Using a Permutation Representation Genetic Algorithm to Implement a Complex Pick Up/Drop Off Mathematical Model

Sundas Choudry[a], Martin C. Serpell[a], Paul Burnham[b]

[a] *Computer Science Research Centre, University of the West of England, Bristol, BS16 1QY, United Kingdom*
[b] *Kelly Bros Solar Signs Ltd., 13-15 Station Rd, Yate, Bristol, BS37 5HT, United Kingdom*

Abstract

The Variable Message Sign Problem (VMSP) aims to optimise the delivery, collection and maintenance of Variable Message Signs. Firstly, the problem was formulated as a rigorous mathematical model. It was then reformulated using a permutation representation removing the need for some hard constraints and the formation of illegal sub-tours within the solution. The reformulated model was shown to be solvable using different artificial intelligence search techniques. The problem of parameter selection was then solved using self-adaption, which produced superior solutions with $> 99.9\%$ confidence by avoiding numerous local optima on the fitness landscape.

Key words: Variable Message Sign Problem, Genetic Algorithm, Self-Adaption, Mathematical Model, Vehicle Routing, Pick-Up/Drop-Off

1. Introduction

The Variable Message Sign Problem (VMSP) is a variant of the Vehicle Routing Problem (VRP) where Variable Message Signs (VMS) are hired out to customers for a given time window. VMS are solar and battery powered roadside signs; the message displayed on VMS is configured remotely. A typical scenario of their use would be by the roadside instructing diversions. The service provided includes the delivery and collection of VMS to and from locations as well as maintenance while at a customer location. Therefore, transportation of goods within a time window is a key aspect of the service

provided. Cost involving transportation has a direct and significant impact on profit margins. The objective of the VMSP is to keep the costs to a minimum while meeting customer obligations. This is mainly achieved by optimising vehicle routes and scheduling of crew and resources.

The rest of this paper proceeds as follows. Section 2 examines various vehicle routing problems and solutions. Section 3 develops a rigorous mathematical model for the VMSP. Section 4 demonstrates how this mathematical model was converted into a permutation representation. Section 5 compares the use of Local Search and Global Search with this new representation. Section 6 looks at how self-adaption was added to the new representation. Section 7 describes how the performance of the self-adaption was evaluated. Section 8 examines the results, section 9 draws conclusions and looks at future research.

2. Background

The problem of delivering, collecting and maintaining Variable Message Signs (VMS) is similar to many Vehicle Routing Problems [1]. Like the Vehicle Routing Problem (VRP), the Variable Message Sign Problem (VMSP) is a route optimisation problem faced by organisations where a fleet of vehicles is required to serve a set of customers by delivering goods and services. The objective is to find a route or set of routes which minimises the total traveling distance and/or time while satisfying all applicable constraints. It was reported that the use of a combinatorial optimisation such as VRP would often result in savings ranging from 5%-20% in transportation costs [2]. As the number of locations to be visited increases, the number of feasible solutions increases exponentially [3]. VRP is a NP-Hard type problem [4] therefore an algorithm does not exist that can find the optimal solution in polynomial time in every instance [5].

The delivery, collection and maintenance of Variable Message Signs (VMS) has similarities to the following variants of the VRP.

- Capacitated Vehicle Routing Problem (CVRP)
- Vehicle Routing Problem with Time Windows (VRPTW)
- Vehicle Routing Problem with Pickup and Delivery (VRPPD)
- Vehicle Routing Problem with Product Backhauls (VRPPB)

- Vehicle Routing Problem with Heterogeneous Fleet (VRPHF)

The Capacitated Vehicle Routing Problem (CVRP) has one depot and a fleet of identical vehicles where the capacity of each vehicle must not be exceeded. CVRP produces optimal delivery routes where each vehicle only travels one route and each route must start and end at the depot. CVRP is also known as the classical VRP and is the most common variant as vehicles are rarely assumed incapacitated [6]. The Vehicle Routing Problem with Time Windows (VRPTW) is another variant of VRP in which, optimal routes must be constructed while ensuring that each task is carried out within a given time interval at minimum cost without violating the capacity and total trip time constraints of each vehicle [7] [8]. Time Windows are defined as hard constraints when it is not allowed to deliver outside of the given time [9] [10] and when it is allowed outside of the given time against a penalty, they are considered to be soft constraints [11] [12]. In the Vehicle Routing Problem with Pickup and Delivery (VRPPD), either objects or people need to be picked up from a certain location and dropped off at another. The main objective is to transport from an origin to a destination [13]. The pick up and drop off must be done by the same vehicle, therefore both locations must be included in the route [11]. The Vehicle Routing Problem with Product Backhauls (VRPPB) is when a vehicle does deliveries as well as pickups in one route [14]. The routes must consider collecting items from some customers and delivering items to another in the same route, either in a sequential or discontinuous manner. In the Vehicle Routing Problem with Heterogeneous Fleet (VRPHF), the vehicles do not share the same characteristics; they may have different capacities and fixed costs depending on vehicle type and configuration [15]. Exact Methods produce optimal solutions [16] when applied to a small-sized VRP. However, larger VRPs are best solved using heuristics because exact algorithms cannot be guaranteed to find the optimal tours within reasonable computing time when the number of locations to be visited is large. Heuristics allow acceptable performance at acceptable costs for a wide range of VRP variants [16].

