Expert Systems With Applications 65 (2016) 383–397

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

A hybridisation of adaptive variable neighbourhood search and large neighbourhood search: Application to the vehicle routing problem

Jeeu Fong Sze[a,b,*], Said Salhi[a], Niaz Wassan[a]

[a] Centre of Logistics and Heuristics Optimisation (CLHO) Kent Business School, University of Kent, Canterbury, CT2 7PE, United Kingdom
[b] Faculty of Computer Science and Information Technology, Universiti Malaysia Sarawak, 94300 Kota Samarahan, Sarawak, Malaysia

ARTICLE INFO

Article history:
Received 13 April 2016
Revised 22 August 2016
Accepted 22 August 2016
Available online 23 August 2016

Keywords:
Adaptive search
Variable neighbourhood
Large neighbourhood
Data structure
Neighbourhood reduction
Hybridisation

ABSTRACT

In this paper, an adaptive variable neighbourhood search (AVNS) algorithm that incorporates large neighbourhood search (LNS) as a diversification strategy is proposed and applied to the capacitated vehicle routing problem. The AVNS consists of two stages: a learning phase and a multi-level VNS with guided local search. The adaptive aspect is integrated in the local search where a set of highly successful local searches is selected based on the intelligent selection mechanism. In addition, the hybridisation of LNS with the AVNS enables the solution to escape from the local minimum effectively. To make the algorithm more competitive in terms of the computing time, a simple and flexible data structure and a neighbourhood reduction scheme are embedded. Finally, we adapt a new local search move and an effective removal strategy for the LNS. The proposed AVNS was tested on the benchmark data sets from the literature and produced very competitive results.

1. Introduction

The vehicle routing problem (VRP) was first introduced by Dantzig and Ramser (1959) as the Truck Dispatching Problem, which has been extensively studied thereafter because of its high practicability in transportation logistics. The VRP is concerned about the determination of a set of routes for a fleet of vehicles such that each vehicle starts and ends at a single depot, while satisfying all the customers'requirement and operational constraints with the objective of minimising the total transportation cost.

Due to the limited success of exact methods in handling large size problems, most research on the VRP have devoted to the use of heuristic approaches. Among the most popular metaheuristics include tabu search, simulated annealing, genetic algorithm, large neighbourhood search and variable neighbourhood search. Since there is a tremendous amount of work devoted to the classical VRP, we only discuss some of the research which has proven to show relatively good results in solving the problem. Toth and Vigo (2003) proposed the granular tabu search strategy based on the concept of restricted neighbourhoods. Golden, Wasil, Kelly, and Chao (1998) and Li, Golden, and Wasil (2005) combined the record-to-record travel with variable-length neighbourhood list and found a number of new best results. Mester and Bräysy (2005) put forward the active guided evolutionary strategies (AGES) and obtained many best known results. This is partly due to the use of a good quality initial solution. Other methods such as the Greedy Randomized Adaptive Search Procedure (Prins, 2009) and the threshold accepting algorithm of Tarantilis and Kiranoudis (2002) have also been successfully applied. Variable neighbourhood search (VNS) has proved to be one of the most successful metaheuristics for solving different variants of the VRP (Chen, Huang, & Dong, 2010; Fleszar, Osman, & Hindi, 2009; Imran, Salhi, & Wassan, 2009; Kytöjoki, Nuortio, Bräysy, & Gendreau, 2007; Polacek, Hartl, Doerner, & Reimann, 2004; Polat, Kalayci, Kulak, & Günther, 2015). VNS is also known to be competitive at solving many other combinatorial optimisation problems. For instance among the very recent studies include the train scheduling and timetabling problem (Hassannayebi & Hessameddin, 2016; Samà, Ariano, Corman, & Pacciarelli, 2016) and a related VRP known as the swap-body VRP (Todosijević, Hanafi, Urošević, Jarboui, & Gendron, 2016).

The metaheuristics are capable to provide satisfactory results within a reasonable time. However, these approaches face the challenges of dealing with premature convergence. One way to overcome such limitation is by developing hybrid approaches, which have the added advantage of considering the strengths of more than one metaheuristic. Xiao, Zhao, Kaku, and Mladenovic (2014) proposed an interesting hybridisation of the VNS with sim-

* Corresponding author.
E-mail addresses: J.F.Sze@kent.ac.uk, jfs9@kent.ac.uk (J.F. Sze), S.Salhi@kent.ac.uk (S. Salhi), N.A.Wassan@kent.ac.uk (N. Wassan).

http://dx.doi.org/10.1016/j.eswa.2016.08.060
0957-4174/

ulated annealing. Most recently, Akpinar (2016) presented a hybrid algorithm that combined LNS with ant colony optimisation yielding satisfactory results when tested on some VRP instances. For more details about hybrid metaheuristics, interested reader can refer to Raidl (2006). With the success of such hybridisations in solving many combinatorial opimisation problems, this study aims to explore the two metaheuristics, namely VNS and LNS.

The idea of adaptive search in developing algorithms for various combinatorial problems including the VRP has emerged rather strongly in the past decade. The important feature is that the parameters of the algorithm are no longer fine-tuned but adjusted dynamically using the information obtained during the search. Adaptive large neighbourhood search (ALNS) is one of the most popular and successful adaptive approaches, where the selection of the removal and repair operators is performed favouring the ones with a higher success rate. Stenger, Vigo, Enz, and Schwind (2013) integrated a similar selection rule in the shaking phase of VNS. Related research addressing the ALNS include Ropke and Pisinger (2006), Pisinger and Ropke (2007) and Azi, Gendreau, and Potvin (2014). Kritzinger, Doerner, Tricoire, and Hartl (2015) provided an interesting and informative review about adaptive search techniques for a variety of VRPs. For the VNS implementation, the adaptive aspect is usually integrated in the shaking phase. For instance, in Stenger et al. (2013) and in Li, Pardalos, Sun, Pei, and Zhang (2015), the neighbourhood strategy is selected using a roulette wheel method based on the success rate of each neighbourhood. This research is different from the others as it focuses on integrating the adaptive search at the local search phase. In other words, this study is motivated by the idea of incorporating adaptive strategies to guide the search more effectively. To reduce the risk of being trapped at the local optima, we develop a powerful LNS with interesting removal and insertion rules that is integrated with the proposed AVNS acting as a strong diversification strategy.

The contributions of this paper are fourfold:

(i) To propose an effective hybridisation of VNS and LNS,
(ii) To develop an efficient AVNS which integrates learning in its local search phase,
(iii) To introduce a new and effective data structure and neighbourhood reduction scheme,
(iv) To propose a new local search operator and a removal strategy for LNS that incorporates a VNS structure.

To the best of our knowledge, this is the first study that addresses the integration of VNS and LNS with the latter prescribing to a VNS structure. Moreover, the local search operators in our algorithm are selected adaptively reflecting their individual success during the search. In this study, we also adapt a local search operator with a new feature. Within LNS, several removal strategies including a new one which we propose are used. In addition, a newly adapted neighbourhood reduction scheme which restricts the search to a set of neighbouring customers only is developed resulting in a significant decrease in computational effort. The data structure which we introduce is also simple but very effective in expediting the algorithm. The proposed algorithm is tested on the classical VRP, which is used as a platform for our experiments. Since this problem is the basis for other related routing problems, successful implementations could then be easily adapted and extended to cater for other related variants with minimal changes.

The rest of the paper is organized as follows. Section 2 presents the proposed algorithm. The explanation of the mains steps along with their components are provided in Section 3. The computational results are presented in Section 4. Finally, the last section concludes our findings.

2. An adaptive variable neighbourhood search (AVNS) algorithm

The main idea of VNS is to apply a systematic change of neighbourhoods within a local search algorithm in order to escape from local optimality. This metaheuristic, originally proposed by Mladenović and Hansen (1997), attempts to visit the next 'larger' neighbourhood when there is no improvement found and then reverts back to the first 'smaller' one if a better solution is obtained. By returning to the first neighbourhood, the size of the search space is reduced and therefore the search could be performed relatively quicker with the aim of getting a better local optimum. For an overview of heuristic search including VNS, see Salhi (2006) but for more details on VNS and its variants, the reader will find the review paper by Hansen, Mladenović, and Moreno Pérez (2010) to be very informative.

In this paper, an adaptive VNS (AVNS) algorithm presented in Fig. 1 is developed. The AVNS consists of two stages: Stage 1 uses the best improvement VNS whereas Stage 2 applies the multi-level k^{th} improvement VNS. The best improvement VNS in Stage 1 is adapted from the basic VNS of Hansen and Mladenović (2001) by introducing some extra features such as the addition of an empty route, use of a special data structure, a guided shaking strategy, the Dijkstra's post-optimiser, a diversification step and a learning strategy. In Stage 1, the improvements for all defined operators are found but only the best move is performed. The frequency of success of all the operators is recorded and this useful information is then used in Stage 2 to guide the search more effectively.

Stage 2 of the AVNS combines the idea of the multi-level composite heuristic (Salhi & Sari, 1997) with the VNS (Hansen & Mladenović, 2001). In other words, the multi-level k^{th} improvement approach acts as the local search in the VNS. In addition, we adopt a scheme that sits between the best and the first improvement strategies which we define as the k^{th} improvement. The main difference between the two stages of the AVNS algorithm is in the local search. In Stage 2, a small number of the local search operators is selected pseudo-randomly at each iteration based on their respective success rate found in Stage 1.

2.1. Stage 1 of the AVNS

The AVNS algorithm starts by defining the initial configurations which include a set of neighbourhood structures S_j, $j = 1, \ldots, p_{max}$, local search operators L_q, $q = 1, \ldots, q_{max}$, the scores and diversification parameters. An initial solution is generated and used as the global best, x_{best}. The data structure is also defined in this step.

In Step 1, a dummy empty route is added to allow for more flexibility in the search. The core steps of the VNS are stated in Step 2 and Step 3. In the shaking step (Step 3a), a random solution x' is generated from $S_1(x)$ using the guided shaking strategy which will be discussed in Section 3.7, and then improved by the best improvement procedures iteratively to obtain x'' (Step 3b). The local search strategy is applied until there is no more improvement. If the local minimum x'' is better than the incumbent best solution x, we set $x = x''$ and the search reverts back to the first neighbourhood S_1, otherwise the search will explore the next neighbourhood (i.e. $p = p + 1$) where Step 3 is then repeated. This process continues until the number of neighbourhood structures reaches p_{max}. At the end of Step 3b, the score for each operator is computed using Eq. (1), by adding the current score to the marginal relative gain. Other operators, though their corresponding best moves may not be as good as the overall best move, are also worth considering since only positive gains are recorded. Therefore, this ratio provides a more appropriate measure while not favouring exclusively

the best operator only. In addition, a data structure is embedded in Step 3 to accelerate the search, in which the details are provided in Section 3.3.

After all the neighbourhoods are evaluated, a post-optimiser using the Dijkstra's algorithm is implemented as indicated in Step 4. In this step, the incumbent best solution is used to construct a giant tour by combining the obtained routes. Then, a directed cost network is constructed and Dijkstra's algorithm is applied to obtain a new solution $\hat{x}$. Note that $\hat{x}$ is either unchanged or better than the incumbent solution (see Section 3.8). If $\hat{x}$ is better than the incumbent solution x, we set $x = \hat{x}$ and the search reverts back again to S_1.

