transportation category problem which has attracted much attention from the academic community in recent years is the truck driver scheduling problem (Knust and Schumacher, 2011; Goel et al., 2012; Goel, 2012; Palmgren et al., 2004; Goel, 2010). This problem, in its general form, is NP-complete. That is, the time required to solve it using any currently known algorithm increases very quickly as the size of the problem grows. As a result, a deterministic algorithm, giving an effective and efficient solution in polynomial time, cannot be found. Next, some very interesting and effective approaches to solve this problem are presented.

Goel et al. (2012) present and study the Australian Truck Driver Scheduling Problem which is the problem of "determining whether a sequence of locations can be visited within given time windows in such a way that driving and working activities of truck drivers comply with Australian Heavy Vehicle Driver Fatigue Law". The authors present an exact method and describe in detail all dominance criteria used in order to reduce computational effort. Moreover, they apply four different heuristic procedures so as to cope with the computationally expensive steps of the exact algorithm. Experimental results show that, for almost all instances, for which a feasible schedule exists, the most effective proposed heuristic manages to find a feasible schedule.

Knust and Schumacher (2011) deal with the shift scheduling problem for a small oil company. As input, all necessary information concerning tank tracks and drivers are given. The authors aim to construct a shift schedule that assigns a feasible driver to every shift of the tank trucks under the following constraints: i) all respective legal and safety restrictions should be satisfied, ii) the total working times of all drivers should be within desired intervals, iii) requested vacation of all drivers should be respected and iv) trucks should be assigned to more favored drivers. A two-phase solution algorithm is proposed in order to solve this problem. Its first phase is based on a mixed integer linear programming formulation while its second phase constitutes an improvement procedure. Experimental results confirm that the two-phase proposed algorithm is able to generate feasible schedules quickly and efficiently.

Goel (2012) presents both a mixed integer programming formulation and an iterative dynamic programming approach in order to solve the Canadian minimum duration truck driver scheduling problem. This problem deals with the minimisation of the duration of truck driver working hours complying with Canadian hours of service regulations. Transport companies in Canada have to make sure that the working hours of all their truck drivers are compliant with Canadian Commercial Vehicle Drivers Hours of Service Regulations. Experimental results denote that the durations of the resulted schedule have been reduced significantly compared to a previously presented approach.

Palmgren et al. (2004) propose a solution for a log-truck scheduling problem which consists of scheduling the transportation of logs between forest areas and wood-mills together with determining the route of all vehicles to satisfy these transportation requests. Their approach is based on column generation and pseudo branch and price algorithm. The main objective is to minimize the total cost of non-productive activities (waiting time of trucks, waiting time of forest log-loaders, empty driven distance of vehicles). The combined scheduling and routing problem is solved by using a constraint programming model while the optimization of deadheads is addressed using an integer programming one.

In (Goel, 2010) a method for constructing driving and working hours' schedule of truck drivers with respect to regulation (EC) No.561/2006 is proposed. As input, a sequence of locations to be visited within specified time windows is given. The presented approach is capable of finding a schedule complying with regulation (EC) No.561/2006, in case such a schedule exists.

In this paper, the problem of scheduling the shifts of tank trucks for a small oil company is investigated (Knust and Schumacher, 2011). This problem which belongs to the wide category of truck driver scheduling problems is also NP-complete in its general form. As inputs, two different sets are given. The first set consists of the tank trucks of the company having different properties. The second set is the set of the company's drivers which also have different skills. Our aim is to establish a one-to-one assignment between the set of available drivers and the set of tank trucks shifts, under the following constraints:

- Legal and safety restrictions should be strictly followed
- Total working time of each driver should be within desired intervals
- Requested vacation of each driver should be respected
- Trucks should be assigned to drivers according to certain preferences.

Any algorithm applied to this problem, in order to be considered as efficient and effective, should:

- Provide the managers of the company with an answer to the critical question of whether the number of current drivers is sufficient in order to cover company's demands of man hours or not.

- Compress the company's costs by assigning drivers to shifts more efficiently (e.g. reducing over-time assignments)
- Reduce the time spent for the construction of the shift schedule.

The proposed algorithm belongs to the wide category of soft computing techniques. Our motivation to use a soft computing algorithm in order to optimize the shifts scheduling of tank trucks for a small oil company comes from the fact that in recent years many soft computing approaches have been applied to different optimization problems with very satisfactory results (David et al., 2013; Khan, 2014; Torres et al., 2013; Valdez et al., 2011; Yazdani et al., 2013; Zăvoianu et al., 2013).

Specifically, the proposed algorithm comprises a stochastic variable neighbourhood search (VNS) approach (Solos et al., 2013) which aims in optimizing the company's shift scheduling. It incorporates a set of nine different swap mechanisms, four of which are innovative, namely Column General Move, Matrix General Move, Swap Sorted Successive Rows and Collapsible Window Stochastic Swap (see Section 3.1), while the remaining five are well established swap mechanisms in the respective literature (Burke et al., 2003; Solos et al., 2013; Lü et al., 2011). In addition, it uses a simple perturbation schema that is triggered in case there is no fitness improvement for a specific number of algorithm's cycles.

Lü et al. (2011) use the curriculum-based timetabling problem in order to perform "an in-depth analysis of neighborhoods relations for local search algorithms". The interested reader can find in (Lü et al., 2011) a well presented theoretical proof of variable neighborhood approaches performance. Out of the nine mutation procedures used by the proposed VNS approach only four of them follow the neighborhood structures, namely *Neighborhood N_1* and *Neighborhood N_2*, presented in Burke et al. (2003). More specifically, mutation procedures Column Simple Move and Matrix Simple Move (see Section 3.1) follow neighborhood structure N_1, while mutation procedures Column Non Empty Move and Matrix Non Empty Move (see Section 3.1) follow neighborhood structure N_2.

Solos et al. (2013) apply a two-phase stochastic VNS approach in order to solve the nurse rostering problem. Experimental results presented, show that the proposed method achieves to solve optimally seven different well known nurse rostering cases reported in the respective literature. The VNS approach presented in current contribution uses only one mutation procedure out of the three utilized by Solos et al. (2013). The common mutation procedure is Swap Rows Randomly (see Section 3.1).

Burke et al. (2003) present a VNS approach attempting to access hidden parts of the solution space in order to solve effectively nurse rostering problems within a short calculation time. Experimental results demonstrate that the proposed approach "allows for a better exploration of the search space, by combining shortsighted neighborhoods and very greedy ones". The proposed VNS approach uses only three mutation procedures from those utilized by Burke et al. (2003). Procedures Column Simple Move and Matrix Simple Move (see Section 3.1) follow the *single shift-day* neighborhood, while procedure Swap Rows Randomly (see Section 3.1) follows the *greedy shuffling* neighborhood presented by Burke et al. (2003). Moreover, the perturbation schema used is similar to the *shake a shift* procedure proposed in the same contribution.

Burke et al. (2007) introduce the idea of *block neighborhood* and apply it in order to solve efficiently the nurse rostering problem. Procedure Collapsible Window Stochastic Swap (see Section 3.1) used in current contribution constitutes a variation of this approach. Specifically, it has two major differences. The first one concerns the block length within which swaps are performed between two rows (drivers). In Burke et al. (2007) the block length is fixed while in our approach it varies from 1 to the length of the planning horizon. The second difference concerns the number of performed swaps. In Burke et al. (2007) all available swaps are executed. So, time consuming exhaustive search is performed. In our approach, we introduce a skip swap mechanism by which some swaps are omitted based on a probability value *p* (see Section 3.1). The interested reader can find in Burke et al. (2003) and Burke et al. (2007) a well presented theoretical proof of variable neighborhood approaches performance.

Concluding, the application of nine different swap operators and the perturbation schema enriches the search capability of the proposed algorithm and helps it reach better shift schedules. Each different swap operator assists the proposed VNS approach to search in a different neighbourhood of the problem’s search space.Two of them are selected alternatively, with a dynamically computed probability, while the rest of these swap operators are executed consecutively. The way these nine swap operators, as well the perturbation schema, are combined and applied, in order to solve the tank track shift scheduling problem, comprise the main algorithmic innovation of the proposed approach. This is the main difference and added value of the proposed algorithm with respect to the contributions presented above (Burke et al., 2003; Solos et al., 2013; Lü et al., 2011).