A VRP with a set of vehicles and a set of locations, where each vehicle visits at least one location, can be formulated and solved as a Traveling Salesman Problem. The Travelling Salesman Problem (TSP) is a combinatorial optimisation problem which was first formulated as a mathematical problem in 1930, [17]. Since its introduction, TSP has been intensively studied in operations research and theoretical computer science. TSP in its pure form

appears in the planning and transportation problems. However, it often appears as a sub-problem in more complex combinatorial problems with added constraints, for example vehicle routing, airport flight scheduling, time and job scheduling machines, DNA sequencing and manufacturing of microchips, [18]. The VMSP contains an asymmetric TSP sub-problem where there is a set of jobs at different locations, each job must only be carried out once, an employee must travel from the depot to a set of job locations and return to the depot, and the route through all the jobs must be optimised to ensure optimal use of company resources such as vehicle mileage and fuel. A TSP can be solved using either an exact algorithm or an approximate algorithm. An exact algorithm would fully search the solution space for the global optimal solution while an approximate algorithm provides a solution as close as possible to the optimal value in a reasonable amount of time but it does not ensure the optimal solution. With an exact algorithm, it is difficult to solve a large-scale problem because of its exponential time complexity. Exact algorithms used to solve the TSP include branch and bound [19] [20], Branch and cut, and brute force. Approximate algorithms used to solve the TSP include Simulated Annealing algorithm (SA) [21] [22], Tabu Search (TS) [21], Ant Colony Optimization (ACO) [23], and the Genetic algorithm (GA) [24] [25]. The hybridisation of algorithms has become popular in recent years as it combines useful features of multiple algorithms, while overcoming the limitations of individual algorithms. It was shown that a combination of genetic algorithm and 2-opt method produced a more efficient algorithm for solving TSP compared to using a genetic algorithm alone [26]. The hybridisation of a Genetic Algorithm with Nearest Neighbour heuristic produced an algorithm that was more efficient in terms of computational time and space, and had better convergence than a pure Genetic Algorithm or Nearest Neighbour heuristic [27]. Other examples of hybrid algorithms are; the glowworm swarm optimization and the complete 2-opt algorithm [28] and a stochastic approach combined with a variable neighborhood search algorithm [29]. New algorithms such as the water cycle algorithm [30] or improvements on older algorithms such as variations of the basic GA that produce larger numbers of offspring [31] have also been shown to be effective.

The Genetic Algorithm (GA) has been adapted for use in solving permutation problems such as the Traveling Salesman Problem (TSP) [32] [33]. The GA was introduced in 1975 [34] [35] and mimicked biological evolution as described by Charles Darwin in his 1880 book 'On the origin of species by means of natural selection' [36]. It consists of a population of chromosomes

that goes through various stages, the stages are; parent selection, crossover providing an offspring, offspring mutation that adds variation in the population, and finally replacement. Replacement happens when the fitness of the offspring is superior to the fitness of a member of the existing population. The stages are repeated iteratively leading to the fitness of the population improving over time.

A drawback of using a genetic algorithm is that often, time must be spent fine-tuning its parameters [37] [38]. Such parameters may control population size, crossover type, crossover probability, mutation operator and mutation probability. To ease this problem self-adaption of some or all of these parameters has been developed. In particular, mutation operator and mutation probability have been self-adapted in the continuous [39] [40], binary [41] [42] [43] [44] and permutation domains [45]. Self-adaption of the mutation operator in permutation GAs has been shown to be advantageous. This is because each permutation mutation operator sees the fitness landscape in a very different way [46] [47] [48] [49]; what is a local optimum to one mutation operator is not to another and hence switching between mutation operators help prevent the GA getting stuck in a local optimum. It was found that self-adaptive permutation GA's produced better or comparable results compared to the best tuned non-adaptive GA's and this was irrespective of other GA parameter settings [45]. They demonstrated that the choice of mutation operator changed over time as each in turn became trapped in a local optimum within its associated fitness landscape. The next successful mutation operator to spread through the population was one that had a fitness landscape with no local Optima at that location. Since then self-adaption, in permutation GAs, has been applied to many problem types [50] [51] [52] [53] [54] [55].

3. Mathematical model for the Variable Message Sign Problem

A rigorous mathematical model for the VMSP was developed. The objective function of the VMSP had to ensure that the most urgent tasks were carried out whilst also trying to minimise the travel time between jobs and hence fuel costs.

$$Maximise \quad \sum_{b \in B} x_b(\alpha \sum_{j \in J} c_{bj} p_j - \beta t_b) \tag{1}$$

Where J is the set of jobs that need to be done, B is the set of batches (jobs that are to be done together), x_b is weighting allocated to Batch b, c_{bj}

indicates that Batch b contains Job j, p_j is the priority allocated to Job j, and t_b is the time to complete the jobs in Batch b. The batch weighting, x_b, allows for the most current work to be weighted higher than the work to be done in the following days. This is important as the priorities of the work to be carried out can change on a daily basis. The VMSP objective function has two parts; one to ensure that the most urgent tasks are carried out ($\sum_{j \in J} c_{bj} p_j$) and one to minimise the travel time (t_b). The weighting of these two components are controlled by two constants, α and β.