In Step 5, if the incumbent x is better than the global best, x_{best}, we set $x_{best} = x$ and $\kappa = \kappa_{min}$. The diversification control parameter, κ is used to guide the diversification process. Here, the large neighbourhood search is adopted to create a new perturbed solution from which the process reverts back to Step 2. The details of Step 5 will be revisited in Section 3.9. The search terminates when a predefined maximum number of diversifications is reached. In Step 6, the probability and cumulative probability for each operator are computed using Eq. (2), in which the information is used in Stage 2 for the selection of the operators.

2.2. Stage 2 of the AVNS

In Step 7 and Step 8, the parameters are initialized. The number of operators to be selected, m_{max} is chosen in the range of $[l_{cmin}, l_{cmax}]$ in Step 9a. This number can be critical to the success of the search. The idea is to get a small enough number that requires a reasonable computational time while retaining the solution qual-

Stage 1

Step 0 *Initialisation*. Define a set of neighbourhood structures S_p, for $p = 1, ..., p_{max}$ and a set of local search operators L_q, for $q = 1, ..., q_{max}$. Initialise $Score(L_q) = 0$, diversification control parameter, $\kappa = \kappa_{min}$ and $numDiv = 1$. Define the maximum number of diversifications, $MaxDiv$. Generate an initial solution x, set $x_{best} = x$ and define the data structure.

Step 1 Add an empty route in the initial solution.

Step 2 Set $p \leftarrow 1$.

Step 3 *Repeat* the following until $p = p_{max}$:

(a) *Shaking:* Generate a solution x' at random from the p^{th} neighbourhood of $x(x' \in S_p(x))$ using the guided shaking strategy.

(b) *Best improvement local search:* For each operator L_q, find the gain, $Gain(L_q)$ using the best improvement strategy. Select the operator with the maximum gain and perform the move to obtain the best neighbouring solution x''.

Set

$$Score(L_q) = Score(L_q) + \frac{Gain(L_q)}{\max\limits_{1 \le h \le q_{max}} Gain(L_h)} \tag{1}$$

(c) *Move or not:* If the local minimum x'' is better than the incumbent x, set $x \leftarrow x''$ and go to Step 2, otherwise set $p \leftarrow p + 1$.

Step 4 Construct a giant tour and a directed cost network based on the incumbent best solution x and apply Dijkstra's algorithm to get $\hat{x}$. If $\hat{x}$ is better than the incumbent x, set $x \leftarrow \hat{x}$ and go to Step 2.

Step 5 If the solution x is better than x_{best}, set $x_{best} \leftarrow x$ and $\kappa \leftarrow \kappa_{min}$, else set $\kappa \leftarrow \kappa + 0.05N$;

If $numDiv > MaxDiv$, go to Step 6;

Diversification: Apply the large neighbourhood strategy based on κ to get a new solution $\tilde{x}$. If $\tilde{x}$ is better than x_{best}, set $x_{best} \leftarrow \tilde{x}$;

Add/drop the empty route(s) and go to Step 2.

Step 6 *Learning:* For each operator L_q, compute the probability, $Prob(L_q)$ and cumulative probability, $F(L_q)$ for $q = 1, ..., q_{max}$

$$Prob(L_q) = \frac{Score(L_q)}{\sum\limits_{q=1}^{q_{max}} Score(L_q)}, \qquad F(L_q) = \sum_{L_q \le L_{q_{max}}} Prob(L_q) \tag{2}$$

Stage 2

Step 7 Initialise $\kappa = \kappa_{min}$ and $nonImproveDiv = 0$. Define l_{cmin}, l_{cmax} and maximum number of unimproved diversifications, $MaxDiv2$.

Step 8 Set $p \leftarrow 1$.

Fig. 1. An adaptive VNS algorithm.

Step 9 *Repeat* the following until $p = p_{max}$:

(a) *Operator selection from learning:* Generate a random number $m_{max} \in [l_{cmin}, l_{cmax}]$ and set $m = 1$. Repeat the following until $m = m_{max}$:

(i) Generate $\alpha \in (0, 1)$ and compute $\hat{L}_q = F^{-1}(\alpha)$ with F defined in Equation (2).

(ii) Pick the operator based on $\hat{L}_q$, set $R_m \leftarrow \hat{L}_q$, and $m \leftarrow m + 1$.

Sort the operator R_m from simpler to complex for $m = 1, ..., m_{max}$

(b) *Shaking:* Generate a solution x' at random from the k^{th} neighbourhood of $x(x' \in S_p(x))$ using the guided shaking strategy.

(c) *Multi-level k^{th} improvement local search:*

(i) Set $m \leftarrow 1$.

(ii) Repeat the following until $m = m_{max}$:

Apply R_m to get the solution x'', if x'' is better than x', set $x' \leftarrow x''$, and go to Step 9c(i), otherwise set $m \leftarrow m + 1$.

(d) *Move or not:* If the local minimum x' is better than the incumbent x, set $x \leftarrow x'$ and go to Step 8, otherwise set $p \leftarrow p + 1$.

Step 10 Construct a giant tour and a directed cost network based on the incumbent best solution x and apply Dijkstra's algorithm to get $\hat{x}$. If $\hat{x}$ is better than the incumbent x, set $x \leftarrow \hat{x}$ and go to Step 8.

Step 11 If the solution x is better than x_{best}, set $x_{best} \leftarrow x$, $\kappa \leftarrow \kappa_{min}$, and $nonImproveDiv = 0$, else set $\kappa \leftarrow \kappa + 0.05N$ and $nonImproveDiv \leftarrow nonImproveDiv + 1$;

If $nonImproveDiv > MaxDiv2$, stop;

Diversification: Apply the large neighbourhood strategy based on κ to get a new solution $\tilde{x}$. If $\tilde{x}$ is better than x_{best}, set $x_{best} \leftarrow \tilde{x}$;

Add/drop the empty route(s) and go to Step 8.

Fig. 1. (*Continued*).

ity. Empirical results show that $[l_{cmin}, l_{cmax}] = [3, 5]$ is appropriate. The AVNS algorithm is adaptive in the sense that m_{max} is not fixed beforehand but randomly selected in the above range as shown in Step 9a. Besides, the choice of the operator at any iteration is carried out pseudo-randomly (see Step 9a(i)–Step 9a(ii)). In other words, the higher the score for an operator is (Eq. (1)), the greater the chance for such an operator to be chosen.

After selecting a set of local search operators, R_m $(m = 1, \ldots, m_{max})$ and sorting them based on the complexity of the moves, as defined in Section 3.8, the guided shaking step with the same neighbourhood structures defined in Step 0 is performed (Step 9b). Next, the multi-level k^{th} improvement local search is applied, as presented in Step 9. Initially, we set $m = 1$ and apply the first level of refinement with the local search R_1. If the solution x'' found is better than x', we set $x' = x''$ and the search reverts back to $m = 1$, otherwise we proceed to the next level by setting $m = m + 1$. Note that x'' could be found as the k^{th} improved solution or the best improved one if k is not reached. The record of information that we discuss in Section 3.4 is therefore based on this aspect.

The operators are sorted such that the operator in Level 1 is the simplest, followed by Level 2 which is the second simplest until Level m_{max} the most complex one. Therefore when the solution has changed, it is worth reverting back and applying the search in Level 1 which is relatively less computationally expensive. When the current level could not improve the solution, we use the operator in the next level to refine the solution. This process continues until m reaches m_{max}. Similar to Stage 1, the Dijkstra's post-optimiser is performed in Step 10. The LNS which will be described in Section 3.9 is used as the diversification strategy in Step 11. The search stops when there is no improvement on the solution after a maximum number of consecutive diversifications.

3. Explanation of the main steps

3.1. Initial solution

The saving method (Clarke & Wright, 1964) is adopted to generate the initial solution. The initial solution is then refined using 2-opt (Lin, 1965) and 2-opt* (Potvin & Rousseau, 1995) before proceeding to the AVNS.

3.2. Neighbourhood reduction

In this study, a neighbourhood reduction scheme is developed and used throughout the AVNS algorithm with the aim to avoid performing unnecessary calculations. This is achieved by identifying the customers that could be potentially placed next to each other in a route hence restricting the search to those customers only (Salhi & Sari, 1997). There are two types of possible moves for a customer: inserting the customer (i) between a pair of customers, or (ii) between a customer and the depot. We propose two logical matrices, $flag1, flag2 \in \mathbb{R}^{(N+1)\times(N+1)}$ for these two moves respectively. More specifically, *flag*2 considers a larger set of potential customers compared to *flag*1 because some customers are worth considering when the vehicle is en-route to and from the depot, even they are not in close proximity to the depot. In addition, we set *flag*1(*i, j*) and *flag*2(*i, j*) 'true' if customers i and j are potential customers for local search move (i) and (ii) respectively, and 'false' otherwise.

To assign values to these logical matrices, we propose two new criteria and also adapt the one given in Salhi and Sari (1997). The first criterion is devoted to *flag*1 and the last two are designed for *flag*2. Using these three criteria is necessary because Criterion 1 is solely based on the distance between the customers and therefore

Step 1 Find the average distance of all customers from the depot, $\bar{d}_0 = \frac{\sum_{i=1}^{N} d_{0i}}{N}$.

Step 2 Determine the set of customers near the depot, as $N_0 = \{i = 1, ..., N \mid d_{0i} < \bar{d}_0\}$.

Step 3 For each customer i,

(a) compute the insertion of j between i and the depot 0, $\delta_{ij}^0 = d_{ij} + d_{0j} - d_{0i}, \forall j \in N_0$;

(b) determine the average cost of insertion, $\hat{\delta}_i = \frac{\sum_{j \in N_0} \delta_{ij}^0}{|N_0|}$.

(c) For all $j \in N_0$:

If $\delta_{ij}^0 < \hat{\delta}_i$, set flag2 (i, j) = TRUE (i.e. the pair (i, j) cannot be excluded for possible moves)

Fig. 2. Criterion 2 – potential insertion of customer j between customer i and the depot.

Step 1 Compute the angle, $\gamma(i, 0, j)$

Step 2 Case 1: If $\gamma(i, 0, j) \leq \theta_{min}$

flag2 (i, j) = TRUE

Case 2: else

If $\gamma(i, 0, j) \leq \theta_{max}$

If ($d_{0i} \leq \bar{d}_0$ and $d_{0j} \leq \bar{d}_0$) or ($d_{0i} \leq \frac{d_{0j}}{2}$ or $d_{0j} \leq \frac{d_{0i}}{2}$)

flag2 (i, j) = TRUE

Fig. 3. Criterion 3 – angle between the depot and customers.

the percentage of the customers flagged can be relatively small where the customers that are far from the depot are excluded in this case. Criterion 2 and Criterion 3, which consist of a larger sets of potential customers are proposed to complement Criterion 1 and they are only used for the evaluation of move (ii). The percentages of flagged customers using these criteria are shown in Section 4.1.

Criterion 1: Distance between customers

We propose the qth quantile method where for each customer i, the top qth quantile nearest customers to i are assigned a 'true' value. With this, each customer has an approximate equal number of flagged neighbours. From a series of experiments conducted using different q values, it is found that $q = 3\%$ is sufficient. An example of the flagged customers using this criterion is illustrated in Fig. 4(a). Note that *flag*1(i, j) is 'false' using Criterion 1 because customer i and j are distant from each other.