Moreover, according to our knowledge, it is the first time that a stochastic VNS approach is applied in order to solve the (tank) truck scheduling problem. This is the main difference and added value of the proposed algorithm with respect to the contributions presented above (Knust and Schumacher, 2011; Goel et al., 2012; Goel, 2012; Palmgren, 2004; Goel, 2010). Analytical details about the general structure of the main algorithm and the swap and perturbation schemas are presented in Section 3. Experimental results, presented in Section 4, illustrate that the proposed VNS approach outperforms an existing method, which is applied to the same problem instances, producing shift schedules that are significantly optimized than the ones constructed by Knust and Schumacher (2011).

This paper is organized as follows. Section 2 presents the problem specification and the constraints used. Section 3 describes the proposed algorithm. Section 4 assesses and compares the performance of the proposed algorithm to that of existing approach, while Section 5 presents the discussion. Finally, Section 6 provides summary and future extensions.

2. PROBLEM SPECIFICATION

The tank track scheduling problem considered in this contribution is the one introduced by Knust and Schumacher (2011) and consists of assigning shifts to tank trucks and drivers in accordance with a given set of constraints. Two types of constraints are defined, namely, hard and soft constraints. Hard constraints are the ones that have to be strictly satisfied under any circumstances, while soft constraints are the ones which should not necessarily be satisfied but whose violations should be desirably minimized. A schedule satisfying all hard constraints is called a feasible schedule. A single violation of a hard constraint renders the solution infeasible. The number of soft constraints satisfied by a feasible shift schedule characterizes its quality. The test instances used in order to check the performance of the proposed approach are the exact ones used by Knust and Schumacher (2011), where the interested reader can find a detailed description of them. A brief presentation of all hard and soft constraints used, as well as a mathematical formulation of the tank truck scheduling problem faced in current contribution, is provided in the following subsections.

2.1. Hard and soft constraints

The following hard and soft constraints are taken into consideration:

- Hard constraints
  H1: Coverage requirement – all shift type demands during the planning period must be met;
  H2: Single shift per day – a driver cannot work more than one shift per day;
  H3: Drivers' availability – a driver is assigned a shift on a specific day only if he is available on that day;
  H4: Drivers' max working time per week – it is prohibited for any driver to work more than 55 hours per week;
  H5: No early shift after a late shift – it is prohibited for any driver to have an early shift directly after a late shift;
  H6: No early and late shifts in same week – it is prohibited for any driver to have both early and late shifts in the same week;
  H7: No same shift type for specific drivers in consecutive weeks – for some specific drivers it is prohibited to have the same type of shifts in consecutive weeks;
  H8: Compact working/non-working periods for permanent drivers – all permanent drivers must have compact working/non-working periods:
    - ➢ it is prohibited to work only one day per week;
    - ➢ if there are some free days in a week which are not specified as vacation days, these days must be consecutive days before or after Sunday;
- Soft constraints
  S1: Desired total working time interval – each driver wants to have all his shifts inside his desired total working time interval;
  S2: Preferred drivers assignment – trucks should be assigned to more preferred drivers;
  S3: Different trucks assignment – drivers should not be assigned to a large number of different trucks per week;
  S4: Trucks change assignment– drivers should not often change the assigned trucks in a week;

2.2. Mathematical model and formulation

In this subsection we present a mathematical formulation of all hard and soft constraints described in the previous subsection as well as the fitness function used by the proposed algorithm in order to evaluate each candidate feasible solution. As known, the formulation of an optimization problem's mathematical model is the first major step in order to solve it (Kazakov and Lempert, 2015). The basic variables (parameters and data sets) needed for modeling the problem at hand are the following:

- N is the set of drivers for which the shift assignment is performed ($N=\{1,\ldots,|N|\}$)
- T is the set of days (planning horizon) during which drivers are to be scheduled ($T=\{1,\ldots,|T|\}$, $\omega\in\{4,5,6\}$ is the number of weeks in T)
- F is the set of tank trucks ($F=\{1,\ldots,|F|\}$)
- S_t is the set of all different shift types (combination of tank truck and shift type) on day $t\in T$

Each candidate solution is represented by a $|N|$ x $|T|$ matrix **S**, where $f_{n,t}$ is the number of the tank truck assigned to driver n at day t (figure 1). If driver n has no tank track assigned at day t then $f_{n,t}$ equals to -1. Using this representation we fulfill the second hard constraint (H2) by default, since we ensure that no driver can work more than one shift at the same day. In the next paragraphs, the mathematical formulation of the problem is given. A first attempt to present a mathematical formulation of the problem can be found in (Knust and Schumacher, 2011).

$$\begin{array}{cc|ccccc|}
 & & \multicolumn{5}{c}{T\text{ days}} \\
 & & \text{day }1 & \dots & \text{day }t & \dots & \text{day }|T| \\
\hline
 & \text{driver }1 & f_{11} & \cdots & f_{1t} & \cdots & f_{1|T|} \\
 & \dots & \cdots & \cdots & \cdots & \cdots & \cdots \\
N\text{ drivers} & \text{driver }n & f_{n1} & \cdots & f_{nt} & \cdots & f_{n|T|} \\
 & \dots & \cdots & \cdots & \cdots & \cdots & \cdots \\
 & \text{driver }|N| & f_{|N|1} & \cdots & f_{|N|t} & \cdots & f_{|N||T|} \\
\hline
\end{array}$$

Figure 1. Candidate solution representation

The hard constraints addressed can be formulated as follows:

- constraint_H1: $\sum_{j\in N_s\cup d} r_{jst}=1,\ \forall\ s\in S_t, t\in T$,

where $r_{jst}\in\{0,1\}$ equals 1 if shift $s\in S_t$ is assigned to driver $j\in N_s\cup\{d\}$ on day $t\in T$ and 0 if it is not. $N_s\subseteq N$ is the set of all drivers which may be assigned to shift *s,* while *d* is a dummy driver who is allowed to drive any truck in every shift. Of course, $r_{jst}=0, \forall t\in T, s\in S_t, j\in N/N_s$.

- constraint_H2: $\sum_{s\in S_t} r_{jst}\le 1,\ \forall\ j\in N, t\in T$.

- constraint_H3: $\sum_{s\in S_t} r_{jst}=0,\ \forall\ j\in N, t\in T\cap Unavailable_j$, where $Unavailable_j \subseteq T$ is the set of days of driver $j \in N$ for which he is unavailable.

- ***constraint_H4:*** $\sum_{t \in T_\tau, s \in S_t} length(s,t) \cdot r_{jst} \leq 55, \forall\ j \in N, \tau \in \{1,...,\omega\}$,

where $length(s,t)$ estimates the length of shift $s \in S_t$ on day $t \in T$ and $T_\tau \subset T$ is the set of all days in week τ.

- ***constraint_H5:*** $\sum_{s \in S_t^{late}} r_{jst} + \sum_{s \in S_{t+1}^{early}} r_{js(t+1)} \leq 1, \forall\ j \in N, t \in T^{Monay-Friday}$,

where $T^{Monday-Friday} \subset T$ is the set of all days from Monday to Friday, $S_t^{late} \subset S_t$ is the set of early shifts on day $t \in T$ and $S_{t+1}^{early} \subset S_t$ is the set of late shifts on day $(t+1) \in T$.

- ***constraint_H6:*** $v_{j\tau}^{early} + v_{j\tau}^{late} \leq 1, \forall\ j \in N, \tau \in \{1,...,\omega\}$,

where $v_{j\tau}^{early} \in \{0,1\}$ equals 1 if driver $j \in N$ is assigned to early shifts in week τ and 0 if it is not and $v_{j\tau}^{late} \in \{0,1\}$ equals 1 if driver $j \in N$ is assigned to late shifts in week τ and 0 if it is not, respectively. The following two inequalities guarantee that binary variables $v_{j\tau}^{early}$ and $v_{j\tau}^{late}$ are set correctly:

$\sum_{t \in T_\tau} \sum_{s \in S_t^{early}} r_{jst} - |T_\tau| \cdot v_{j\tau}^{early} \leq 0, (\forall\ j \in N, \tau)$ and $\sum_{t \in T_\tau} \sum_{s \in S_t^{late}} r_{jst} - |T_\tau| \cdot v_{j\tau}^{late} \leq 0, (\forall\ j \in N, \tau)$.