The time component, t_b, is made up from the time it takes to carry out each task or job and the time it takes to travel between the jobs.

$$t_b = \sum_{i=1,\ldots,|k_b|} (D_{j_{i-1}j_i} + T_{j_i}) + D_{|k_b|0}, \quad \forall b \in B \tag{2}$$

Where k_b is the sequence of Jobs in Batch b, D_{ij} is the time to travel between Job i and Job j, D_{0i} is the time to travel between the Depot and Job i, D_{i0} is the time to travel between Job i and the Depot, and T_i is the time taken to complete Job i.

The objective function is subject to a set of constraints. The jobs that can be carried out are put into batches. Each batch of jobs is allocated to a pair of workers and also a vehicle which will carry the signs and the batteries to be replaced. Each job may only be in one batch so each job which might be moving a sign or swapping batteries can only be carried out by one pair of workers using one vehicle.

$$\sum_{b \in B} c_{bj} = 1, \quad \forall j \in J \tag{3}$$

Each batch of work may only contain no more than a given number of jobs in this case we set the number to N.

$$\sum_{j \in J} c_{bj} \leq N, \quad \forall b \in B \tag{4}$$

Each batch is associated with a single vehicle.

$$\sum_{v \in V} h_{bv} = 1, \quad \forall b \in B \tag{5}$$

Where V is the set of Vehicles that are available, and h_{bv} indicates that Batch b uses Vehicle v.

Each vehicle can only be used once in a given day. If L batches of work can be carried out each day we need to ensure that no vehicle is used more than once on the first day. We only need to check the first day as a new schedule is generated daily.

$$\sum_{b=1..L} h_{bv} \leq 1, \quad \forall v \in V \tag{6}$$

There is a load weight restriction each vehicle is allowed to carry. Each batch may involve the picking up and dropping off of multiple signs. Therefore, two constraints were used, the first to ensure a positive weight limit is not broken, when signs are being collected.

$$\sum_{j \in J} c_{bj} (\sum_{s \in S^s} g_{js} W^s + \sum_{s \in S^m} g_{js} W^m + \sum_{s \in S^l} g_{js} W^l + \sum_{s \in S^r} g_{js} W^r + r_j W^b) \leq \sum_{v \in V} h_{bv} W^v_v, \quad \forall b \in B \tag{7}$$

The second constraint ensuring that the weight limits are not broken when signs are being dropped off.

$$\sum_{j \in J} c_{bj} (\sum_{s \in S^s} g_{js} W^s + \sum_{s \in S^m} g_{js} W^m + \sum_{s \in S^l} g_{js} W^l + \sum_{s \in S^r} g_{js} W^r + r_j W^b) \geq -\sum_{v \in V} h_{bv} W^v_v, \quad \forall b \in B \tag{8}$$

For both constraints S is the set of all signs. However, there are four types of sign that can be allocated to each job; S^s is the set of all small sized signs, S^m is the set of all medium sized signs, S^l is the set of all large sized signs, and S^r is the set of all radar speed signs.

$$S^s \cup S^m \cup S^l \cup S^r = S \tag{9}$$

$$S^s \cap S^m = \emptyset, \quad S^s \cap S^l = \emptyset, \quad S^s \cap S^r = \emptyset, \quad S^m \cap S^l = \emptyset, \quad S^m \cap S^r = \emptyset, \quad S^l \cap S^r = \emptyset \tag{10}$$

Each size of sign has a weight; W^s is the weight of a small sized sign, W^m is the weight of a medium sized sign , W^l is the weight of a large sized sign, and W^r is the weight of a radar speed sign. The variable g_{js} indicates that Sign s is allocated to Job j. $g_{js} \in \{-1, 0, +1\}$ where -1 indicates that a sign is being dropped off and $+1$ that a sign is being picked up. The variable r_j indicates the number of batteries allocated to Job j, and W^b is the weight of a single battery. W_v^v is the weight capacity of Vehicle v.

The combination of constraints allows for signs both being picked up and dropped off in the same batch of work. Similarly, we need constraints to deal with the volume of materials that can be carried by the vehicles.

$$\sum_{j \in J} c_{bj} (\sum_{s \in S^s} g_{js} M^s + \sum_{s \in S^m} g_{js} M^m + \sum_{s \in S^l} g_{js} M^l + \sum_{s \in S^r} g_{js} M^r + r_j M^b) \leq \sum_{v \in V} h_{bv} M_v^v, \quad \forall b \in B \tag{11}$$

$$\sum_{j \in J} c_{bj} (\sum_{s \in S^s} g_{js} M^s + \sum_{s \in S^m} g_{js} M^m + \sum_{s \in S^l} g_{js} M^l + \sum_{s \in S^r} g_{js} M^r + r_j M^b) \geq -\sum_{v \in V} h_{bv} M_v^v, \quad \forall b \in B \tag{12}$$

Where M^s is the volume of a small sized sign, M^m is the volume of a medium sized sign, M^l is the volume of a large sized sign, M^r is the volume of a radar speed sign, M^b is the volume of a battery, and M_v^v is the volume capacity of Vehicle v.