Criterion 2: Distance between the depot and customers

When a vehicle is departing from or arriving at the depot, some customers can be worth assigning, though they may not necessary be located within close proximity to the depot. This second criterion is based on the insertion of customer j between customer i and the depot. This simple mechanism is presented in Fig. 2. Fig. 4(b) illustrates that *flag*2(i, j) is 'true' based on Criterion 2 because customer j is worth checking for the insertion between the depot and customer i when the vehicle is en-route returning to the depot, which may result in a reduced cost. This shows that Criterion 2 is necessary to be added as such an insertion will have been excluded for evaluation using Criterion 1 otherwise.

Criterion 3: Angle between the depot and customers

In addition to Criterion 2, the angle between the customers with respect to the depot, $\gamma(i, 0, j)$ is also taken into account when defining *flag*2 (Salhi & Sari, 1997). If customers i and j are located within a certain angle from the depot, they should not be excluded for evaluation in subsequent moves. The steps of this criterion are displayed in Fig. 3 and an illustration is shown in Fig. 4(c) with guaranteed minimum angle θ_{min} and maximum threshold angle θ_{max}. In the example, Criterion 3 is complementary to Criterion 2 because *flag*2(i, j) is already 'true' using Criterion 2. Preliminary experiments show that $\theta_{min} = \pi/12$ and $\theta_{max} = \pi/6$ are sufficient for this reduction test to be useful.

3.3. A special data structure in AVNS Stage 1 (Step 3 in Fig. 1)

To speed up the local search process, a special memory structure that is initially proposed by Osman and Salhi (1996) is adapted by introducing several new elements. The objective is to make use of the previously computed information and update the memory when necessary. A similar idea is also put forward by Zachariadis and Kiranoudis (2010) through the use of static move descriptor (SMD) to reduce the computational complexity of the local search. In their approach, each operator is represented in a different SMD matrix defined by its move point and rule. Each time when the best admissible move is performed, the SMD matrices are updated. However, one drawback of this method is the requirement of a large computer memory if a large number of local search operators is used.

The use of data structure is integrated in Step 3 in Fig. 1, which is motivated by the idea that not every route changes when a move is performed, therefore the subsequent search will only be evaluated based on the affected routes. There are two levels of memory structures that need monitoring. In the first level, the cost of removing each customer, $Z \in \mathbb{R}^N$ (N being the total number of customers) is stored. Z is used for all operators when a single customer is removed from a route, such as the 1-insertion and 1-1 exchange (see Section 3.8). When customer i has changed its position, the value of Z_i and the adjacent customers will also be updated.

In the second level, the set of all the best moves are stored in three matrices, namely the gain matrix, $G \in \mathbb{R}^{(r \times r)}$, the inter-route information matrix, $I \in \mathbb{R}^{(\frac{r^2-r}{2} \times a)}$, and the intra-route information matrix, $I_2 \in \mathbb{R}^{(r \times a)}$ (see Fig. 5) where r is the number of routes in the solution and a is the number of attributes. In this study, we use a value of ten for a, indicating that ten attributes are to be saved

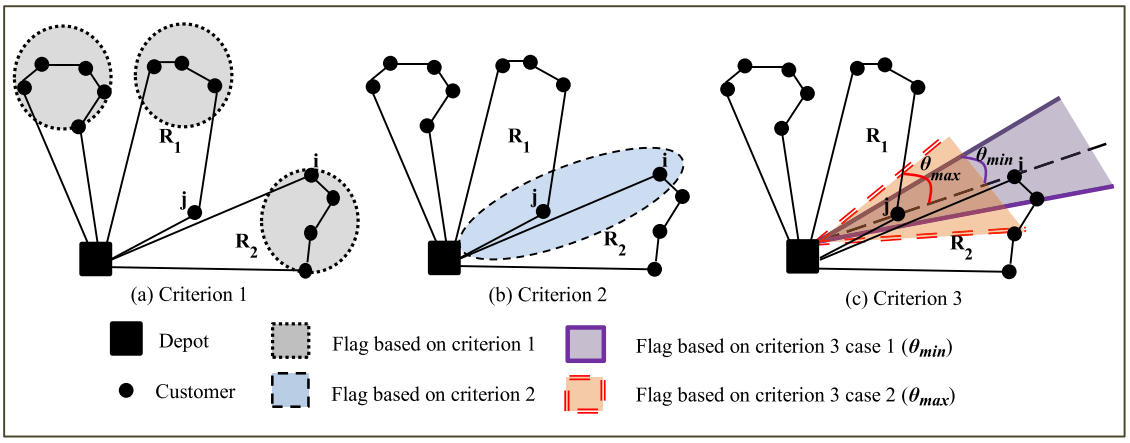

Fig. 4. An illustration of the neighbourhood reduction scheme using three criteria.

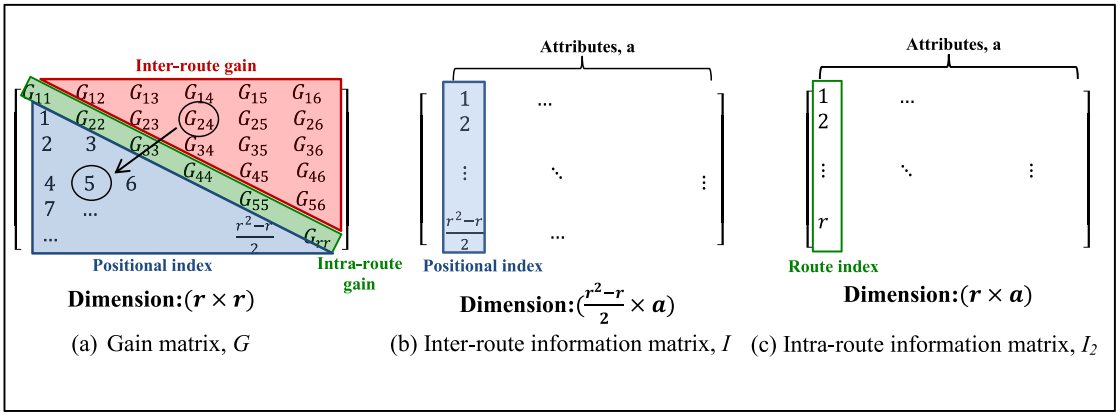

Fig. 5. Data structure consists of the gain matrix, G and information matrices, I and I_2.

in both I and I_2. The value of a can be easily increased if more information need to be stored in the matrices. It is worth noting that in the original study of Osman and Salhi (1996), only the gain matrix and the inter-route information matrix of six attributes are considered. However, here we introduce three matrices to store the information of the moves as compared to Zachariadis and Kiranoudis (2010) where the number of matrices adopted depends on the number of local search operators which can be relatively large. While the SMD uses a set of move points and rules to define the matrices dimension, our information stored in the matrices is route-based. Each time after performing the best move, there are at most two routes that can be changed whereas the other routes remain intact and hence do not need to be re-evaluated in the subsequent iteration.

In our gain matrix G, the upper triangular matrix (G_{jk}) records the best gain for each pair of inter-route move, the diagonal matrix (G_{jj}) for the intra-route move whereas the lower triangular matrix (G_{kj}) is assigned a positional index that is used to refer to the position in I for any pair of routes $j, k = 1, \ldots, r$, as illustrated in Fig. 5(a) and Fig. 5(b). The positional index, G_{kj} is defined as: $G_{kj} = j + ((k-1)(k-2)/2)$, where $j < k$. For instance, the gain between route 2 and route 4 is given in G_{24} which is obtained as the best gain that can be found based on all the operators (i.e. 1-insertion, 1-1,...). The corresponding positional index is $G_{42} = 2 + (3(2)/2) = 5$ as shown in Fig. 5(a).

For the intra-route move, G_{jj} does not have the corresponding positional index assigned to it, therefore I_2 (Fig. 5(c)) is created to record this information. The information matrices are used to store all the information needed for the move with the ten attributes defined as follows: (i) positional index (for I) or route index (for I_2), (ii) gain, G_{jk}, (iii) route j, (iv) route k, (v) position s (from route j), (vi) position t (to route k), (vii) position u (from route k), (viii) position v (to route j), (ix) type of operator and (x) reverse status. Note that I_2 acts exactly the same like I except that the attributes (i), (iii) and (iv) contain the same value referring to the route index. In this study, the data structure is enhanced by having an extra intra-route information matrix in addition to the recording of more information that is needed for further evaluating a given move.

The data structure mechanism is given in Fig. 6. At the first iteration, all intra and inter-route moves are evaluated and the best moves are stored in the matrices G, I and I_2 (Step 2 in Fig. 6). In Step 4 of Fig. 6, the best move with the highest gain is selected from the upper diagonal (for inter-route moves) or the diagonal matrix (for intra-route moves) of G. This move is then performed based on the information recorded in I or I_2 (Step 5 in Fig. 6). Next, the gains for the routes that have changed are initialized to zero (Step 6 in Fig. 6), resulting in recomputing the new gains for these routes. When the search returns to Step 2 of Fig. 6, only $2 \times (r-1)$ pairs of inter-route and two pairs of intra-route need to be updated instead of evaluating all the $r(r-1)/2$ pair of routes. The computational effort saved using the data structure is presented in Section 4.1.

3.4. The record of useful information in AVNS Stage 2 (Step 9 in Fig. 1)

The local search used in Stage 2 is the multi-level with the k^{th} improvement strategy. In any level of the local searches if no better move is found, it means a complete search is performed on all

Step 1 Initialize $G_{jj} = 0, G_{jk} = 0$ and $G_{kj} = j + ((k-1)(k-2)/2)$ for $j = 1, ..., r$, $k = j+1, ..., r$. Define $v, w = 1, ..., r$.

Step 2 For each local search operator, find the gain from the routes if one of the routes involves v or w and record it in the upper (inter-route) or diagonal matrix (intra-route) of G. Based on the positional index in the lower diagonal matrix of G, record the attributes of the move in I or I_2 (using the positional index as the row number).

Step 3 Initialize $gain = 0$

Step 4 From the upper diagonal and diagonal matrix $G_{jk}, j \leq k$, find the maximum gain and set $gain = G_{vw}$, where v and w refer to the route numbers associated with $gain$. The corresponding positional index is defined as:

$$P = \begin{cases} G_{wv} & \text{if } v < w \quad \text{(i)} \\ G_{vw} & \text{if } v > w \quad \text{(ii)} \\ v & \text{otherwise} \quad \text{(iii)} \end{cases}$$

Step 5 If $gain \leq 0$, stop.
Else, using P as the row index in I (for (i) and (ii)) or I_2 (for (iii)), access the related attributes recorded and perform the move.

Step 6 Initialize the upper diagonal and diagonal matrix $G_{vt} = 0$ if $v \leq t$; $G_{wt} = 0$ if $w \leq t$; $G_{tv} = 0$ if $v \geq t$; $G_{tw} = 0$ if $w \geq t$ for $t = 1, ...r$

Step 7 Go to Step 2.

Fig. 6. Data structure mechanism.

routes using that operator. Therefore in the subsequent iteration, the search only needs to be performed on the routes that have been modified. With this respect, we record the complete search status for the operator based on each pair of routes. We use a three-dimensional logical matrix, $B \in \mathbb{R}^{(m_{max} \times r \times r)}$ to represent the status where m_{max} is the number of operators and r is the number of routes.