- ***constraint_H7:*** $\sum_{s \in S_t^{early}} r_{jst} + \sum_{s \in S_{t'}^{early}} r_{jst'} \leq 1, \forall\ j \in N^{no\_con\sec utive\_weeks}, t \in T_\tau, t' \in T_{\tau+1}, \tau \in \{1,...,\omega\}$

and

$\sum_{s \in S_t^{late}} r_{jst} + \sum_{s \in S_{t'}^{late}} r_{jst'} \leq 1, \forall\ j \in N^{no\_con\sec utive\_weeks}, t \in T_\tau, t' \in T_{\tau+1}, \tau \in \{1,...,\omega\}$,

where $N^{no\_con\sec utive\_weeks}$ is the set of drivers which should not have the same type of shift in two consecutive weeks.

- ***constraint_H8:*** $\sum_{t \in T} \sum_{s \in S_t} r_{jst} - 2\left(v_{j\tau}^{early} + v_{j\tau}^{late}\right) \geq 0, \forall\ j \in N^{permanent}, \tau \in \{1,...,\omega\}$,

$\sum_{s \in S_{t-1}} r_{js(t-1)} - \sum_{s \in S_t} r_{jst} \leq 0,\ \forall\ j \in N^{permanent}, t \in T^{Tuesday} \setminus Unavailable_j$,

$\sum_{s \in S_{t-2}} r_{js(t-2)} + \sum_{s \in S_{t-1}} r_{js(t-1)} - 2\sum_{s \in S_t} r_{jst} \leq 0,\ \forall\ j \in N^{permanent}, t \in T^{Wednesday} \setminus Unavailable_j$,

$\sum_{s \in S_{t+1}} r_{js(t+1)} + \sum_{s \in S_{t+2}} r_{js(t+2)} - 2\sum_{s \in S_t} r_{jst} \leq 0,\ \forall\ j \in N^{permanent}, t \in T^{Thursday} \setminus Unavailable_j$,

$$\sum_{s \in S_{t+1}} r_{js(t+1)} - \sum_{s \in S_t} r_{jst} \leq 0 \,, \forall \; j \in N^{permanent}, t \in T^{Friday} \setminus Unavailable_j \,,$$

where $N^{permanent} \subset N$ is the set of permanent drivers, $T^{Tuesday} \subset T$ is the set of all Tuesdays, $T^{Wednesday} \subset T$ is the set of all Wednesdays, $T^{Thursday} \subset T$ is the set of all Thursdays and $T^{Friday} \subset T$ is the set of all Fridays.

The mathematical formulation of the costs regarding the four soft constraints is as follows:

- *cost_S1:* $\sum_{j \in N \cup \{d\}} \left(w_{1j}^{+} \cdot \Delta_j^{+} + w_{1j}^{-} \cdot \Delta_j^{-} \right),$

where $\Delta_j^{+}, \Delta_j^{-} \geq 0$, are the deviations of driver $j \in N \cup \{d\}$ from desired total working time, $\sum_{t \in T} \sum_{s \in S_t} length(s,t) \cdot r_{jst} - Z_j^{max} - \Delta_j^{+} \leq 0$, $Z_j^{min} - \sum_{t \in T} \sum_{s \in S_t} length(s,t) \cdot r_{jst} - \Delta_j^{-} \leq 0$, $\left[Z_j^{min}, Z_j^{max} \right]$ is the desired total working time interval for driver $j \in N$ and w_{1j}^{+}, w_{1j}^{-} are the weights associated with a deviation from the desired total working times for driver $j \in N$, overtime and undertime, respectively.

- *cost_S2:* $\sum_{t \in T} \sum_{s \in S_t} \sum_{j \in N_s} w_2 \cdot suitable(tan\,k\_truck(s), j) \cdot r_{jst}$,

where *tank_truck*(*s*) is the tank truck corresponding to shift *s*, $suitable(tan\,k\_truck(s), j) \in [0,1]$ equals 1 if driver $j \in N$ is unsuitable for driving tank truck corresponding to shift *s* and 0 otherwise and w_2 is the weight associated with driver $j \in N$ in case he is not the most favored one for the assigned truck.

- *cost_S3:* $\sum_{\tau=1}^{\omega} \sum_{j \in N} w_3 \cdot number\_trucks\_assigned\,(j, \tau),$

where $number\_trucks\_assigned\,(j,\tau) \geq 0$, $(j \in N, \tau = 1,...,\omega)$ is the number of different trucks minus one assigned to driver *j* in week *τ*and w_3 is the weight associated with driver $j \in N$ in case he is assigned to more than one truck in the same week. Let us now define variable $truck\_assigned(i,j,\tau) \in \{0,1\}$ which equals 1 if truck $i \in F$ is assigned to driver $j \in N$ in week *τ* and 0 otherwise. To ensure that binary variables $truck\_assigned(i,j,\tau)$ are set correctly, the following inequalities must hold:

$truck\_assigned(i,j,\tau) - \sum_{t \in T_\tau} \sum_{\{s \in S_t | tan\,k\_truck(s)=i\}} r_{jst} \leq 0, (\forall \; i \in F, j \in N, \tau)$ and

$6 \cdot truck\_assigned(i,j,\tau) - \sum_{t \in T_\tau} \sum_{\{s \in S_t | tan\,k\_truck(s)=i\}} r_{jst} \geq 0, (\forall \; i \in F, j \in N, \tau).$

Moreover, the following inequality guarantees that variables $number\_trucks\_assigned(j,\tau)$ are set correctly:

$$number\_trucks\_assigned(j,\tau) - \sum_{i \in F} truck\_assigned(i,j,\tau) \geq -1, (\forall\ j \in N, \tau)$$

- *cost_S4:* $\sum_{t \in T^{Monday-Friday}} \sum_{j \in N} \sum_{i \in F} \sum_{k \in F, i \neq k} w_4 \cdot trucks\_assigned\_sucessive\_days(j,i,k,t)$,

where $trucks\_assigned\_sucessive\_days(j,i,k,t) \in \{0,1\}$ is set to 1 if driver $j \in N$ is assigned to truck $i \in F$ on day $t \in T$ and to truck $k \in F$ on day $t+1$ and *w*$_4$is the weight associated with driver $j \in N$ in case he is assigned different trucks in successive days. The following inequality ensures that binary variables $trucks\_assigned\_sucessive\_days(j,i,k,t)$ are set correctly:

$$trucks\_assigned\_succesive\_days(j,i,k,t) - \sum_{\{s \in S_t | tank\_truck(s)=i\}} r_{jst} - \sum_{\{s \in S_t | tank\_truck(s)=k\}} r_{js(t+1)} \geq -1,$$
$$(\forall\ i,k \in F, j \in N, t \in T^{Monday-Friday}).$$

So, the objective function of the problem can be expressed as follows:

$$min(cost\_S1 + cost\_S2 + cost\_S3 + cost\_S4)$$

3. ALGORITHM DESCRIPTION

3.1. Swap mechanism's structure

Firstly, we have to present some fundamental definitions regarding the possible moves upon which, the six out of nine swap mechanisms used, are based. These six swap mechanisms, as already mentioned in Section 1, are well established in the respective literature (Burke et al., 2003; Solos et al., 2013; Lü et al., 2011). A *move* is simply an elementary swap of the content of two cells. We define as a *simple move* the content swap between an empty cell and a non-empty cell. In addition, we define as a *non-empty move* the content swap between non empty cells. Finally, we define as *general move* the content swap between two cells, no matter what their content is. Also, *initial fitness* is the fitness of the candidate solution before any swap mechanism is applied. Next, we give the pseudo code of each one of the first six swap mechanism that the proposed algorithm uses.

1. Column Simple Move

```
for each column {
        try all different simple moves between cells belonging to this column
        retain the simple move that reduces the initial fitness at most
}
```

The above mechanism is applied to all columns one by one. For each column, all simple moves are tested. After testing all simple moves in a column, we apply the unique move that reduces the initial fitness at most, if such a move exists, and restore all other moves. Afterwards, we proceed to the next column and repeat the same procedure.

2. Column Non Empty Move

```
for each column {
        try all non-empty moves between cells belonging to this column
        retain the non-empty move that reduces the initial fitness at most
}
```

The above mechanism is applied to all columns one by one. For each column, all non-empty moves are tested. After testing all non-empty moves in a column, we apply the unique move that reduces the initial fitness at most, if such a move exists, and restore all other moves. Afterwards, we proceed to the next column and repeat the same procedure.