Three batches of work are carried out each day, and we need a constraint that ensures that the number of batteries that are used on that day is less than or equal to the number of batteries in reserve. All spare batteries are left on charge overnight at a secure unit.

$$\sum_{b=1,2,3} \sum_{j \in J} c_{bj} r_j \leq R \tag{13}$$

Where R is the number of spare batteries in stock.

The jobs that are allocated to each batch are ordered, that is to say the

permutation in which the jobs are done is very important as this affects the time taken to do the jobs and the fuel usage. Therefore, the ordered set, k_b, was introduced to hold the permutation of the jobs in each batch. With the introduction of this variable it is important again to ensure that no job is allocated to more than one batch.

$$\sum_{b \in B} c_{bj_i} = 1, \quad i = 1, ..., |k_b|, \forall b \in B \tag{14}$$

Further checks must be carried out on the permutation to ensure that both the weight of signs and batteries placed on a vehicle and the volume of the signs and the batteries placed on the vehicle do not break its load constraint.

$$\begin{aligned} w_{bj_i} &= w_{bj_{i-1}} + \sum_{s \in S^s} g_{j_i s} W^s + \sum_{s \in S^m} g_{j_i s} W^m \\ &+ \sum_{s \in S^l} g_{j_i s} W^l + \sum_{s \in S^r} g_{j_i s} W^r, \quad i = 1, ..., |k_b|, \forall b \in B \end{aligned} \tag{15}$$

$$w_{bj_i} \leq \sum_{v \in V} h_{bv} W^v_v, \quad \forall i \in k_b, \forall b \in B \tag{16}$$

Where w_{bi} is the weight of the items on the vehicle after Job i in Batch b, and w_{b0} is the weight of the items on the vehicle in Batch b when it leaves the Depot.

$$\begin{aligned} v_{bj_i} &= v_{bj_{i-1}} + \sum_{s \in S^s} g_{j_i s} M^s + \sum_{s \in S^m} g_{j_i s} M^m \\ &+ \sum_{s \in S^l} g_{j_i s} M^l + \sum_{s \in S^r} g_{j_i s} M^r, \quad i = 1, ..., |k_b|, \forall b \in B \end{aligned} \tag{17}$$

$$v_{bj_i} \leq \sum_{v \in V} h_{bv} M^v_v, \quad \forall i \in k_b, \forall b \in B \tag{18}$$

Where v_{bi} is the volume of the items on the vehicle after Job i in Batch b, and v_{b0} is the volume of the items on the vehicle in Batch b when it leaves the Depot.

Each batch of work should not take more than 8 hours (480 minutes) to complete; however in exceptional circumstances they are allowed up to 10 hours (600 minutes). To cater for this, two soft constraints are added.

$$t_b \leq 480 \tag{19}$$

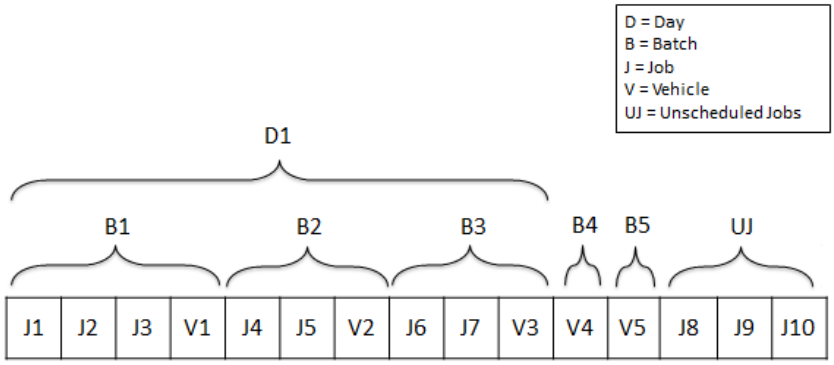

Figure 1: Permutation Encoding

$$t_b \leq 600 \tag{20}$$

Figure 7 shows the time that each batch is expected to take; it is these calculated times that are used in constraint equations 19 and 20.

4. Reformulating the VMSP mathematical model using a permutation representation

The VMSP mathematical model attempts to find the shortest path between jobs in a given batch. Mathematical models and mathematical solvers have been shown to be very poor at solving this type of permutation problem. They can provide invalid solutions which contain a set of sub-tours which are not connected into a single tour.

A reformulation of the mathematical model into one that uses a permutation representation was developed; such a representation does not have the problem of creating sub-tours as by its very nature it can only create a single tour. The permutation representation contained the jobs that need to be carried out and the vehicles available to carry out those jobs. The fitness of a given permutation was calculated using the mathematical model's objective function and constraints after copying the jobs and vehicles within the permutation representation into the mathematical model variables c_{bj}, h_{bv}, g_{js}, r_j, and k_b.