Initially, the values of B_{mjk} are 'false' for $m = 1, \ldots, m_{max}$, and $j, k = 1, \ldots, r$. There are three situations that can be encountered when performing the search using an operator d on a pair of routes e and f:

(i) an improvement is found but the move is not performed because it is not the best among all the k improvements,
(ii) an improvement is found and the move is performed, and
(iii) no improvement is found.

For situation (i), the value of B_{def} remains 'false'. In (ii), we set B_{mef} = 'false' for $m = 1, \ldots, m_{max}$, therefore every operator will re-evaluate the pair of routes that have been changed in the subsequent iterations. As for (iii), B_{def} = 'true' which implies that operator d will not have to re-check routes e and f in the next iteration.

3.5. Penalized objective function

In this study, the algorithm only considers moves that can improve the current solution. To allow more flexibility, we extend the definition of improvement moves to both feasible and infeasible moves by using the idea of penalized objective function. In other words, the algorithm accepts infeasible improvement move but each violated route is penalized. For each route k (where $k = 1, \ldots, r$), let d_k be the route length, q_k the total route demand, η_c^k the unit penalized cost for capacity and η_d^k the unit penalized cost for distance. The route cost is defined by Eq. (3) whereas the penalized costs are given in Eqs. (4) and (5) with Q and D representing the capacity and distance constraint respectively. The penalized value, ζ is controlled by two user-defined parameters, β and γ. β is defined as the percentage of the capacity or distance violation from the maximum allowed, while γ is the maximum percentage of increase allowed in the route cost. Finally, ζ can be obtained using Eq. (7). Note that for instances with no distance constraint, Eq. (7) is stated as $\zeta \leq \gamma d_k/\beta$. Preliminary experiments show that $\beta = 0.05$ and $\gamma = 0.10$ are appropriate. The effect of using the penalized objective function is discussed in Section 4.2.

$$c_k = d_k + \eta_c^k + \eta_d^k \tag{3}$$

$$\eta_c^k = max\left(0, \frac{q_k - Q}{Q} \times \zeta\right) \qquad q_k \leq (1+\beta)Q \tag{4}$$

$$\eta_d^k = max\left(0, \frac{d_k - D}{D} \times \zeta\right) \qquad d_k \leq (1+\beta)D \tag{5}$$

$$d_k + \eta_c^k + \eta_d^k \leq (1+\gamma)d_k \tag{6}$$

$$\zeta \leq \gamma d_k/(2\beta) \tag{7}$$

3.6. Neighbourhood structures

We examine five inter-route neighbourhood structures that are carried out in the following order. N_1: 2-insertion*; N_2: 2-1 interchange; N_3: 2-1 interchange*; N_4: 2-2 swap and N_5: cross exchange. All the moves are conducted by randomly choosing a route (donor route) and the customer(s) and then moving the customer(s) to other route(s) (receiver routes) by considering the positions in the receiver routes systematically until the first feasible move is found.

2-insertion* This neighbourhood involves three routes where two consecutive customers are taken from a randomly selected donor route, and each of these customers is inserted into two different receiver routes.

2-1 exchange and 2-1 interchange* The 2-1 exchange operator attempts to exchange two consecutive customers from the

donor route with one customer from the receiver route. An extension of this operator is by considering three routes simultaneously, by exchanging one of the two customers from the donor route with a customer from the first receiver route and inserting the other customer into the second receiver route, leading to the 2-1 interchange*.

2-2 swap In the 2-2 swap, a pair of consecutive customers from the donor route is swapped with another pair of consecutive customers from the receiver route.

Cross-exchange This neighbourhood is also known as the block exchange where a segment of m_1 customers in the donor route is exchanged with a segment of m_2 customers from the receiver route. The values of m_1 and m_2 are randomly generated between the sizes of 3 and 5.

3.7. A guided shaking strategy (Step 3a and Step 9b in Fig. 1)

In the basic VNS, the change of neighbourhood is performed based on a randomly generated point. However, to avoid getting a poor quality solution that can subsequently lead to a longer time, we propose the following guided shaking based on the route location. Firstly, the centre of gravity, G_k for route k, is computed as follows:

$$G_k = (X_k, Y_k) = \left(\frac{\sum_{j \in R_k \cup \{0\}} x_j}{N_k + 1}, \frac{\sum_{j \in R_k \cup \{0\}} y_j}{N_k + 1} \right);$$

$k = 1, \ldots, r$

$r =$ number of routes

$R_k =$ set of customers in route k; $k = 1, \ldots, r$

$N_k =$ number of customers in route k; $k = 1, \ldots, r$ (i.e. $N_k = |R_k|$)

$(x_j, y_j) =$ coordinate for customer j; $j = 0, \ldots, N_k (j = 0$ is the *depot*)

A customer is randomly selected from one of the routes and its distances to the centre of gravity of the routes are computed and sorted in ascending order. Then, the first nearest route from the sorted list is chosen as the receiver route. This method is modified from a recent paper by Polat et al. (2015) where the shaking is performed using the route-first strategy (i.e. choosing the route pair that is in close proximity to each other). However, our version applies the customer-first mechanism where the customer is first selected followed by the route selection favouring the one nearer to the customer.

3.8. Local search engine

The local search engine consists of six operators, which include: (i) 1-insertion (ii) 1-1 exchange, (iii) 2-insertion, (iv) 2-opt, (v) 2-opt* and (vi) Cross-tail. The operators (i) - (iii) are performed for both intra and inter-routes whereas the operators (iv) - (vi) are for inter-routes. In the local searches, we first evaluate the moves for each customer to every position systematically and eventually choose the one with the overall best improvement (for Stage 1 AVNS) or the best value among k improvements (for Stage 2 AVNS).

1-insertion In the 1-insertion, we remove a customer from its position and relocate it elsewhere in order to have a better solution.

1-1 exchange In the 1-1 exchange intra-route, two customers' positions are swapped in the same route. For the inter-route procedure, we remove a customer from one route and assign it to other route and vice versa. Note that the positions of removal and insertion for inter-route 1-1 are not necessarily the same.

2-insertion For the 2-insertion operator, two consecutive customers are removed from a route and inserted either in the same route or in a different route, following the original or the inverted sequence of the customers.

2-opt The 2-opt was initially proposed by Lin (1965), where two non-adjacent edges are removed and replaced with two new edges in the same route.

2-opt* The 2-opt* is similar to 2-opt except that the operator is applied for a pair of routes instead of one route. This operator is simple but effective at removing crossed-edges between the routes.

Cross-tail This operator works by considering all possible sizes of the tails (see Fig. 7) of a route and exchanging it with the tails that are not necessarily of the same size from the other route. The size of the tails can be critical. If one tail is empty and the other tail consists of the entire route, this reduces to a merge if the move is feasible resulting in a reduced cost due to triangular inequality (see Fig. 7(c)). In other words, this scheme has the potential in reducing the total number of routes. This local search is originally proposed by Jun and Kim (2012) but in this study, a new feature is added by considering the original and the reverse order of the tails. In addition, whenever an iteration is finished (Step 5 and Step 11 in Fig. 1), the solution x_{best} is copied such that the order of the customers in some randomly chosen routes is reversed. By doing this, the cross-tail operator has the chance to consider the head-tail and head-head exchange between two routes and vice-versa.

Use of the Dijkstra's algorithm as a post-optimiser

In Step 4 and Step 10 in Fig. 1, the Dijkstra's algorithm is used as the post-optimiser to improve the incumbent best solution. To do this, we first construct a giant tour by connecting all the nearest endpoints of the routes from the incumbent solution. Then, a directed cost network is constructed such that each arc represents a feasible route. The cost of arc ij is expressed as follows:

$$C_{ij} = d_{0,i+1} + \sum_{k=i+1}^{j-1} d_{k,k+1} + d_{j,0}$$

This is followed by applying the Dijkstra's algorithm to find the optimal partition of the giant tour. This special post-optimiser aims to either improve the solution or guarantee that the current solution cannot be improved based on the optimal partitioning of the new constructed directed network. Also this refinement is used only when the incumbent best solution is obtained. For more details of this approach, the reader may find the paper by Imran et al. (2009) to be useful and informative.

3.9. A LNS diversification strategy with a VNS structure (step 5 and step 11 in Fig. 1)

In Step 5 and Step 11 of the AVNS, a diversification strategy is performed after all the neighbourhoods are explored. Such a strategy is considered as a powerful shaking procedure which aims to explore other promising regions that may not have been visited otherwise. Here, the perturbation intensity needs to be carefully monitored so that it is effective enough to guide the search toward escaping from the local optimum but not degrading the solution to a random restart point. To achieve this compromise, we opt for the large neighbourhood search (LNS) that follows a VNS structure.

In the LNS diversification, a control parameter, κ that represents the number of customers to be deleted and reinserted to the routes is used to guide the procedure. Note that a simple LNS that uses a random removal strategy reduces to 1- and 2-insertion (i.e. $\kappa = 1$ and 2). Hence, this procedure can be considered as a strong insertion by having large values of κ. Here we define κ in the range $[\kappa_{min}, \kappa_{max}]$ with $\kappa_{min} = \max(5, 0.05N)$ and $\kappa_{max} = \min(400, 0.4N)$. Initially $\kappa = \kappa_{min}$ and is gradually increased if no better solution is obtained so that a stronger diversification is applied in the next iteration. Once a new best

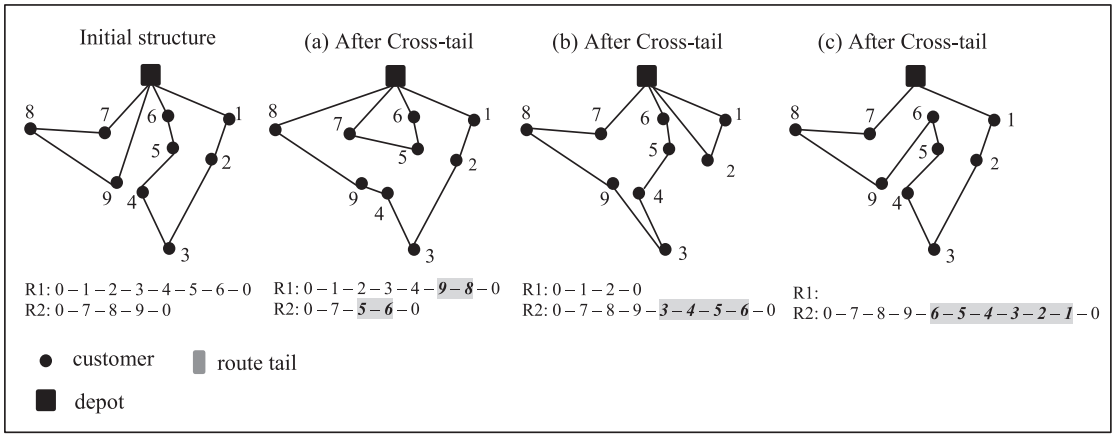

Fig. 7. The *Cross-tail* operator.

solution is found, it is initialized again to the minimum value, κ_{min}. This LNS implementation follows the VNS structure which, to the best of our knowledge, has not been formally explored so far. In addition, we propose four removal strategies including a new one (Removal strategy 4) alongside an effective insertion strategy.