3. Column General Move

```
for each column {
        try all general moves between cells belonging to this column
        retain the general move that reduces the initial fitness at most
}
```

The above mechanism is applied to all columns one by one. For each column, all general moves are tested. After testing all general moves in a column, we apply the unique move that reduces the initial fitness at most, if such a move exists, and restore all other moves. Afterwards, we proceed to the next column and repeat the same procedure.

4. Matrix General Move

```
for each column {
        try all general moves between cells belonging to this column
}
retain the general move that reduces the initial fitness at most
```

The above mechanism considers all columns one by one. For each column, all general moves are tested. When all general moves of all columns have been tried, the one that reduces the initial fitness at most is applied while all others are restored. This mechanism is similar to the Column General Move mechanism. The difference is that, here, we select only one general move among the general moves of all columns, i.e. we consider the matrix as a whole instead of examining the matrix in a column wise fashion.

5. Matrix Simple Move

```
for each column {
        try all simple moves between cells belonging to this column
}
retain the simple move that reduces the initial fitness at most
```

The above mechanism considers all columns one by one. For each column, all simple moves are tested. When all simple moves of all columns have been tried, the one that reduces the initial fitness at most is selected and applied while all others are restored. It is obvious that there is a similarity of this mechanism with the Column Simple Move mechanism. The difference is that, here, the matrix is considered as a whole and not in a column wise manner.

6. Matrix Non Empty Move

```
for each column {
        try all non-empty moves between cells belonging to this column
}
retain the non-empty move that reduces the initial fitness at most
```

The above mechanism selects the non-empty move, among non-empty moves of all columns, which reduces the initial fitness at most and applies it, while all others are restored. This mechanism is similar to Column Non Empty Move mechanism. The difference is that, here, we select only one non empty move among the non-empty moves of all columns.

Next, we introduce a function on which the two of the rest three swap mechanisms are based. This function is called *Core_Swap*().

- *Core_Swap*($line_1$, $line_2$)

```
1    $f_{11} \leftarrow$ fitness value of $line_1$
2    $f_{12} \leftarrow$ fitness value of $line_2$
3    create a random list $L$ of all days (columns)
4    for each element (i.e. day) $day_1$ of list $L$ { /* 1st for loop */
5            for each element $day_2$ next to $day_1$ of list $L$ { /* 2nd for loop */
6                    for each $day$ between $day_1$ and $day_2$ { /* 3rd for loop */
7                            swap the content of cells [$line_1$][$day$] and [$line_2$][$day$]
8                    } /* end 3rd for loop */
9            $f_{21} \leftarrow$ fitness value of $line_1$
10                   $f_{22} \leftarrow$ fitness value of $line_2$
11           if ($f_{11} + f_{12} < f_{21} + f_{22}$){
12                           for each $day$ between $day_1$ and $day_2$ { /* 4th for loop */
13                                   cancel the swap of cells [$line_1$][$day$] and [$line_2$][$day$]
14                   } /* end of 4th for loop */
15           } /* end if */
16           else {
17                           solution's fitness$\leftarrow$ solution's fitness - $f_{11}$ - $f_{12}$ + $f_{21}$ + $f_{22}$
18                           $f_{11} \leftarrow f_{21}$
19                           $f_{12} \leftarrow f_{22}$
20                   } /* end else */
21                   } /* end 2nd for loop */
22   } /* end 1st for loop */
23   Return solution and fitness
```

The above function utilizes a random list *L* of all columns (days). Then, each column of the list is paired with all next columns, sequentially. Each pair of columns (days) defines a range of days where the swaps are going to take place. For each day belonging to this range and for lines *line*$_1$ and *line*$_2$, which are given as input arguments to the function, all the elementary cell swaps are tested, i.e. cell of that day (column) that belongs to *line*$_1$ is swapped with the corresponding cell of *line*$_2$. Each swap is retained if the fitness value after the swap is smaller than or equal to the fitness before the swap. Next, we proceed to the next column belonging to the specified range, applying the same procedure. When all columns inside this range have been selected, the range is altered as far as its end point (right point) is concerned. The new day that is selected from the random list of columns, which lies next to the firstly selected day, specifies the new right end point of the range. This new range comprises the bank from which columns are going to be chosen for the swap performance.

After presenting the *Core_Swap*() function we proceed to the pseudo code of the two swap mechanisms that are based on it.

7. Swap Sorted Successive Rows

```
Sort the rows in descending order based on their fitness and store them in list L
for each element i, except the last, of L {
        for each element j of L {
                apply function Core_Swap(i, j)
        }
}
return solution with its fitness
```

8. Swap Rows Randomly

```
Create two lists $L_1$ and $L_2$ comprised of all randomly selected rows
/* both lists have length equal to the number of all days (columns) */
for each element i of $L_1${
        for each element j of $L_2${
                apply function Core_Swap(i, j)
        }
}
return solution and its fitness
```

Observing the two snippets of pseudo code presented above, we easily conclude that the two swap mechanisms have a lot in common, but they differ in the way rows are selected for swapping. In Swap Sorted Successive Rows mechanism, we first sort the rows in descending order based on their fitness. Hence, each row is swapped only with other rows having worse fitness value. On the other hand, Swap Rows Randomly mechanism selects rows for swapping randomly. That is, each row is swapped with all other rows in a random order.

Before presenting the ninth swap mechanism, we introduce the function on which it is based, called *Core_Segment_Swap*().

- *Core_Segment_Swap*(*$line_1$, $line_2$, $column_1$, $column_2$*)

```
$f_{11} \leftarrow$ fitness of $line_1$
$f_{12} \leftarrow$ fitness of $line_2$
for each column $i$ between $column_1$ and $column_2$ {
    swap the content of cells [$line_1$][$i$], [$line_2$][$i$]
} /* end for */
$f_{21} \leftarrow$ fitness of $line_1$
$f_{22} \leftarrow$ fitness of $line_2$
if ($f_{11} + f_{12} < f_{21} + f_{22}$) {
    for each column $i$ between $column_1$ and $column_2$ {
        cancel the swap of cells [$line_1$][$i$], [$line_2$][$i$]
    } /* end for */
} /* end if */
else {
    Retain the swaps
    solution's fitness$\leftarrow$ solution's fitness - $f_{11} - f_{12} + f_{21} + f_{22}$
} /* end else */
Return solution and fitness
```

We call the above function *Core_Segment_Swap*()because, on the one hand, it is the *core* of the ninth swap mechanism, as mentioned above, and, on the other hand, it attempts all swaps between two lines (rows) that are limited between two columns, forming a *segment*. Next, we present the pseudo code of the ninth swap mechanism called Collapsible Window Stochastic Swap. The name is justified by the observation that the swaps are tried within a rectangle parallelogram area that forms a moving collapsible window and the swaps are done with a certain probability.

9. Collapsible Window Stochastic Swap

```
Create a random list L of all rows
for each column $col_1${
    for each column $col_2$ that is next to $col_1${
        for $i = 0$ to total_number_of_rows - 1{
            for $j = 0$ to total number of rows - 1{
                apply function Core_Segment_Swap($L(i)$, $L(j)$, $col_1$, $col_2$) with
                a certain probability $p$ (see respective formula below)
            }
        }
    }
}
return solution and fitness
```

The probability p of applying function *Core_Segment_Swap*() is given by the following formula: $p = 1 - \frac{d}{D}$, where d is the distance of columns $col_1$and col_2 (i.e. col_2 - col_1) and D is the total length of the planning horizon, i.e. the total number of columns. It is worthwhile to mention that, as derived from the definition of probability p, the greater the value of parameter d is, the smaller the probability of trying the swaps becomes. So, bigger segments of columns have fewer odds to be selected for swapping.

Next, we present the perturbation schema used by the proposed algorithm. As it is previously stated, this schema is activated when there is no evolution of the candidate solution's fitness value for a specific number of cycles.

- Perturbation Schema

```
    select two different rows $r_1$ and $r_2$ at random
    select a column c at random
    swap the content of cells [$r_1$][c], [$r_2$][c]
    return new solution and new fitness
```

This schema assists the VNS approach to escape from local optima, since it causes a sufficient perturbation to current individual. Exhaustive experimental results have shown that using this specific form of perturbation enhances the searching ability of the proposed VNS approach since it serves its cause effectively, that is it causes the necessary perturbation in order to help the proposed algorithm to escape from local optima.