The genotype is a permutation of the pending jobs and the available vehicles, as shown in the example in Fig. 1. Within this representation an individual solution is a string of jobs and vehicles. A batch is represented

as one or more adjacent job followed by a vehicle. A batch may contain 0 or more jobs and always end with a vehicle. A batch with 0 jobs is called an empty batch. The vehicle belonging to this batch is not scheduled. Jobs which do not belong to a batch are not scheduled; for example, J8, J9 and J10 in Fig. 1. The sequence of jobs in a batch determines the route that the vehicle will follow. For example, vehicle V1, belonging to batch 1, will travel from the depot to J1, J2 and J3, in that order, before returning to the depot. In this example, each batch of jobs is required to be carried out by a pair of employees. If the company has 6 employees, first three non-empty batches define the workload of day 1; second 3 batches define the workload of day 2 and so on. Each vehicle object is present in the representation 5 times. A vehicle can be used once per day and the schedule is created for 5 days. If the jobs in Fig. 1 are as follows.

- J1 = drop-off
- J2 = drop-off
- J3 = pickup
- J4 = battery change
- J5 = drop-off
- J6 = pickup
- J7 = drop-off
- J8 = drop-off
- J9 = drop-off
- J10 = drop-off

Then the mathematical model variables c_{bj}, h_{bv}, g_{js}, r_j, and k_b are as shown in Figures 2, 3, 4, 5, and 6. The time taken to complete each batch of jobs is calculated and stored in t_b, as shown in Fig. 7. Implementing the time constraints shown in equations 19 and 20 required consideration. As soft constraints, they can be broken with the penalty increasing with the length of time that they are broken by. This is because with a single value penalty there is no reason for any search strategy to distinguish between a slight time overrun on the day's work and a massive one.

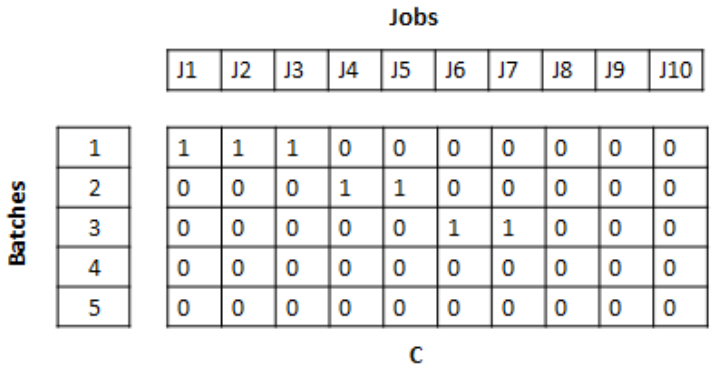

Figure 2: Mathematical Model Variable c_{bj}

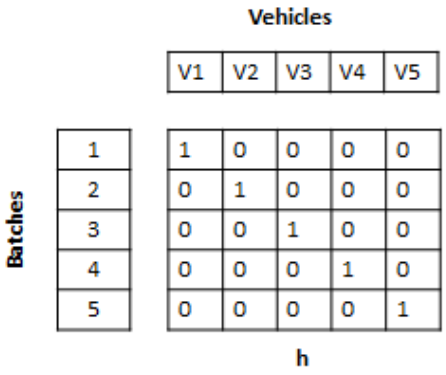

Figure 3: Mathematical Model Variable h_{bv}

Signs

Jobs	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13
J1	-1	0	0	0	0	0	0	0	0	0	0	0	0
J2	0	-1	0	0	0	0	0	0	0	0	0	0	0
J3	0	0	0	+1	0	0	0	0	0	0	0	0	0
J4	0	0	0	0	0	0	0	0	0	0	0	0	0
J5	0	0	0	0	-1	0	0	0	0	0	0	0	0
J6	0	0	+1	0	0	0	0	0	0	0	0	0	0
J7	0	0	0	0	0	-1	0	0	0	0	0	0	0
J8	0	0	0	0	0	0	-1	0	0	0	0	0	0
J9	0	0	0	0	0	0	0	0	0	-1	0	0	0
J10	0	0	0	0	0	0	0	0	0	0	0	-1	0

g

Figure 4: Mathematical Model Variable g_{js}

Jobs	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10
r	0	0	4	6	0	0	0	0	0	0

Figure 5: Mathematical Model Variable r_j

Batches										
1	J1	J2	J3	null	null	null	null	null	null	null
2	J4	J5	null	null	null	null	null	null	null	null
3	J6	J7	null	null	null	null	null	null	null	null
4	null	null	null	null	null	null	null	null	null	null
5	null	null	null	null	null	null	null	null	null	null

k

Figure 6: Mathematical Model Variable k_b

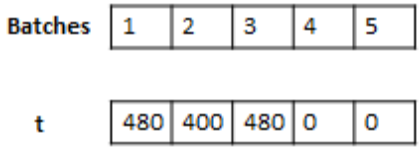

Figure 7: Mathematical Model Variable t_b

Using this permutation representation leads to the constraints shown in equations 3, 5 and 14 becoming redundant. This is because each job only occurs once in the permutation and therefore can only ever occur once in the Mathematical Models variables c and h.