Removal strategy 1: Gain ratio deletion

The first removal strategy adopted is based on the gain ratio obtained in deleting a particular customer. The ratio in deleting i is calculated as follows:

$$\text{ratio}(i) = \text{demand}(i)/\text{gain}(i)$$

More specifically, the gain(i) is the difference between the cost when customer i is in the solution and the cost when i is removed. The customers are then sorted in ascending order based on the ratio, and removed one-by-one starting from the smallest ratio, until the control value, κ is reached. This rule was initially proposed by Li, Golden, and Wasil (2005), which aims to remove the customer that has a small demand while saving a high profit (i.e reduced cost). The reasoning behind this choice is that it is a simple look-ahead strategy which allows more freedom at the insertion stage.

Removal strategy 2: Overlapping deletion

The removal strategy 2 is based on the number of overlaps in each route, where the overlap is defined by the number of intersections between the arcs in one route and the arcs in all other routes. This strategy, originally introduced by Jun and Kim (2012) starts by determining the number of overlaps for each route and sorting them in descending order. By selecting the route with the highest overlaps, the following steps are repeated until κ is reached: if the number of customers in the route is greater than κ, remove all customers and go to the next route, otherwise partially remove customers based on the gain ratio formula given in Removal Strategy 1.

Removal strategy 3: Worst-edge deletion

This strategy starts by identifying the distance of every arc in all the routes, and sort them in decreasing order. The customers that are connected by the longest arcs are removed. This removal scheme is used in Imran et al. (2009) to avoid considering giant tours with large arc costs.

Removal strategy 4: Conflicting sector deletion

This is a new removal strategy which we propose in this study. We first divide the route plane into different sectors based on the angle from the depot and identify the number of routes belongs to each sector. Then, we remove the customers in the sector that contains the most number of routes in it. The customers who belong to the same sector, are somehow neighbouring customers and hence are expected to be easier to reshuffle.

Initial experiments show that the angle used to define the sector with value $\pi/12$ produces good results. Note that when the angle is too large, there would be many routes lying in the same sector. For the case when there are two or more sectors having the same number of routes, we randomly select one of those sectors and remove the customers from the chosen sector. This method can be extended by considering the angle from the depot at different starting points clock-wisely and anti-clock-wisely.

Insertion strategy

In the repair phase, the basic greedy least-cost insertion is adopted. We let $\Delta f_{i,k}$ be the change of the objective value by inserting customer i into route k at the position that increases the objective value the least. If customer i cannot be inserted to route k, then we set $\Delta f_{i,k} = \infty$. For each unassigned customer i, we compute the cost of inserting i at its overall best position when checked over all routes. Finally, we choose the request that has a minimum cost of insertion and insert it at its corresponding best position. We repeat this process until all requests have been inserted into the route plan.

However, if a feasible insertion is not found for a customer, we proceed with the following steps:

(i) A new perturbation operator that will be discussed next is first applied.
(ii) If there is no feasible insertion found, we apply the variable neighbourhood descent (VND) strategy which consists of the 2-opt, 2-opt*, cross-tail and cross-exchange as the set of local searches to refine the partially constructed routes.
(iii) Eventually, if there is still no feasible insertion, an empty route is added. This generally happens when the routes have very tight constraints.

The new perturbation operator used in step (i) given above is adapted from Salhi and Rand (1987) where three routes are considered simultaneously. The process starts by considering a customer which is initially removed from a route and inserting it into another route, while withdrawing a customer from the second route and trying to put it into the third route. This is continued until a feasible insertion (irrespective of the cost of insertion) is found for both the initial customer considered and the customer removed from the second route.

4. Computational experiments

In this section, we present the details of the computational experiments carried out to test the AVNS algorithm. The proposed

Table 1
Comparison of the computational time and cost for the data structure and neighbourhood reduction scheme.

#(*N*)	Neighbour (%)		T_{avg} (C_{avg})			
			With NR		Without NR	
	Flag1	*Flag2*	With DS	Without DS	With DS	Without DS
C1(50)	2.92	26.07	0.89 (532.96)	1.56 (533.00)	2.06 (530.60)	3.59 (531.95)
C2(75)	3.50	27.32	1.32 (842.66)	2.04 (837.36)	3.68 (845.28)	5.65 (842.87)
C3(100)	3.96	27.64	1.54 (836.50)	2.99 (836.50)	4.68 (832.74)	6.87 (836.69)
C4(150)	3.46	27.50	2.87 (1037.67)	4.68 (1036.93)	6.25 (1037.60)	10.64 (1042.30)
C5(199)	3.84	27.42	3.04 (1315.44)	7.89 (1314.87)	15.96 (1312.21)	19.87 (1312.77)
G1(240)	3.45	26.69	5.06 (5653.39)	9.91 (5654.45)	23.54 (5652.37)	32.56 (5653.39)
G2(320)	3.23	26.80	7.64 (8485.38)	14.65 (8484.50)	38.65 (8477.20)	54.32 (8477.82)
G3(400)	3.54	27.08	9.98 (11045.80)	23.65 (11049.90)	47.65 (11052.80)	75.62 (11047.40)
G4(480)	3.34	26.98	12.87 (13641.10)	35.64 (13640.30)	70.65 (13639.30)	92.65 (13695.30)
L1(560)	3.57	27.01	14.96 (16248.60)	42.32 (16251.00)	83.21 (16268.80)	115.68 (16260.00)
L2(600)	3.49	26.76	18.65 (14667.10)	47.5 (14676.90)	98.7 (14682.60)	123.67 (14662.50)
L3(640)	3.45	27.09	17.53 (18853.70)	55.65 (18864.50)	105.65 (18833.50)	138.9 (18836.00)
L4(720)	3.40	27.10	22.56 (21440.60)	59.65 (21449.40)	116.73 (21459.40)	159.85 (21488.80)
L5(760)	3.54	26.83	27.98 (17066.40)	63.48 (17180.80)	129.98 (17094.70)	170.65 (17093.70)
L6(800)	3.48	27.10	30.21 (24063.80)	70.52 (24085.80)	145.62 (24053.60)	182.65 (24068.70)
L7(840)	3.42	26.74	36.56 (17690.90)	79.65 (17693.10)	155.38 (17706.30)	199.32 (17709.70)
L8(880)	3.48	27.24	33.05 (26638.60)	85.54 (26668.60)	162.32 (26653.40)	215.65 (26639.40)
L9(960)	3.52	27.24	35.64 (29253.50)	98.54 (29264.60)	182.12 (29305.40)	256.97 (29674.20)
L10(1040)	3.50	27.26	39.06 (31807.00)	121.32 (31899.40)	220.6 (31852.30)	298.64 (31865.40)
L11(1120)	3.45	27.27	42.32 (34465.60)	132.41 (34496.8)	243.65 (34498.50)	330.65 (34387.80)
L12(1200)	3.52	27.34	45.65 (37612.10)	158.92 (37580.00)	287.65 (37595.60)	389.21 (37590.00)
Avg	3.48	27.07	19.49 (15866.61)	53.26 (15880.89)	102.13 (15875.44)	137.31 (15891.27)

Neighbour (%): Average % of neighbour per customer, T_{avg}: Average computational time in minute, C_{avg}: Average cost

AVNS algorithm is coded in C++ and executed on a Pentium Core i7 3.4 GHz PC with 8 GB RAM. We first empirically investigate the effects of the data structures and neighbourhood reduction scheme in our algorithm. Then, we analyse the effectiveness of the penalized objective function as well as the performance of Stage 2. In addition, the use of learning in the selection of the local searches in Stage 2 and the effect of the diversification strategy are also evaluated. Finally, we perform a full experiment to assess the performance of our algorithm when compared to existing methods from the literature.

We use three benchmark data from the literature, namely small (Christofides & Eilon, 1969), medium (Golden et al., 1998) and large (Li et al., 2005) data sets, which we refer to as Set 1, Set 2 and Set 3 hereinafter. These data sets consist of 14, 20, and 12 instances respectively and they can for simplicity be downloaded from the website CLHO (2016). The instances in Set 1 are made up of 50–199 customers. Both C1–C5 and C6–C10, C11–C12 and C13–C14 are the same except that the first and third groups impose route-length and service time restriction. In addition, the customers are randomly distributed for instances C1–C10, and clustered otherwise. The size of the instances in Set 2 ranges from 240 to 483 customers, where route-length constraint is found in the first eight instances. Each problem exhibits a geometric structure: the customers in G1-G8 are located in concentric circle around the depot, G9–G12 in concentric square with the depot in the corner, G13–G16 in concentric square around the depot, and G16–G20 in a six-pointed star around the depot. For Set 3, there is a range of 560–1200 customers with route-length constraint. All instances establish a geometric structure with customers located in concentric circle around the depot.

For comparisons purpose, we present the best results found in some of the most effective VRP algorithms proposed in the literature. For each instance, we compute the relative percentage deviation as $((C_{Best} - C_{BKS})/C_{BKS}) \times 100$, where C_{Best} and C_{BKS} represent the best cost found using our algorithm and the best known result in the literature respectively.

4.1. Effects of the data structure and the neighbourhood reduction scheme

This subsection discusses the effects of using two aspects in the AVNS, namely: (i) the data structure and (ii) the neighbourhood reduction scheme. For (i), we apply the data structure (DS) to both stages in the AVNS and compare them against the ones without the DS. Similarly for (ii), we test the neighbourhood reduction (NR) scheme on a number of instances as well as without the scheme. For simplicity, we call the algorithm with NR and DS as Variant 1, with NR and without DS as Variant 2, without NR and with DS as Variant 3, and without NR and DS as Variant 4. The computational time and the objective function values are presented in Table 1 where three diversifications are used as the termination criterion for both Stage 1 and Stage 2. Such criterion is used to avoid the algorithm running for too long as the main purpose here is to evaluate the effect of the NR and DS. Preliminary experiments show that the effect of DS and NR is more significant when N is large, therefore we present the results for a sample of instances from Set 1 and Set 2 whereas for Set 3, all instances are included.

By using the neighbourhood reduction scheme, there is about 3.48% of neighbouring customers identified for each customer i (*Flag*1) and 27.07% of potential customers that can be inserted next to the depot (*Flag*2). In other words, only the neighbouring customers are evaluated for each relevant insertion. In Table 1, it is observed that the computational time for Variant 1 is the lowest comparing to other variants, where the effect of NR is more significant than DS. For example, without the use of DS, the average time for Variant 2 is about 2.7 times longer than Variant 1, whereas it is 5.2 times longer for Variant 3 if NR is not implemented in the algorithm. In addition, the effect is very significant when the size of the instances is large, which can be observed in the largest instance ($N = 1200$), the time spent on the algorithm with NR and DS is approximately 8.5 times shorter than without. Fig. 8 depicts the average time taken for the AVNS with and without the DS and NR. The figure clearly shows that the use of DS and NR resulting in a considerable decrease in computational time. On average, the

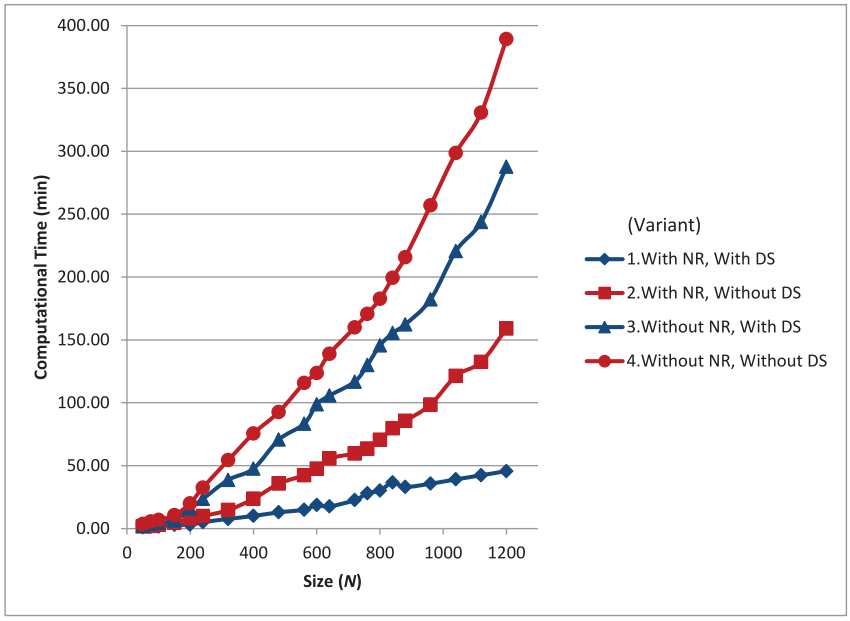

Fig. 8. Comparison of computational time for AVNS with and without the DS and NR.