3.2. Main algorithm's structure

In this section, the flowchart of the main algorithm is presented (figure 2), followed by explanatory comments on some critical points of it. The algorithm uses one individual (candidate solution) and tries to improve its fitness by applying various swap mechanisms to it (see subsection 3.1). Except for that, throughout its execution, it keeps the best individual found and its respective fitness. The application of each swap mechanism leads the algorithm to search in a different neighborhood of the search space, thus enhancing its searching capability. Whenever the fitness value calculation of the current individual (candidate solution) is needed, this is performed using the equation of the objective function presented in section 2.2. This objective function incorporates all equations of the proposed algorithm as presented in section 2.2.

Initially, we employ a simple heuristic in order to determine an initial solution, regardless if it is feasible or not (step 1). Next, the best solution is set to this initial solution (step 2) and its fitness value is assigned to *current_f*, which is the parameter having the current's solution fitness, *previous_best_f*, which is the parameter having the best fitness of previous cycle's solution and *best_f*, which is the parameter having the best fitness achieved so far (step 3). After that, the values of algorithm's user defined parameters pN_1, h and k are set (step 4). Please note that pN_1 is the probability of Swap

Sorted Successive Rows mechanism to be selected for execution. If this swap mechanism is not selected due to the probability value used, then Swap Rows Randomly mechanism is executed. The tuning of the algorithm's user defined parameters has been decided after having conducted exhaustive experiments and observing the behavior of the proposed algorithm for various combinations of its parameters' values. In section 5 the effect of the main user defined parameters to algorithms' performance is investigated and the results of different specific parameter values are presented. So, the initial value of pN_1 is set equal to 0.5 while *h*, which is the no fitness evolution tolerance limit of the algorithm's cycles, is set to 2. Also, the perturbation tension *k*, which is the number of times that the perturbation schema is applied to current individual, is set to 5.After setting parameters pN_1,*h*and *k*, the main body of the proposed algorithm starts to run for as long as the termination criterion is not met (step 5). This criterion is a time limit one, which is set to 10 minutes, so as an approximate fair comparison can be established between the proposed algorithm and the former effort to solve the same problem, which uses the same time limit (Knust and Schumacher, 2011). At step 6 (Fig. 2) we set parameter *no_change_counter* equal to 0. This parameter counts the loop cycles during which the fitness is not improving. As long as parameter *no_change_counter* is less than *h* a while loop is initiated (step 7). At step 8, the Swap Sorted Successive Rows mechanism is selected for execution with probability pN_1, which initially is set to 0.5, while afterwards it is dynamically updated (step 21). If this swap mechanism is not selected, then Swap Rows Randomly mechanism is executed. What follows, is the updating of current fitness (step 9). Next, if the execution of either Swap Sorted Successive Rows or Swap Rows Randomly mechanism has improved the best solution's fitness (step 10) the update of either variable N_1*_improvements* (step 12) or N_2*_improvements* (step 13) occurs. Parameter N_1*_improvements* is a counter showing how many times Swap Sorted Successive Rows mechanism, while parameter N_2*_improvements* is a counter showing how many times Swap Sorted Successive Rows mechanism has improved the best solution's fitness, respectively. After that, the update of best solution and its fitness is performed (step 14). Next, at step 15, the serial execution of seven swap mechanisms is applied. After the execution of each one of them, there is a potential update of best solution and its fitness, in case any of these swap mechanisms improves global best. For a detailed description of these mechanisms please refer to Section 3.1. If there is an improvement to the best fitness value (step 16) then the value of parameter *previous_best_f* is set equal to parameter *best_f* and parameter *no_change_counter* is set to zero (step 17), while at the opposite case parameter *no_change_counter* is increased by one (step 18). Immediately after exiting the while loop of step 7 the current solution is set to the best solution found so far with probability equal to $q = 0.5$ (step 19). The current solution, no matter whether it is substituted by best solution or not, is then involved in a perturbation procedure which is performed according to the perturbation schema described in Section 3.1. The perturbation schema is applied for *k* times, where *k* is set to 5, as stated above (step 20). This value has also been selected after having conducted exhaustive experiments and observing the behavior of the proposed algorithm for many different values of it (see Section 5). By alternatively selecting either the current solution or the best solution found so far to apply the perturbation procedure on it, we offer the algorithm an extra flexibility in order to overcome the obstacle of premature convergence and therefore escaping entrapment in local optima. An important point that needs to be commented is step 21, at which the dynamic computation of the selection probability pN_1 is done. Our aim is to force a kind of biased execution of Swap Sorted Successive Rows and Swap Rows Randomly mechanisms depending on which of them has improved the best solution’s fitness the more times. It is obvious that the more Swap Sorted Successive Rows mechanism improves the best solution’s fitness (step 11) the greater the value of variable N_1_*improvements* gets (step 12). Consequently, the value of pN_1 is getting higher (see formula at step 21). As a result, the priority of Swap Rows Randomly mechanism is decreased. Finally, at step 22, after the chosen termination criterion is satisfied, the algorithm returns the best found solution and terminates.

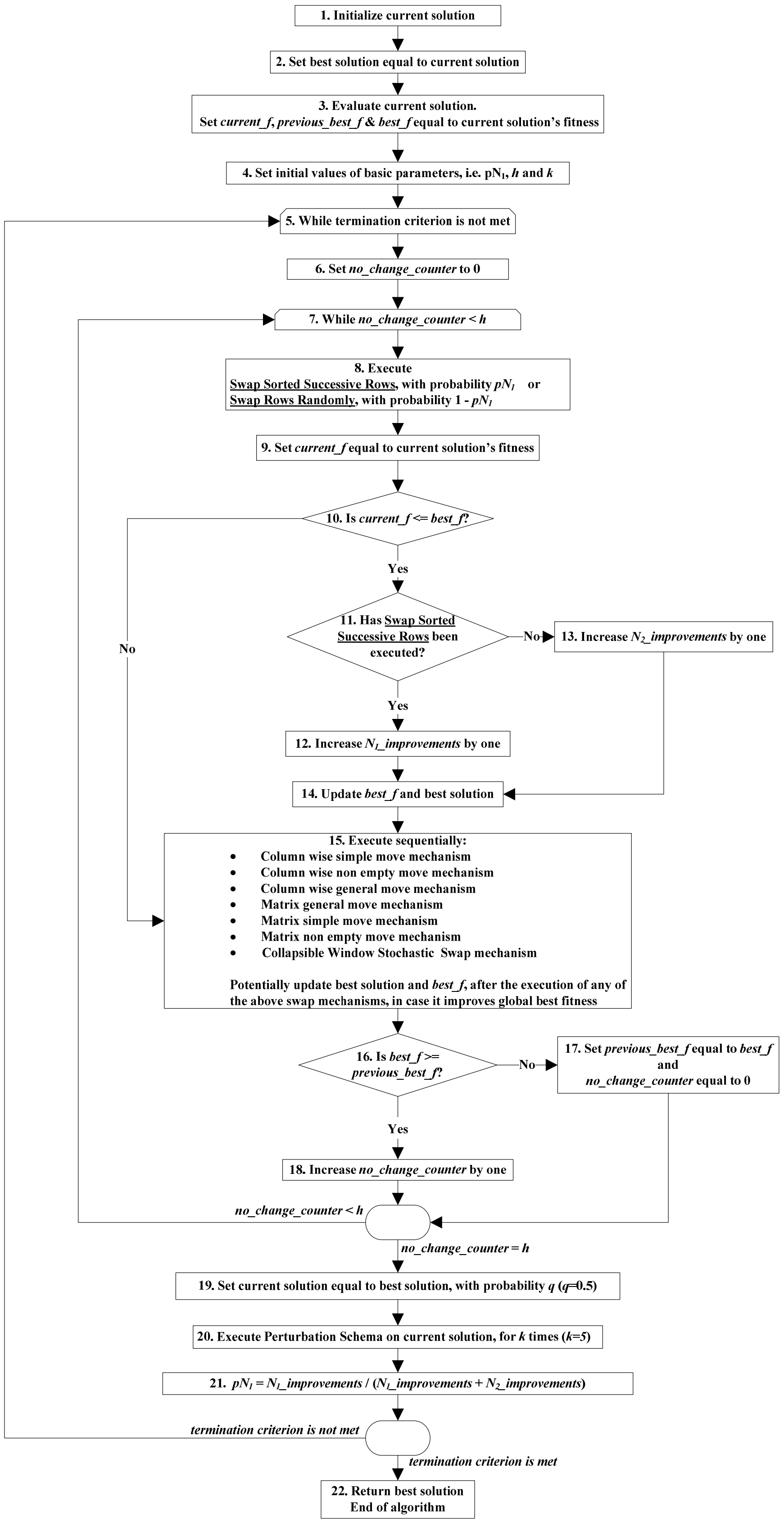
1. Initialize current solution
2. Set best solution equal to current solution
3. Evaluate current solution. Set *current_f*, *previous_best_f* & *best_f* equal to current solution's fitness
4. Set initial values of basic parameters, i.e. pN_1, *h* and *k*
5. While termination criterion is not met
6. Set *no_change_counter* to 0
7. While *no_change_counter* < *h*
8. Execute Swap Sorted Successive Rows, with probability pN_1 or Swap Rows Randomly, with probability 1 - pN_1
9. Set *current_f* equal to current solution's fitness
10. Is *current_f* <= *best_f*?
Yes
No
11. Has Swap Sorted Successive Rows been executed?
No
13. Increase N_2*_improvements* by one
Yes
12. Increase N_1*_improvements* by one
14. Update *best_f* and best solution
15. Execute sequentially:
Column wise simple move mechanism
Column wise non empty move mechanism
Column wise general move mechanism
Matrix general move mechanism
Matrix simple move mechanism
Matrix non empty move mechanism
Collapsible Window Stochastic Swap mechanism
Potentially update best solution and *best_f*, after the execution of any of the above swap mechanisms, in case it improves global best fitness
16. Is *best_f* >= *previous_best_f*?
No
17. Set *previous_best_f* equal to *best_f* and *no_change_counter* equal to 0
Yes
18. Increase *no_change_counter* by one
no_change_counter < *h*
no_change_counter = *h*
19. Set current solution equal to best solution, with probability *q* (*q*=0.5)
20. Execute Perturbation Schema on current solution, for *k* times (*k*=5)
21. $pN_1 = N_1\_improvements / (N_1\_improvements + N_2\_improvements)$
termination criterion is not met
termination criterion is met
22. Return best solution End of algorithm