The fitness function, for the search strategy, is a combination of the objective function value (obj fn) and the weighting (W_i) of the broken constraints (BC_i) as shown in equation 21. The values of α and β in the objective function described by equation 1 balance the importance of the priority of the tasks and the time taken to carry them out. The value of α was set to 20 and β to 1 to ensure that high priority jobs took precedence over shorter routes. The strategy for dealing with broken constraints was chosen for both its simplicity and its ability to reflect the number and importance of any constraint violations. The weighting (W_i) was set punitively high, so that any broken constraints would give a fitness function with a negative value. As the objective is to maximise utility, the higher the fitness value the better the solution.

$$fitness \quad = \quad obj\ fn - \sum_{i=constraint\ 3..20} W_i BC_i \tag{21}$$

One modification must be made when implementing this fitness function. The objective function contains a weighting, x_b for each batch, which ensures that the jobs for the current day (and those to shortly follow) are given a higher weighting. This weighting must also be applied to the broken constraints BC_i.

The permutation representation has been designed to be compatible with multiple search strategies.

5. Local versus global search

A comparison of a local and global search technique using the newly designed permutation representation was carried out. The local search and GA search strategies were kept as similar as possible. Both used the same mutation operator, both were allowed to run for the same numbers of calls to the fitness function and both had the same termination criteria. Local search was represented by a Greedy Local Search algorithm that used the swap operator. Global search was represented by a Genetic Algorithm (GA) which used swap for its mutation operator; the GA parameters are given in Table 1. Each search strategy was allowed to run for up to 1,000,000 calls to the fitness

Table 1: GA parameter settings for local versus global search

Parent population size	40
Parent population initialisation	Completely random
Mating population size	2
Offspring population size	1
Selection	Binary tournament between two random parents
Crossover probability	0.7
Crossover operator	Partially Mapped Crossover
Mutation probability	Fixed: 0.4
Mutation operator	Fixed: SWAP
Replacement	Binary tournament between the offspring and a random parent

function and terminate if 100,000 showed no improvement. Each algorithm was run ten times on five different datasets. A Mann-Whitney Test showed with $> 99.9\%$ confidence that the global search performed best. Detailed examination of the search data showed that the local search became stuck in a local optimum and terminated early. The global search, however managed to avoid becoming stuck and went on to find better solutions; terminating much later.

The important finding however is that this permutation representation is compatible with multiple search strategies. Many operators have been developed for permutation representations and selecting between them can be problematic.

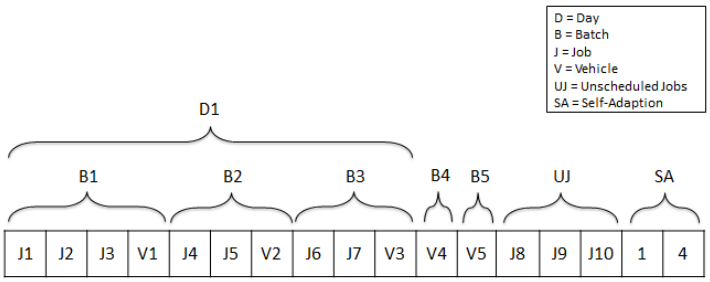

Figure 8: GA Encoding Self-Adaption

6. Using a permutation genetic algorithm with self-adaption of the mutation operator and probability

Previous research [45] has shown that self-adapting the mutation operator and probability has proved beneficial for solving permutation problems such as the Traveling Salesman Problem. Changes to the pool size, selection algorithm, crossover operator and replacement strategies were shown not to affect performance significantly. There are a large variety of mutation operators that can be used in a permutation representation genetic algorithm as well as a variety of mutation rates. To avoid the need to choose a particular mutation operator or probability, self-adaption was added into each chromosome with the addition of two new genes at the end of the chromosome. The first gene represented the mutation operator and was an integer between 0 and 4 where 0 = Swap, 1 = Insertion, 2 = Inversion, 3 = Scramble, and 4 = Translocation. The second gene represented the mutation probability and was an integer between 0 and 7 where $0 = 0.0, 1 = 0.05, 2 = 0.1, 3 = 0.15, 4 = 0.2, 5 = 0.3, 6 = 0.4$, and $7 = 0.5$. At population initialisation, these integers were generated randomly in their given ranges and a 0.1 mutation probability was applied to each gene during an evolution. A set of experiments was carried out to evaluate the performance of self-adaption.

7. Evaluating the performance of self-adaption

A steady-state GA was used in all experiments, the GA's parameters are given in Table 2. In these experiments the mutation probability is the probability of a single mutation operation taking place once on the chromosome

as a whole. Ten realistic synthetic datasets describing a busy period were generated. To enable statistical analysis, each time one of the settings of the GA was applied to a dataset and run twenty times with a different seed and the results were recorded. When self-adaption of the mutation operator or mutation probability is in play, then the offspring has an equal chance of inheriting the mutation operator or mutation probability from each of its parents. The GAs were allowed to run for up to $100,000$ calls to the fitness function or until no improvement was detected for $10,000$ iterations. It is not

Table 2: GA parameter settings for the self-adaption experiments

Parent population size	40
Parent population initialisation	Completely random
Mating population size	2
Offspring population size	1
Selection	Binary tournament between two random parents
Crossover probability	0.7
Crossover operator	Partially Mapped Crossover
Mutation probability	Fixed: 0.05, 0.1, 0.15, 0.2, 0.3, 0.4 and 0.5 Self-adaptive: Randomly switch between the above with probability 0.1
Mutation operator	Fixed: INSERT, INVERSION, SCRAMBLE, SWAP or TRANSLOCATION [32] Self-adaptive: Randomly switch between the above with probability 0.1
Replacement	Binary tournament between the offspring and a random parent

uncommon to initialise the initial parent population with some best guesses at a reasonable solution. In this case it would have been easy to sort the jobs by both latitude and longitude and insert these possible solutions into the initial population. This however would not have allowed such a rigorous examination of the mutation operators, as initialising the parent population completely randomly would do. The GAs were only allowed to run up to

100, 000 calls to the fitness function, this value was chosen simply for practicality. Choosing a larger value would have meant much larger execution time without showing us much more about the behaviour of the algorithm. In reality, when this software is put into real use the number of calls to the fitness function will be increased.