Table 2
Best fits of computation time (T) against the problem size (N) for Variant 1–Variant 4.

Variant	Equation	Fitting	Adjusted R^2	p-value
1	$T = kN + c$	$T = 0.0479N - 10.0721$	0.94	2.23×10^{-7}
2	$T = kN^c$	$T = 0.0009N^{1.6894}$	0.98	2.03×10^{-10}
3	$T = kN^c$	$T = 0.0048N^{1.5444}$	0.99	2.58×10^{-11}
4	$T = kN^c$	$T = 0.0047N^{1.5880}$	0.99	2.17×10^{-11}

time is reduced by 35% when applying the data structure in Stage 1 and by 15% when the record of information is used in Stage 2.

In terms of the objective function values, the results are not very different for both with and without the strategies. This is due to the termination criterion used. When the algorithm is executed without NR and DS, it means that a large portion of the time is spent on "unnecesary searches". For example this includes searching on customers which are far apart from each other (without NR) and evaluating unchanged routes that have already been assessed in previous iterations (without DS). It is worth noting that another way of measuring the effect of the strategies in terms of the solution quality is by allowing the algorithm to run with the added strategies of DS and NR, and then recording the corresponding time spent. This can then be used as a stopping criterion when running the other algorithm without the added strategies and then record the objective values at that time.

The computational time for the four variants are plotted in graph as shown in Fig. 8. It is observed that the graphs for Variants 2, 3 and 4 tend to rise sharply with N. Therefore to further evaluate the effect of DS and NR on the large data set, we fit the computational time, T with the problem size, N using two models: the linear model ($T = kN + c$) and the power model ($T = kN^c$). For the power model, we first transform T_{avg} and N for instances L1–L12 in Table 1 to $ln\ T$ and $ln\ N$ respectively and then regress the equations using simple linear regression. Prior to that, all assumptions of the linear regression are checked and satisfied. The best fits for Variant 1–Variant 4 are presented in Table 2. The statistical results show that Variant 1 is best fitted with the linear model as indicated by a high adjusted R^2 value of 0.94, whereas Variants 2, 3 and 4 have very good fits with the power model with the respective adjusted R^2 values 0.98, 0.99 and 0.99. Therefore, it is reasonable to conclude that the computational time increases linearly with the problem size when the NR and DS are incorporated. This demonstrates that the use of the two components makes the search rather fast and stable as N increases, and hence these methods can be used for large instances.

4.2. Effects of Stage 2 and the penalized objective function in the AVNS

To evaluate the effect of Stage 2, we first run Stage 1 of the algorithm and record the computational time and cost, and then continue with Stage 2. The termination criterion for Stage 1 is set at four diversifications, whereas for Stage 2, four consecutive non-improved diversifications are used. The reason for this is the solutions tend to be stagnant after four diversifications in Stage 1, therefore we intend to proceed with Stage 2 which is relatively faster due to the selection of a group of local search operators only. In addition, the algorithm is executed with and without the use of penalized objective function in both stages.

It is worth noting that to assess the performance of the AVNS without Stage 2, we also conduct preliminary experiments by running Stage 1 only using the same amount of total time spent when running both stages. It is observed that there is an average of 0.5% gap in the cost comparing with the results obtained from running two stages. Therefore in this section, we present the results of the deviation in terms of cost and time from the initial solution to Stage 1, and from Stage 1 to Stage 2, as shown in Table 3.

On average, the cost is reduced by 3.53% after Stage 1 (without penalized function) and 3.80% (with penalized function) from the initial solution. With the use of about 61% of the total computational time, the cost decreases approximately 0.59% and 0.53% after Stage 2 without and with the penalized function respectively. The rate of decrease in Stage 2 is expected to be lower than Stage 1 because the solution is more stagnated and harder to improve in the latter stage.

It is observed that the use of the penalized objective function can generally improve the solution further. For example in Stage 1, the solution gap with the penalized function is 0.28% lower than those without, with an 11.84% increase in the computational time. In Stage 2, there is about 10.79% additional time spent with the

Table 3
Average solution gap and time gap after Stage 1 and Stage 2 with and without penalized objective function (in %).

Data	After Stage 1		After Stage 2			
	Without P	With P	Without P		With P	
	gap^n (I)	gap^p (I)	gap^n (S)	CPU^n (S)	gap^p (S)	CPU^p (S)
Set 1	3.39	3.82	0.69	64.71	0.45	63.22
Set 2	4.11	4.36	0.63	61.68	0.63	61.43
Set 3	2.74	2.86	0.40	58.89	0.45	58.92
Average	3.53	3.80	0.59	61.87	0.53	61.32
$gap^{S1,\,S2}$ (Pen)		0.28			0.23	
$CPU^{S1,\,S2}$ (Pen)		11.84			10.79	
$gap^{n,\,p}$ (BKS)			0.43		0.33	

n: without penalized objective function, p: with penalized objective function, $S1$: after Stage 1, $S2$: after Stage 2,
Let $k = \{n, p\}$ and $m = \{S1, S2\}$, C: Cost, T: Time,
$gap^k\ (I) = (C_{Initial} - C^k_{S1})/C^k_{Initial} \times 100$,
$gap^k\ (S) = (C^k_{S1} - C^k_{S2})/C^k_{S1} \times 100$, $CPU^k\ (S) = (T^k_{S2} - T^k_{S1})/T^k_{S2} \times 100$,
gap^k (BKS) $= (C^k_{Best} - C_{BKS})/C_{BKS} \times 100$,
gap^m (Pen) $= (C^n_m - C^p_m)/C^n_m \times 100$, gap^m (CPU) $= (T^p_m - T^n_m)/T^p_m \times 100$.

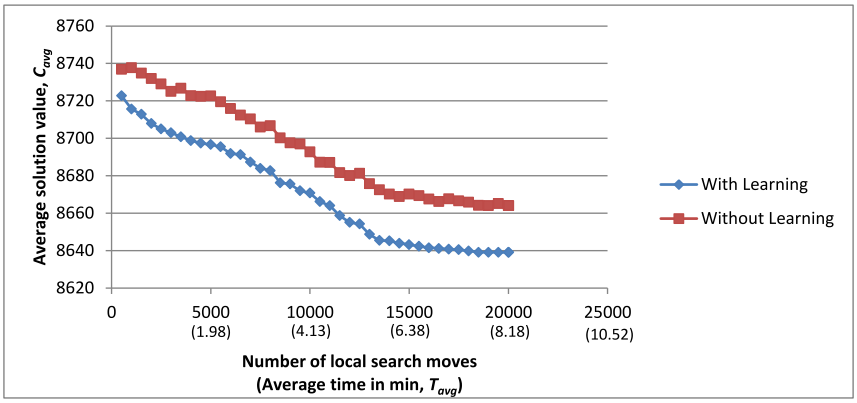

Fig. 9. Comparison of the average objective function values and number of moves in the AVNS Stage 2 (with learning and without learning).

used of penalized function but the solution is 0.23% better than the one with no penalized function. We also evaluate the gap of the best solution obtained using both with and without the penalized function comparing against the best known result published in the literature and found a smaller deviation (0.33%) for the former. This means that the use of penalized objective function is able to obtain better results with a reasonable amount of extra time.

4.3. Effects of the learning phase and the diversification strategy in the AVNS

To test the effect of learning in Stage 2, we compare the solutions obtained for Stage 2 using two variants: (i) with learning, and (ii) without learning, with the same termination criterion specified in Section 4.2. In the case of (i), a number of local search operators are selected based on the information recorded whereas for (ii), all operators are applied iteratively. Fig. 9 depicts the evolution of the average solution value and number of local search moves over all instances for (i) and (ii). The graph shows that the learning mechanism leads to a better convergence behavior and the overall solution quality. More specifically, the number of local searches required to reach a prescribed solution using (i) is about 1.5 times smaller than (ii). For example, to obtain an average solution value smaller than 8670, (i) needs less than 10,000 (on average 4.13 min) local search moves, whereas (ii) requires about 15,000 (on average 6.38 min). In other words, the integration of learning in Stage 2 improves the convergence rate of the solution as the intelligent selection mechanism only selects the operators that could potentially improve the solution.

Table 4
t-test comparison for the average objective function values for the AVNS Stage 2 (with learning and without learning).

	With learning	Without learning
Mean	8669.61	8693.88
Observations	41	41
Hypothesized mean difference	0	
P(T<=t) one-tail	< 0.01	
Kolmogorov–Smirnov P-value	< 0.01	< 0.01

The statistical t-test is conducted to test the difference between the mean of (i) and (ii) and the results are presented in Table 4. Before carrying out the t-test, the normality assumption is checked using the Kolmogorov–Smirnov test and a p-value < 0.01 is obtained which indicates the normality assumption is satisfied. It is shown in the t-test that there is a highly significant difference between the average solution for (i) and (ii) with a p-value < 0.001 (see Table 4).

The results of our computational studies show the high performance of the proposed AVNS. We also analyse here the effect of the diversification strategy used in our algorithm in both stages. With this, a number of instances are solved using the AVNS with diversification strategy as well as without the diversification. For consistency, we first record the time spent by the AVNS with diversification, and then allow the same time for the algorithm without diversification. Table 5 shows that our AVNS works very well with the diversification strategy in which the average solution obtained for the algorithm with diversification strategy is better than those

Table 5
Average solution obtained for the AVNS with diversification vs. without diversification strategy.

#(N)	C_{avg} (No Diversification)	C_{avg} (Diversification)	T_{avg}
C1(50)	527.98	524.61	1.09
C2(75)	843.92	835.26	1.65
C3(100)	831.98	828.65	1.96
C4(150)	1065.37	1032.64	3.12
C5(199)	1344.34	1305.30	3.67
G1(240)	5864.51	5645.90	8.05
G2(320)	8512.65	8465.45	9.68
G3(400)	11364.50	11038.60	11.23
G4(480)	14147.50	13639.30	15.65
L1(560)	16585.40	16232.40	18.69
L3(640)	19241.00	18820.30	23.65
L6(800)	25370.20	24032.50	33.69
L9(960)	29687.50	29192.30	42.65
L12(1200)	38465.40	37410.70	58.93
Average (overall)	12418.02	12071.71	
Average dev (%)	2.80	0.37	

without. The average deviation for the former (0.37%) is approximately 7.5 times lower than the latter (2.80%).