Figure 2. The flow chart of the proposed optimization algorithm.

4. COMPUTATIONAL RESULTS

4.1. Input data

The thirty input instances used in current contribution, comprise real world data provided by a small oil company (Knust and Schumacher, 2011; http://www2.informatik.uni-osnabrueck.de/kombopt/data/tanktrucks/). In each input instance the following information is given sequentially:

- days of weeks in the planning horizon (0=no working day); each week starting with Monday
- number of drivers
- list of permanent drivers
- list of temporary drivers
- dummy driver
- number of trucks
- list of trucks operated in a single shift per day
- list of trucks operated in two shifts (early/late) per day
- list of trucks which are driven on Saturday in a single shift
- lists of feasible drivers for the tank trucks in normal shifts
- lists of feasible drivers for the tank trucks in early shifts
- lists of feasible drivers for the tank trucks in late shifts
- list of drivers who cannot have the same type of shift in two consecutive weeks
- vacation days for drivers
- minimal total working times for drivers
- maximal total working times for drivers
- shift lengths for trucks operated in a single shift
- shift lengths for trucks on Saturdays
- shift lengths for trucks in early shifts
- shift lengths for trucks in early shifts
- priorities (suitability) for trucks and drivers

In all instances there are thirty drivers (number 1 to 30) and seventeen tank trucks (number 1 to 17). Regarding the drivers, twenty-five of them are permanent (number 1 to 25) while five of them are temporary (number 26 to 30). Moreover, two drivers cannot have the same type of shift in two consecutive weeks (number 12 and 21). Regarding the trucks, the first three have to drive in two shifts (early and late) while the remaining fourteen trucks are only to be scheduled in a single shift from Monday to Friday. Furthermore, the first four tank trucks can only be operated within a single shift on Saturdays. Concerning the length of shifts, this equals ten hours for all trucks from Monday to Friday and five hours on Saturdays. Each tank truck has four preferred drivers for an early shift, except for one truck (number 7) for which no most favored driver is defined for an early shift. On the other hand, for the first three trucks (number 1, 2 and 3) four preferred drivers are also defined for a late shift.

Except for that, for seven trucks (number 1, 2, 3, 4, 5 12 and 14) some additional acceptable (miscellaneous) drivers are given.

Regarding the default total working time intervals for permanent drivers, the input instances are divided in two sets. In the first fifteen input instances, the total working time intervals for permanent drivers are set to [122, 180] while for the rest fifteen instances are set to [122, 200]. However, in all thirty instances there is one permanent driver (number 24) whose total working time interval equals [0, 50], because he is a “jumper” for one tank truck. Also, for all thirty instances, the default total working time interval for each temporary driver is set to [0, 50], except for two temporary drivers (number 28 and 29) whose total working time interval is set to [0, 20], since they are only available on Saturdays.

Function $suitable(tan k\_truck(s), j)$, where *tank_truck*(*s*) is the tank truck corresponding to shift *s* and $j \in N$, which measures how suitable is driver j for driving the corresponding tank track of shift *s*, returns 0 for the most favored driver and 0.1, 0.2, 0.3 for the next three acceptable drivers (ordered accordingly to the preference list of each truck). Moreover, $suitable(tan k\_truck(s), j)$ returns 0.5 for all miscellaneous drivers. The weights in the objective function are set to $w_{1j}^{+} = w_{1j}^{-} = 10, w_2 = 3, w_3 = w_4 = 2$ for permanent drivers, $w_{1j}^{+} = w_{1j}^{-} = w_2 = w_3 = w_4 = 1$ for temporary drivers, and $w_d^{+} = 10000$ for the dummy driver, as proposed by Knust and Schumacher (2011). The reason why the weight values for temporary drivers are much smaller is that their driving hours are not evenly dispersed, that is, “in some months they drive more and in other months they drive less hours” (Knust and Schumacher, 2011).

Regarding vacation days, instances 1 and 16 have no vacation days while the other twenty eight instances have. The total number of vacation days for each of these instances is randomly distributed to the drivers and varies from five to forty vacation days. The number of drivers having vacation days in these instances varies from one to six drivers. For a more detailed description of all instances used, the interested reader can refer to Knust and Schumacher (2011) and http://www2.informatik.uni-osnabrueck.de/kombopt/data/tanktrucks/.

4.2. Algorithm’s performance

The stochastic VNS approach presented is coded in C++. It is run on Intel® Core™ 2 Duo CPU E7500 2.93 GHz under the Windows 7 OS. The values used for algorithms’ parameters are the ones discussed in subsection 3.2. In order to evince the algorithm’s stability and efficiency we calculate and present the best, the worst and the average results together with the respective standard deviations, regarding the fitness function value achieved. Also, in column 6 the average execution time of the proposed algorithm for each input instance is presented. Column 7 gives the previous best known solutions reported in Knust and Schumacher (2011), while in column 8 the best solutions found by the VNS algorithm introduced by Solos et al. (2013) for each input instance is presented. Additionally, in column 9 the optimal solution for each input instance is presented while column 10 reports the %

improvement that the proposed algorithm has achieved, compared to the results presented in Knust and Schumacher (2011). Finally, in the last column the % deviation of the solution provided by the proposed algorithm compared to the optimal solution, for each input instance, is reported. Results presented in table 1 have been calculated after executing the proposed VNS approach for 30 Monte Carlo runs under the time limit of 10 minutes.

Table 1: Computational results under the 10 minutes time limit.