8. Results

In total the experiment was run 9,600 times; twenty differently seeded runs for each combination of mutation operator, mutation probability and dataset. Each run of the experiment was configured as described in Table 2 and allowed to execute for up to 100,000 calls to the fitness function. The results are given in Tables 3 and 4. Solutions with broken constraints were heavily penalised by the addition of a large negative penalty. The penalty was so large that any such solution would have a negative fitness value and this allowed easy identification of these solutions. Any solution with a broken constraint is likely to be infeasible; for example the solution might instruct a driver to place four signs on a lorry that can only physically carry three. An analysis of the experimental results was carried out to see if any of the GA configurations produced such infeasible solutions.

Table 3: The number of runs that produced a valid solution (the number of solutions with broken constraints in brackets) for each mutation operator for the ten datasets

Mutation type	Datasets									
	1	2	3	4	5	6	7	8	9	10
INSERT	149 (11)	148 (12)	151 (9)	146 (14)	147 (13)	156 (4)	157 (3)	155 (5)	154 (6)	153 (7)
INVERSION	72 (88)	55 (105)	73 (87)	53 (107)	51 (109)	69 (91)	85 (75)	75 (85)	66 (94)	58 (102)
SCRAMBLE	29 (131)	24 (136)	31 (129)	23 (137)	20 (140)	34 (126)	46 (114)	36 (124)	29 (131)	19 (141)
SWAP	153 (7)	138 (23)	157 (4)	150 (11)	144 (17)	150 (11)	154 (7)	155 (6)	147 (15)	149 (13)
TRANS-LOCATION	58 (102)	52 (108)	70 (90)	62 (98)	35 (125)	68 (92)	75 (85)	71 (89)	68 (92)	52 (108)
Self-adaptive	132 (28)	126 (34)	134 (26)	129 (31)	130 (30)	133 (27)	137 (23)	134 (26)	134 (26)	136 (24)

Table 3 shows the number of feasible and infeasible solutions for each of the mutation operators operating on each of the ten datasets. Each of the mutation operators produced a significant number of infeasible solutions. A simple glance at the table shows that mutation operators insert and swap

produce less infeasible solutions than the other mutation operators. The fact that none of the mutation operators under investigation could guarantee to produce a valid solution was unexpected. Previous research [45] has shown that the number of calls to the fitness function is relatively low and that maybe with more time valid solutions may have been found.

Table 4: The number of runs that produced a valid solution (the number of solutions with broken constraints in brackets) for each mutation probability for the ten datasets

Mutation probability	Datasets									
	1	2	3	4	5	6	7	8	9	10
0.05	33 (87)	20 (101)	36 (85)	29 (92)	23 (98)	35 (86)	41 (80)	36 (85)	30 (92)	33 (89)
0.1	52 (68)	44 (76)	53 (67)	45 (75)	45 (75)	52 (68)	58 (62)	57 (63)	58 (62)	52 (68)
0.15	68 (52)	58 (62)	67 (53)	61 (59)	58 (62)	69 (51)	75 (45)	69 (51)	67 (53)	63 (57)
0.2	75 (45)	67 (53)	78 (42)	75 (45)	65 (55)	77 (43)	80 (40)	85 (35)	78 (42)	75 (45)
0.3	92 (28)	86 (34)	98 (22)	88 (32)	82 (38)	91 (29)	103 (17)	96 (24)	91 (29)	85 (35)
0.4	101 (19)	99 (21)	99 (21)	95 (25)	87 (33)	102 (18)	105 (15)	100 (20)	98 (22)	94 (26)
0.5	103 (17)	102 (18)	107 (13)	103 (17)	102 (18)	106 (14)	107 (13)	106 (14)	105 (15)	99 (21)
Self-adaptive	69 (51)	67 (53)	78 (42)	67 (53)	65 (55)	78 (42)	85 (35)	77 (43)	71 (49)	66 (54)

Table 4 shows the number of feasible and infeasible solutions for each of the mutation probabilities under investigation for each of the datasets. Again Table 4 shows that each mutation probability creates many infeasible solutions. However, this time a striking pattern within the data can be seen. Looking down each column shows that for each of the datasets the number of feasible solutions increases and the number infeasible solutions decreases. So Table 4 indicates that as the mutation probability increases, so does the number of feasible solutions. In fact the best solutions are achieved with the very large mutation probability of 0.5. This is surprising as it tells us that mutation is playing a very significant part in the search for feasible solutions. This implies that there is a landscape pot marked by local Optima and that only by trying more and newer directions will the algorithm be able to move across the fitness landscape.