4.4. Comparisons with existing algorithms

This subsection presents the results of the AVNS algorithm against the other existing algorithms from the literature. In our AVNS, the number of diversifications as specified in Section 4.2 is used as the termination criterion. The algorithm is run for four diversifications in Stage 1 whereas in Stage 2, it is run until no improvement is obtained after four consecutive diversifications.

Tables 6, 7 and 8 report the best results obtained by the AVNS algorithm under 10 runs for each instance, comparing against the best published results from the literature for Set 1, Set 2 and Set 3 respectively. Our results are presented in the last four columns in the tables under AVNS. First, we present our best solution obtained in 10 runs and the individual percentage deviation with the C_{BKS} for each instance. To assess the robustness of our algorithm, the average solution and its deviation from the best solution are also presented. Those results that match with the best known solutions are given in boldfaced. In addition, the average percentage deviations, CPU times and computer specifications are also provided. It is worth noting that the best known results are extremely good which serve as a reliable guide for comparison purpose demonstrating the efficiency of any new proposed algorithm in this area.

For Set 1, we found 9 out of 14 solutions that are as good as the best known results. Other solutions are very close to the best values from the literature with a tiny average deviation of 0.08%. To the best of our knowledge, the optimal solution is reported only for 5 instances of Set 1 (C1-C3 and C11-C12) in the literature (Mingozzi, Roberti, & Toth, 2013), where these instances are marked by a '*' sign in Table 6. Our algorithm found optimal solution for all these five instances. The average running time of our algorithm in Set 1 is approximately 2.37 min which is reasonably competitive with other algorithms in the literature. As seen in Table 6, the AVNS has shown adequate stability, as indicated by a small gap of 0.21% between the average solution values obtained over the 10 runs with the best values.

In Table 7, it is shown that the average percentage deviation achieved by our algorithm for Set 2 is limited to 0.58%. Four of our results match the best known values whereas the rest are fairly close to the best results. Considering the individual relative deviations showing that the results obtained for instances G1–G8 are below 0.40%. However, the gap for instances G9–G20 is slightly higher, but still below 1% (about 0.91%). Among the three data sets, the AVNS seems to have the worst performance in Set 2, especially for G9–G20. This is probably due to the geometric structure established in these test instances. In terms of the computational time, our AVNS is very competitive with the others. With the incorporation of the neighbourhood reduction and the data structure mechanism, the CPU time for the relatively large instances (for example G4) is still within 15 min. In addition, the AVNS is fairly robust as the average gap reported between the best and the average solution is just 0.26%.

Table 8 displays the results for Set 3 where the average deviation of the AVNS is restricted to a satisfactory 0.19%. Our algorithm has obtained two best known solutions with one new result (L5) reported. In addition, the AVNS reaches nearly most of the best solutions with less than 0.04% deviation for each individual instance. The worst performance is recorded for test instance L12 with a 1.18% deviation from the best known solution. In terms of robustness, our AVNS yields satisfactory performance as indicated by a

Table 6
Results for Set 1 (Christofides & Eilon, 1969) instances.

#(N)	C_{BKS}	TK02	T05	MB07	P09	CHD10	AVNS			
							C_{Best}	$\%_{BKS}$	C_{avg}	$\%_{avg}$
C1(50)	*524.61	524.61	524.61	524.61	524.61	524.61	**524.61**	0.00	524.61	0.00
C2(75)	*835.26	835.26	835.26	835.26	835.26	835.26	**835.26**	0.00	835.26	0.00
C3(100)	*826.14	826.14	826.14	826.14	826.14	826.14	**826.14**	0.00	828.65	0.30
C4(150)	1028.42	1030.88	1028.42	1028.42	1029.48	1028.42	1031.07	0.26	1032.64	0.15
C5(199)	1291.29	1314.11	1311.48	1291.29	1294.09	1898.20	1291.45	0.01	1305.30	1.07
C6(50)	555.43	555.43	555.43	555.43	555.43	555.43	**555.43**	0.00	556.89	0.26
C7(75)	909.68	909.68	909.68	909.68	909.68	909.68	**909.68**	0.00	910.54	0.09
C8(100)	865.94	865.94	865.94	865.94	865.94	865.94	**865.94**	0.00	867.35	0.16
C9(150)	1162.55	1163.19	1162.55	1162.55	1162.55	1639.20	1164.89	0.20	1169.01	0.35
C10(199)	1395.85	1408.82	1407.21	1401.12	1401.46	2442.74	1402.32	0.46	1406.57	0.30
C11(120)	*1042.11	1042.11	1042.11	1042.11	1042.11	1042.11	**1042.11**	0.00	1042.11	0.00
C12(100)	*819.56	819.56	819.56	819.56	819.56	819.56	**819.56**	0.00	820.48	0.11
C13(120)	1541.14	1544.01	1544.01	1541.14	1545.43	1726.08	1543.16	0.13	1546.08	0.19
C14(100)	866.37	866.37	866.37	866.37	866.37	866.37	**866.37**	0.00	866.37	0.00
# Best sol		9	11	13	10	10	9			
Avg dev (%)		0.23	0.18	0.03	0.07	0.13		0.08		0.21
Avg CPU (min)		5.22	6.96	2.71	0.27	10.9	2.37			
Comp spec		0.40G	0.40G	2.80G	2.80G	2.93G	3.40G			

*optimal solution (Mingozzi et al., 2013), C_{BKS}: Best known solution, TK02 by Tarantilis and Kiranoudis (2002), T05: Tarantilis (2005), MB07: Mester and Bräysy (2007), P09: Prins (2009), CHD10: Chen et al. (2010)
$\%_{BKS} = (C_{Best} - C_{BKS})/C_{BKS} \times 100$, $\%_{avg} = (C_{avg} - C_{Best})/C_{Best} \times 100$

Table 7
Results for Set 2 (Golden et al., 1998) instances.

#(N)	C_{BKS}	MB07	N07	P09	CHD10	XZKM14	AVNS			
							C_{Best}	$\%_{BKS}$	C_{avg}	$\%_{avg}$
G1(240)	5626.81	5627.54	–	5644.52	5658.88	5626.81	**5626.81**	0.00	5645.90	0.34
G2(320)	8447.92	8447.92	–	8447.92	8464.82	8452.71	8460.93	0.15	8465.45	0.05
G3(400)	11036.22	11036.22	–	11036.22	11059.40	11036.22	**11036.22**	0.00	11038.60	0.02
G4(480)	13624.52	13624.52	–	13624.52	13624.52	13626.92	13628.10	0.03	13639.30	0.08
G5(200)	6460.98	6460.98	–	6460.98	6460.98	6460.98	**6460.98**	0.00	6460.98	0.00
G6(280)	8412.90	8412.90	–	8412.90	8412.90	8412.90	**8412.90**	0.00	8413.81	0.01
G7(360)	10181.75	10195.56	–	10195.59	10267.28	10195.59	10195.60	0.14	10198.40	0.03
G8(440)	11643.90	11663.55	–	11643.90	11756.85	11663.55	11688.90	0.39	11705.70	0.14
G9(255)	580.46	583.39	580.60	586.23	585.10	580.46	588.79	1.44	590.78	0.34
G10(323)	738.92	741.56	738.92	744.36	745.57	739.98	745.45	0.88	749.35	0.52
G11(399)	917.17	918.45	917.17	922.40	923.13	916.35	927.01	1.07	936.54	1.03
G12(483)	1107.19	1107.19	1108.48	1116.12	1117.93	1109.10	1118.36	1.01	1122.43	0.36
G13(252)	857.19	859.11	857.19	862.32	861.65	859.28	866.03	1.03	870.32	0.50
G14(320)	1080.55	1081.31	1080.55	1089.35	1088.87	1080.55	1087.58	0.65	1089.32	0.16
G15(396)	1340.24	1345.23	1340.24	1352.39	1356.32	1345.11	1356.18	1.19	1358.38	0.16
G16(480)	1622.69	1622.69	2171.30	1634.27	1637.13	1622.05	1638.48	0.97	1646.53	0.49
G17(240)	707.79	707.79	707.76	708.85	708.92	708.99	708.78	0.14	710.56	0.25
G18(300)	995.39	998.73	995.39	1002.15	1010.52	1000.58	1007.06	1.17	1010.83	0.37
G19(360)	1366.86	1366.86	1366.14	1371.67	1383.40	1369.41	1375.98	0.67	1378.32	0.17
G20(420)	1820.09	1820.09	1820.54	1830.98	1839.93	1823.75	1833.17	0.72	1837.09	0.21
# Best sol.		10	8	6	3	6	4			
Avg dev (%)		0.12	2.83	0.42	0.58	0.11		0.58		0.26
Avg CPU (min)		24.35	413.68	7.27	284.4	30.00	11.17			
Comp spec		2.80G	3.20G	2.80G	2.93G	1.60G	3.40G			

N07_b: Nagata (2007), XZKM14: Xiao et al. (2014)

Table 8
Results for Set 3 (Li et al., 2005) instances.

#(N)	C_{BKS}	K07_V	K07_G	MB07	PR07	XZKM14	AVNS			
							C_{Best}	$\%_{BKS}$	C_{avg}	$\%_{avg}$
L1(560)	16212.74	16602.99	16221.22	16212.74	16224.81	16214.03	16214.00	0.01	16232.40	0.11
L2(600)	14597.18	14651.27	14654.87	14597.18	14631.08	14618.18	14624.50	0.19	14658.60	0.23
L3(640)	18801.12	19005.37	18810.72	18801.12	18837.49	18801.86	18802.30	0.01	18820.30	0.10
L4(720)	21389.33	21784.43	21401.41	21389.33	21522.48	21391.83	**21389.33**	0.00	21432.00	0.20
L5(760)	16892.70	17151.43	17358.18	17095.27	16902.16	17012.86	**16892.70**	0.00	16979.10	0.49
L6(800)	23971.74	24189.66	23996.86	23971.74	24014.09	23981.33	23980.10	0.03	24032.50	0.22
L7(840)	17432.85	17823.40	18233.93	17488.74	17613.22	17432.85	17576.10	0.82	17686.30	0.63
L8(880)	26565.92	26606.11	26592.05	26565.92	26791.72	26567.24	26568.40	0.01	26594.60	0.10
L9(960)	29156.73	29181.21	29166.32	29160.33	29405.60	29156.73	29162.70	0.02	29192.30	0.10
L10(1040)	31742.51	31976.73	31805.28	31742.51	31968.33	31742.64	31754.60	0.04	31789.50	0.11
L11(1120)	34330.84	35369.17	34352.48	34330.84	34770.34	34332.14	34335.70	0.01	34366.60	0.09
L12(1200)	36928.70	37421.44	37025.37	36928.70	37377.35	37204.05	37363.20	1.18	37410.70	0.13
# Best sol.		0	0	9	0	2	2			
Avg dev (%)		0.85	0.72	0.13	0.61	0.14		0.19		0.21
Avg CPU (min)		0.04	0.13	104.29	497.9	150.30	36.81			
Comp spec		3.00G	3.00G	2.8G	3.00G	1.60G	3.40G			

K07_V and K07_G: Kytöjoki et al. (2007), PR07: Pisinger and Ropke (2007)

tiny gap of 0.21% between the average solution and the best solution values. In addition, the CPU time required for the large data set when NR and DS are used exhibits a linear pattern with the problem size N which demonstrates the efficiency of these mechanisms.