Input instance	**Proposed algorithm**					**Best fitness reported in Knust and Schumacher (2011)**	**Best fitness found by Solos et al. (2013)**	**Optimal solution**	**% Improvement compared to Knust and Schumacher (2011)**	**% Deviation from optimal solution**
	Average fitness	**STD**	**Best fitness**	**Worst fitness**	**Average time (min)**					
122_180_1	271.343	58.70	**231.2**	792.5	8.459	239.9	236.6	219.1	3.63%	5.23%
122_180_2	295.87	193.02	**234.4**	1278.7	8.421	253.2	249.5	219.1	7.42%	6.53%
122_180_3	287.10	103.66	**234.9**	804.2	8.639	254.8	254.5	219.1	7.81%	6.73%
122_180_4	343.99	248.97	**234.7**	1300.0	8.235	272.0	239.2	220.0	13.71%	6.26%
122_180_5	397.96	205.29	**268.8**	963.7	8.254	283.4	279,7	249.1	5.15%	7.33%
122_180_6	536.73	246.84	331.5	1543.2	8.644	**312.0**	328.1	300.1	-6.25%	9.47%
122_180_7	711.29	579.55	**236.7**	2396.5	8.142	247.9	240.5	220.0	4.52%	7.06%
122_180_8	550.39	313.01	**312.2**	1340.3	8.737	317.8	310.4	292.8	1.76%	6.21%
122_180_9	337.01	191.03	**241.1**	1300.1	8.191	269.1	260.4	229.8	10.41%	4.69%
122_180_10	345.26	208.24	**232.4**	878.1	8.755	253.5	244.9	220.0	8.32%	5.34%
122_180_11	445.70	344.22	**297.9**	2315.7	8.485	305.4	301.6	281.3	2.46%	5.57%
122_180_12	321.13	157.59	**243.7**	1301.3	8.629	251.2	250.7	230.6	2.99%	5.38%
122_180_13	415.21	182.12	**297.3**	846.5	8.440	299.0	298.9	264.9	0.57%	10.90%
122_180_14	982.91	371.97	**364.7**	2075.4	8.453	367.2	372.9	353.1	0.68%	3.18%
122_180_15	618.27	363.48	**323.9**	1913.8	8.488	358.7	355.6	306.3	9.70%	5.43%
122_200_1	44.03	4.44	**36.8**	59.2	7.723	67.9	47.8	35.6	45.80%	3.26%
122_200_2	44.76	4.26	**37.9**	57.1	7.894	49.0	43.4	36.2	22.65%	4.49%
122_200_3	43.34	3.15	**37.3**	53.7	7.612	65.3	41,6	37.1	42.88%	0.54%
122_200_4	48.73	7.66	**39.8**	85.4	7.516	90.5	67.9	37.4	56.02%	6.03%
122_200_5	58.95	71.65	**40.8**	553.2	7.569	69.6	47.6	37.4	41.38%	8.33%
122_200_6	50.70	5.16	**39.4**	69.1	7.783	67.8	49	37.5	41.89%	4.82%
122_200_7	48.67	5.02	**41.5**	64.7	7.937	54.9	46.4	38.9	24.41%	6.27%
122_200_8	136.24	180.56	**47.9**	581.9	7.261	81.7	55.3	43.3	41.37%	9.60%
122_200_9	50.75	6.64	**40.7**	69.1	7.585	75.2	47	37.7	45.88%	7.37%
122_200_10	50.64	4.51	**42.1**	59.1	7.954	81.8	49.8	38.8	48.53%	7.84%
122_200_11	74.83	90.49	**43.6**	574.5	7.932	74.2	72.7	41.6	41.24%	4.59%
122_200_12	76.51	132.05	**41.9**	1061.2	7.983	72.6	69.6	39.6	42.29%	5.49%
122_200_13	89.04	131.56	**49.5**	1063.5	8.071	79.7	76.1	41.7	37.89%	15.76%
122_200_14	212.25	203.98	**70.9**	635.1	8.113	103.0	101.4	59.4	31.17%	16.22%
122_200_15	481.90	477.82	**73.4**	2143.4	8.357	87.4	73.5	50.1	16.02%	31.74%

Table 1 presents that for most input instances the average fitness function value achieved by the proposed stochastic VNS approach is very satisfactory. More precisely, in most instances, the mean value is quite close to the best value achieved for each input instance. These results indicate that the proposed VNS approach is quite stable and efficient. Except for that, column 5 shows that the proposed soft computing technique succeeds in finding better best (smallest) fitness in 29/30 instances (96.67%) compared to the two-phase solution algorithm presented in Knust and Schumacher (2011) and the VNS based approach presented by Solos et al. (2013). The only instance for which the proposed algorithm has not succeeded in finding the best till now reported result is instance *122_180_6*. Best known fitness found for each instance is written in bold font. Especially in the last fifteen instances (instance *122_200_1* to instance *122_200_15*), the proposed algorithm's improvement ranges from 16.02% (instance *122_200_15*) to 56.02% (instance *122_200_4*). Moreover, in the feasible shift schedules constructed most drivers have smaller deviations of their desired working times. Using these optimized shift schedules, the oil company can decrease its costs by using employees in a more efficient way. Last but not least, the "% deviation from optimal solution", presented in column 10, is 7.59% on average which also illustrates the efficiency of the proposed approach. The fact that for some input instances the "% deviation from optimal solution" is high (*122_180_6*, *122_180_13*, *122_200_5*, *122_200_8*, *122_200_13*, *122_200_4*, *122_200_15*) demonstrates that these instances are far more difficult than the others, having many local optima. This fact is the main reason why the proposed algorithm does not manage to reach near the optimal solutions for these instances.

Concluding, the proposed algorithm succeeds in finding, for all but one instances, feasible and effective solutions in less than 10 minutes (8.14 minutes on average), which are significantly optimized than previous best known achieved solutions (Knust and Schumacher, 2011). The interested reader can find more information about the txt files used as inputs, the produced txt files containing the resulted shift schedules, as well as, how to run the proposed algorithm online at http://www.deapt.upatras.gr/tank_truck_shift_scheduling/stochastic_variable_neighbourhood_approach_shift_scheduling_tank_trucks.htm. Additionally, at the same webpage the executable of the proposed VNS approach is available in order to be easy for other researchers to reproduce all experimental results reported in table 1.

5. ANALYSIS AND DISCUSSION

5.1. Investigating the performance of the proposed VNS approach

A very important issue of the proposed VNS approach is the effect of user defined parameters to its performance. In the following paragraphs we attempt to investigate the effect of these parameters and come to some very interesting conclusions regarding algorithm's performance. The user defined parameters of the proposed algorithm are the following (see Section 3.2):

- The initial value of probability pN_1, which is the probability of Swap Sorted Successive Rows mechanism to be selected for execution (see figure 2, step 8 of the proposed algorithm).
- *h*, which is the no fitness evolution tolerance limit of the algorithm's cycles (see figure 2, step 18 of the proposed algorithm).
- *k*, which is the perturbation tension, that is, the number of times the perturbation schema is applied to current individual (see figure 2, step 20 of the proposed algorithm).

Since there are no obvious criteria for selecting optimal values for the algorithm's parameters, for all input instances of the problem, we decided to elect them by trial and error, which is common policy in the respective literature (Solos et al. 2013; Tassopoulos and Beligiannis, 2012a; Tassopoulos and Beligiannis, 2012b). As a result, we carried out exhaustive experimental runs and picked the parameters' values that assisted the proposed VNS approach to reach its best performance. Based on these experiments the following conclusions were drawn:

- The initial value of probability pN_1 does not affect the algorithm's performance – so we decided to set it equal to 0.5.
- Setting parameter *h* to a value smaller than 2 or bigger than 5 leads the algorithm to very poor performance – so a valid domain set for parameter *h* is [2 5].
- Setting parameter *k* to a value bigger than 6 leads the algorithm to very poor performance – so a valid domain set for parameter *k* is [1 6].
- The best combination of values for parameters *h* and *k*, which leads the proposed algorithm to its best results for most of the input problem instances, is *h*=2 and *k*=5

In order to justify our last conclusion, we performed exhaustive experiments testing all different combinations of values for *h* belonging to [2 5] and *k* belonging to [1 6]. The respective experimental results demonstrated that setting *h*=2 and *k*=5 is indeed the combination which leads the proposed algorithm to its best results for most of the problem's instances.

5.2. Discussing the applicability of the proposed approach to other related problems

The proposed VNS algorithm, as described in section 3, has been designed in order to optimize the shift scheduling problem of tank trucks for a small oil company (Knust and Schumacher, 2011). However, with minor alterations it can be efficiently applied to other truck driver scheduling problems (Goel et al., 2012; Goel, 2012; Palmgren et al., 2004; Goel, 2010). The flexibility of the proposed stochastic VNS approach is grounded in its inherently adaptive nature. First of all, the solution's encoding (see section 2.2, figure 1) can be easily adjusted in order to model the structure of shift schedules used in other truck driver scheduling problems. Secondly, it is quite easy to modify the procedure which estimates the fitness function value of each candidate solution (see section 2.2), which eventually evaluates the quality of each shift schedule, so as to include (or exclude) other hard and/or soft constraints. Adding or subtracting hard and/or soft constraints, in order the resulting shift schedule to be regarded as feasible and effective for other truck driver scheduling problems, can be easily performed without having to affect the rest procedures of the proposed VNS approach, since the only thing one has to do is to add/subtract the respective cost function from the total objective function of the proposed algorithm (see section 2.2).

6. CONCLUSIONS AND FUTURE WORK

The stochastic VNS approach, presented in this contribution, has been designed in order to optimize the shift scheduling problem of tank trucks for a small oil company. The basic aim of the proposed method is to result in a feasible and efficient shift schedule which would also satisfy the drivers' working times preferences. Experimental results have been compared with a former attempt to solve the exact same problem on the same 30 problem instances. The proposed VNS approach resulted in finding lower bounds for 29 out of 30 problem instances within 10 minutes time. As a result, the proposed approach manages to construct optimized shift schedules in which most drivers have smaller deviations of their desired working times. In this way, the oil company can decrease its costs by using employees in a more efficient way (e.g. by reducing paid overtime). Finally, the application of the proposed algorithm to other truck driver scheduling problems, such as the Canadian, the Australian and the European Union track driver scheduling problem will be one of the main issues of our future work.

7REFERENCES

Burke, E.K., De Causmaecker, P., Petrovic, S., Vanden Berghe, G., 2003, *Variable neighborhood search for nurse rostering problems*. in Resende, M.C.G. and Pinho de Sousa, J. (Eds.), Metaheuristics: Computer Decision-Making, Chapter 7, 153-172, Kluwer Academic Publishers, Norwell, MA, USA.

Burke, E.K., Curtois, T., Qu, R., Vanden Berghe, G., 2007, *A time predefined variable depth search for nurse rostering*. Technical Report No. NOTTCS-TR-2007-6, School of Computer Science and IT, University of Nottingham.

Corominas, A., Lusa, A., Olivella, J., 2012, *A detailed workforce planning model including non-linear dependence of capacity on the size of the staff and cash management*, European Journal of Operational Research, 216, 445-458.

David, R.-C., Precup, R.-E., Petriu E.M., Rădac, M.-B., Preitl, S., 2013, *Gravitational search algorithm-based design of fuzzy control systems with a reduced parametric sensitivity*, Information Sciences, 247, 154-173.

De Matta, R., Peters, E., 2009, *Developing work schedules for an inter-city transit system with multiple driver types and fleet types*, European Journal of Operational Research, 192, 852-865.

Dietz, D.C., 2011, *Practical scheduling for call center operations*, Omega, 39, 550-557.

Dück, V., Ionescu, L., Kliewer, N., Suhl, L., 2012, *Increasing stability of crew and aircraft schedules*, Transportation Research Part C: Emerging Technologies, 20, 47-61.

Elizondo, R., Parada, V., Pradenas, L., Artigues, C., 2010, *An evolutionary and constructive approach to a crew scheduling in underground passenger transport*, Journal of Heuristics, 16, 575-591.

Goel, A., 2010, *Truck driver scheduling in the European Union*, Transportation Science, 44, 429-441.

Goel, A., 2012, *The Canadian minimum duration truck driver scheduling problem*, Computers & Operations Research, 2012, 39, 2359-2367.

Goel, A., Archetti, C., Savelsbergh, M., 2012, *Truck driver scheduling in Australia*, Computers & Operations Research, 39, 1122-1132.

Kazakov, A.L., Lempert, A.A., 2015, *On mathematical models for optimization problem of logistics infrastructure*, International Journal of Artificial Intelligence, 13, 200-210.

Khan, I.H., 2014, *A comparative study of evolutionary algorithms*, International Journal of Artificial Intelligence, 12, 1-17.

Knust, S., Schumacher, E., 2011, *Shift scheduling for tank trucks*, Omega, 2011, 39, 513-521.

Kyngäs, N., Goosens, D., Nurmi, K., Kyngäs, J., 2012, *Optimizing the unlimited shift generation problem*, Applications of Evolutionary Computation, Lecture Notes in Computer Science, 7248, 508-518, Springer, Berlin, Heidelberg.

Li, H.T., Womer, K., 2009, *A decomposition approach for shipboard manpower scheduling*, Military Operations Research, 14, 67-90.

Lü, Z., Hao, J.-K., Glover, F., 2011, *Neighborhood analysis: A case study on curriculum-based course timetabling*, Journal of Heuristics, 17, 97-118.

Maenhout, B., Vanhoucke, M., 2013, *An integrated nurse staffing and scheduling analysis for longer-term nursing staff allocation problems*, Omega, 41, 485-499.

Naudin, E., Chan, P.Y.C., Hiroux, M., Zemmouri, T., Weil, G., 2012, *Analysis of three mathematical models of the staff rostering problem*, Journal of Scheduling, 15, 23-38.

Nissen, V., Gunther, M., 2010, *Automatic generation of optimized working time models in personnel planning*, proceedings of the 7th International Conference on Swarm Intelligence, Brussels, Belgium, Lectures Notes in Computer Science, 6234, 384-391.

Palmgren, M., Rönnqvist, M., Värbrand, P., 2004, *A near-exact method for solving the log-truck scheduling problem*, International Transactions in Operational Research, 11, 447-464.

Rasmussen, M.S., Justesen, T., Dohn, A., Larsen, J., 2012, *The home care crew scheduling problem: preference-based visit clustering and temporal dependencies*, European Journal of Operational Research, 219, 598-610.

Relvas, S., Boschetto-Magatão, S.N., Barbosa-Póvoa, A.P.F.D., Neves, Jr. F., 2013, *Integrated scheduling and inventory management of an oil products distribution system*, Omega, 41, 955-968.

Safaei N, Banjevic D, Jardine AKS. Workforce-constrained maintenance scheduling for military aircraft fleet: a case study. Annals of Operations Research 2011;186:295–316.

Solos, I.P., Tassopoulos, I.X., Beligiannis, G.N., 2013, *A generic two-phase stochastic variable neighborhood approach for effectively solving the nurse rostering problem*, Algorithms, 6, 278-308.

Tassopoulos, I.X., Beligiannis, G.N., 2012a, *Using particle swarm optimization to solve effectively the school timetabling problem*, Soft Computing, 16, 1229-1252.

Tassopoulos, I.X., Beligiannis, G.N., 2012b, *Solving effectively the school timetabling problem using particle swarm optimization*, Expert Systems with Applications, 39, 6029-6040.

Torres, W.C., Quintana, M., Pinzón, H., 2013, *Optimization in Dynamic Environments Utilizing a Novel Method Based on Particle Swarm Optimization*, International Journal of Artificial Intelligence, 11, 150-169.

Tsai, C.-C., Li, S.H.A., 2009, *A two-stage modeling with genetic algorithms for the nurse scheduling problem*, Expert Systems with Applications, 36, 9506-9512.

Uzar, M.F., Catay, B., 2012, *Distribution planning of bulk lubricants at BP Turkey*, Omega. 40, 870-881.

Valdez, F., Melin, P., Castillo, O., 2011, *An improved evolutionary method with fuzzy logic for combining particle swarm optimization and genetic algorithms*, Applied Soft Computing, 11, 2625-2632.

Valls, V., Perez, A., Quintanilla, S., 2009, *Skilled workforce scheduling in service centres*, European Journal of Operational Research, 193, 791-804.

Van den Bergh, J., Beliën, J., De Bruecker, P., Demeulemeester, E., De Boeck, L., 2013, *Personnel scheduling: A literature review*, European Journal of Operational Research, 226, 367-385.

Yazdani D., Nasiri, B., Azizi, R., Sepas-Moghaddam, A., Mohammad Reza Meybodi, M.R., 2013*, A New Algorithm Based on Improved Artificial Fish Swarm Algorithm for Data Clustering*, International Journal of Artificial Intelligence, 11, 170-192.

Zăvoianu, A.-C., Bramerdorfer, G., Lughofer, E., Silber, S., Amrhein, W., Klement, E.P., 2013, *Hybridization of multi-objective evolutionary algorithms and artificial neural networks for optimizing the performance of electrical drives*, Engineering Applications of Artificial Intelligence, 26, 1781-1794.

Zolfaghari, S., Vinh, Q., El-Bouri, A., Khashayardoust, M., 2010, *Application of a genetic algorithm to staff scheduling in retail sector*, International Journal of Industrial and Systems Engineering, 2010, 5, 20-47.