5. Conclusion

In this paper, we propose a novel and effective AVNS algorithm incorporating a LNS diversification strategy. The hybridisation of AVNS and LNS combines the good features of the two heuristics, hence making our algorithm very competitive. While most existing adaptive VNS published in the literature address the adaptive aspect on the change of neighbourhoods, our algorithm focuses on the adaptive selection in the local search engine instead. It is shown that the selection of local search operators using the intelligent selection mechanism yields very promising result when tested on the well-known VRP data sets. Besides, the incorporation of VNS structure within LNS is also novel.

In addition, the data structure and the neighbourhood reduction schemes which we introduce in the AVNS enable competitive computational time for the algorithm while retaining the solution quality. They are particularly efficient in tackling the very large scale test problems. Simple rules and structures are used when defining both of these schemes, making them easily adaptable for other VRP variants. Moreover, we also present a new local search operator and a new removal strategy for the LNS which have positive impact on the overall results.

The proposed AVNS could be extended to solve other variants of the VRP as well as other combinatorial optimisation problems such as the scheduling and timetabling problems as discussed in the introduction. Furthermore, the integration of adaptive aspects in the search which include the neighbourhood selection and the diversification strategy could be worthwhile exploring further to guide the search even more effectively.

Acknowledgements

We would like to thank the editor and both reviewers for their useful comments and suggestions that improved the presentation as well as the content of the paper. The first author would also like to thank the Ministry of Higher Education Malaysia for the PhD scholarship.

References

Akpinar, S. (2016). Hybrid large neighbourhood search algorithm for capacitated vehicle routing problem. *Expert Systems With Applications, 61*, 28–38. doi:10.1016/j.eswa.2016.05.023.

Azi, N., Gendreau, M., & Potvin, J.-Y. (2014). An adaptive large neighborhood search for a vehicle routing problem with multiple routes. *Computers & Operations Research, 41*, 167–173. doi:10.1016/j.cor.2013.08.016.

Chen, P., Huang, H.-K., & Dong, X.-Y. (2010). Iterated variable neighborhood descent algorithm for the capacitated vehicle routing problem. *Expert Systems with Applications, 37*(2), 1620–1627. doi:10.1016/j.eswa.2009.06.047.

Christofides, N., & Eilon, S. (1969). An algorithm for the vehicle dispatching problems. *Operational Research Quarterly, 20*(3), 309–318.

Clarke, G., & Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research, 12*(4), 568–581. doi:10.1287/opre.12.4.568.

CLHO (2016). Data sets - Routing Data Sets - VRP, Centre for Logistics and Heuristic Optimisation, Kent Business School, University of Kent. http://www.kent.ac.uk/kbs/research/research-centres/clho/datasets.html.

Dantzig, G. B., & Ramser, J. H. (1959). The truck dispatching problem. *Management Science, 6*(1), 80–91. doi:10.1287/mnsc.6.1.80.

Fleszar, K., Osman, I. H., & Hindi, K. S. (2009). A variable neighbourhood search algorithm for the open vehicle routing problem. *European Journal of Operational Research, 195*(3), 803–809. doi:10.1016/j.ejor.2007.06.064.

Golden, B., Wasil, E., Kelly, J., & Chao, I. (1998). The impact of metaheuristics on solving the vehicle routing problem: Algorithms, problem sets, and Computational results. In T. G. Crainic, & G. Laporte (Eds.), *Fleet management and logistics* (pp. 33–56). Boston, MA: Springer US. doi:10.1007/978-1-4615-5755-5.

Hansen, P., & Mladenović, N. (2001). Variable neighborhood search: Principles and applications. *European Journal of Operational Research, 130*(3), 449–467. doi:10.1016/S0377-2217(00)00100-4.

Hansen, P., Mladenović, N., & Moreno Pérez, J. A. (2010). Variable neighbourhood search: Methods and applications. *Annals of Operations Research, 175*(1), 367–407. doi:10.1007/s10479-009-0657-6.

Hassannayebi, E., & Hessameddin, S. (2016). Variable and adaptive neighbourhood search algorithms for rail rapid transit timetabling problem. *Computers & Operations Research*. doi:10.1016/j.cor.2015.12.011.

Imran, A., Salhi, S., & Wassan, N. A. (2009). A variable neighborhood-based heuristic for the heterogeneous fleet vehicle routing problem. *European Journal of Operational Research, 197*(2), 509–518. doi:10.1016/j.ejor.2008.07.022.

Jun, Y., & Kim, B.-I. (2012). New best solutions to VRPSPD benchmark problems by a perturbation based algorithm. *Expert Systems with Applications, 39*(5), 5641–5648. doi:10.1016/j.eswa.2011.11.053.

Kritzinger, S., Doerner, K. F., Tricoire, F., & Hartl, R. F. (2015). Adaptive search techniques for problems in vehicle routing, part I: A survey. *Yugoslav Journal of Operations Research ISSN: 0354-0243 EISSN: 2334—6043, 25*(1), 3–31.

Kytöjoki, J., Nuortio, T., Bräysy, O., & Gendreau, M. (2007). An efficient variable neighborhood search heuristic for very large scale vehicle routing problems. *Computers & Operations Research, 34*(9), 2743–2757. doi:10.1016/j.cor.2005.10.010.

Li, F., Golden, B., & Wasil, E. (2005). Very large-scale vehicle routing: New test problems, algorithms, and results. *Computers & Operations Research, 32*(5), 1165–1179. doi:10.1016/j.cor.2003.10.002.

Li, J., Pardalos, P. M., Sun, H., Pei, J., & Zhang, Y. (2015). Iterated local search embedded adaptive neighborhood selection approach for the multi-depot vehicle routing problem with simultaneous deliveries and pickups. *Expert Systems with Applications, 42*(7), 3551–3561. doi:10.1016/j.eswa.2014.12.004.

Lin, S. (1965). Computer solutions of the traveling salesman problem. *Bell System Technical Journal, 44*(10), 2245–2269. doi:10.1002/j.1538-7305.1965.tb04146.x.

Mester, D., & Bräysy, O. (2005). Active guided evolution strategies for large-scale vehicle routing problems with time windows. *Computers & Operations Research, 32*(6), 1593–1614. doi:10.1016/j.cor.2003.11.017.

Mester, D., & Bräysy, O. (2007). Active-guided evolution strategies for large-scale capacitated vehicle routing problems. *Computers & Operations Research, 34*(10), 2964–2975. doi:10.1016/j.cor.2005.11.006.

Mingozzi, A., Roberti, R., & Toth, P. (2013). An exact algorithm for the multitrip vehicle routing problem. *INFORMS Journal on Computing, 25*(2), 193–207.

Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research, 24*(11), 1097–1100. doi:10.1016/S0305-0548(97)00031-2.

Nagata, Y. (2007). Edge assembly crossover for the capacitated vehicle routing problem. *Evolutionary Computational in Combinatorial Optimization, LNCS, 4446*, 142–153. doi:10.1007/978-3-540-71615-0.

Osman, I., & Salhi, S. (1996). Local search strategies for the vehicle fleet mix problem. In V. Rayward-Smith, I. Osman, C. Reeves, & G. Smith (Eds.), *Modern heuristic search methods* (pp. 131–154). Wiley.

Pisinger, D., & Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research, 34*(8), 2403–2435. doi:10.1016/j.cor.2005.09.012.

Polacek, M., Hartl, R. F., Doerner, K., & Reimann, M. (2004). A variable Neighborhood search for the multi depot vehicle routing problem with time windows. *Journal of Heuristics, 10*(6), 613–627. doi:10.1007/s10732-005-5432-5.

Polat, O., Kalayci, C. B., Kulak, O., & Günther, H.-O. (2015). A Perturbation based variable neighborhood search heuristic for solving the vehicle routing problem with simultaneous pickup and Delivery with time limit. *European Journal of Operational Research, 242*(2), 369–382. doi:10.1016/j.ejor.2014.10.010.

Potvin, J.-Y., & Rousseau, J.-M. (1995). An exchange heuristic for routeing problems with time windows. *Journal of the Operational Research Society, 46*(12), 1433–1446. doi:10.1057/jors.1995.204.

Prins, C. (2009). A GRASP × Evolutionary local search hybrid for the vehicle routing problem. In F. B. Pereira, & J. Tavares (Eds.), *Bio-inspired algorithms for the vehicle routing problem* (pp. 35–53). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-540-85152-3.

Raidl, G. R. (2006). A unified view on hybrid metaheuristics. In F. Almeida, M. Aguilera, C. Blum, J. Moreno-Vega, M. Pérez, A. Roli, & M. Sampels (Eds.), *Lecture notes in computer science, vol. 4030.* (pp. 1–12). Berlin, Heidelberg: Springer. doi:10.1007/11890584_1.

Ropke, S., & Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and Delivery problem with time windows. *Transportation Science, 40*(4), 455–472. doi:10.1287/trsc.1050.0135.

Salhi, S. (2006). Heuristic search in action: The science of tomorrow. In S. Salhi (Ed.), *Or 48 keynote papers* (pp. 39–58). Operational Research Society.

Salhi, S., & Rand, G. K. (1987). Improvements to vehicle routeing heuristics. *Journal of the Operational Research Society, 38*(3), 293–295. doi:10.1057/jors.1987.47.

Salhi, S., & Sari, M. (1997). A multi-level composite heuristic for the multi-depot vehicle fleet mix problem. *European Journal of Operational Research, 103*, 95–112.

Samà, M., Ariano, A. D., Corman, F., & Pacciarelli, D. (2016). A variable neighbourhood search for fast train scheduling and routing during disturbed railway traffic situations. *Computers & Operations Research*. doi:10.1016/j.cor.2016.02.008.

Stenger, A., Vigo, D., Enz, S., & Schwind, M. (2013). An adaptive variable neighborhood search algorithm for a vehicle routing problem arising in small package shipping. *Transportation Science, 47*(1), 64–80.

Tarantilis, C. (2005). Solving the vehicle routing problem with adaptive memory programming methodology. *Computers & Operations Research, 32*(9), 2309–2327. doi:10.1016/j.cor.2004.03.005.

Tarantilis, C., & Kiranoudis, C. (2002). Boneroute: An adaptive memory-Based method for effective fleet management. *Annals of Operations Research, 115*(1–4), 227–241. doi:10.1023/A:1021157406318.

Todosijević, R., Hanafi, S., Urošević, D., Jarboui, B., & Gendron, B. (2016). A general variable neighborhood search for the swap-body vehicle routing problem. *Computers & Operations Research*. doi:10.1016/j.cor.2016.01.016.

Toth, P., & Vigo, D. (2003). The granular tabu search and Its application to the vehicle-Routing problem. *INFORMS Journal on Computing, 15*, 333–346.

Xiao, Y., Zhao, Q., Kaku, I., & Mladenovic, N. (2014). Variable neighbourhood simulated annealing algorithm for capacitated vehicle routing problems. *Engineering Optimization, 46*(4), 562–579. doi:10.1080/0305215X.2013.791813.

Zachariadis, E. E., & Kiranoudis, C. T. (2010). A strategy for reducing the computational complexity of local search-based methods for the vehicle routing problem. *Computers & Operations Research, 37*(12), 2089–2105. doi:10.1016/j.cor.2010.02.009.