Table 5 contains the counts of feasible and infeasible solutions by mutation probabilities and mutation operators. Scanning down the columns of Table 5 again shows that the number of feasible solutions increases with the mutation

Table 5: The number of runs that produced a valid solution (the number of solutions with broken constraints in brackets) for each GA configuration for the ten datasets combined; GA configurations where all solutions were valid are in bold

Mutation probability	INSERT	INVERSION	SCRAMBLE	SWAP	TRANS-LOCATION	Self-adaptive
0.05	142 (58)	4 (196)	1 (199)	121 (90)	1 (199)	47 (153)
0.1	190 (10)	13 (187)	3 (197)	181 (19)	15 (185)	114 (86)
0.15	190 (10)	38 (162)	14 (186)	197 (3)	41 (159)	175 (25)
0.2	198 (2)	77 (123)	18 (182)	199 (1)	67 (133)	196 (4)
0.3	**200 (0)**	129 (71)	59 (141)	**200 (0)**	124 (76)	**200 (0)**
0.4	199 (1)	157 (43)	79 (121)	**200 (0)**	145 (55)	**200 (0)**
0.5	199 (1)	176 (24)	95 (105)	**200 (0)**	170 (30)	**200 (0)**
Self-adaptive	198 (2)	63 (137)	22 (178)	199 (1)	48 (152)	193 (7)

probability. With the data displayed in this format table 4 shows that some combinations of mutation operator and mutation probability have produced feasible solutions. In particular, mutation operator insert with mutation probability 0.3, swap with mutation probability 0.3 to 0.5 and self-adaptive with mutation probability 0.3 to 0.5 produced only feasible solutions. This indicates that the mutation operators insert and swap are well suited for solving this particular problem and it is assumed that when self-adaption is used that it will make good use of these two operators. This assumption will be examined later in this paper. Table 5 also clearly shows that the higher mutation probabilities produced the best results. Self-adaption of the mutation probability produced most feasible solutions for mutation operators insert, swap and self-adaptive. At the beginning of each run, when there is a lot of variation in the population, crossover should dominate the search across the fitness landscape. Once the population converges to a few similar solutions, then mutation should become more important to the search. Hence, when self-adaption of the mutation probability is used, the probability typically starts with a low value but increases over time ensuring that variation is maintained within the population. In these experiments, self-adaption of the mutation probability has not guaranteed that only feasible solutions will be found. This may be explained by the fact [45] that when using self-adaption the GA will typically need to run longer. One of the runs will be examined in more detail later in this paper to see if that is the case here.

Table 6 shows the mean fitness by mutation operator and mutation probability. The table shows that the fitness increases with the mutation probability as was seen in Tables 4 and 5. The highest mean fitness was produced

Table 6: Mean fitness for each GA configuration for the ten datasets; GA configurations where all solutions were valid are in bold

Mutation probability	INSERT	INVERSION	SCRAMBLE	SWAP	TRANS-LOCATION	Self-adaptive
0.05	487.04	-2027.64	-2290.24	12.92	-1958.90	-989.03
0.1	2249.58	-1357.80	-1877.32	994.72	-1290.52	154.78
0.15	2668.96	-926.03	-1409.52	1518.14	-752.04	1685.80
0.2	3026.88	-477.66	-1190.07	2130.19	-344.85	2805.89
0.3	**3480.01**	369.37	-697.59	**2681.28**	199.42	**3831.60**
0.4	3696.30	961.40	-405.01	**3274.05**	484.40	**4428.89**
0.5	3921.02	1307.52	-107.11	**3586.01**	925.14	**4773.14**
Self-adaptive	3073.74	-535.96	-1252.59	1958.75	-736.51	2428.72

with a self-adapting mutation operator and the highest mutation probability of 0.5. When only considering the combinations of mutation operator of mutation probability that produced only feasible solutions, a self-adaptive mutation operator performed the best. The Kruskal-Wallis Test indicated with $> 99.9\%$ confidence that there was a significant difference in the performance of the seven combinations of mutation probabilities and mutation operators used in the GA that produced only feasible results. The Mann-Whitney Test indicated, with 99.9% confidence, that the performance of the GA using the combination of mutation operator and mutation probability that gave the highest mean fitness value was significantly better than that with the second highest. Hence, these results indicate that the GA that uses the self-adaptive mutation operator and a mutation probability of 0.5 significantly outperforms other GAs. Table 6 clearly shows the superior performance of the self-adaptive mutation operator. Whereas the other single operators will always see the same fixed fitness landscape the self-adaptive mutation operator can switch between the fitness landscapes seen by the single operators at will, giving it the ability to move past any obstacles found in a single landscape.

Figure 9 indicates which mutation operator was used whilst solving one of the pick-up/drop-off problems. The initial fitness readings have not been plotted as they contained large negative numbers indicating broken constraints. If these values had been plotted, then the remaining values would not be distinguishable. The fitness plot contains many plateaus which indicate that the search is stuck in a local optimum as a result of both crossover and mutation failing to find a better solution. Figure 9 shows us that different mutation operators dominate mutation at different times during the

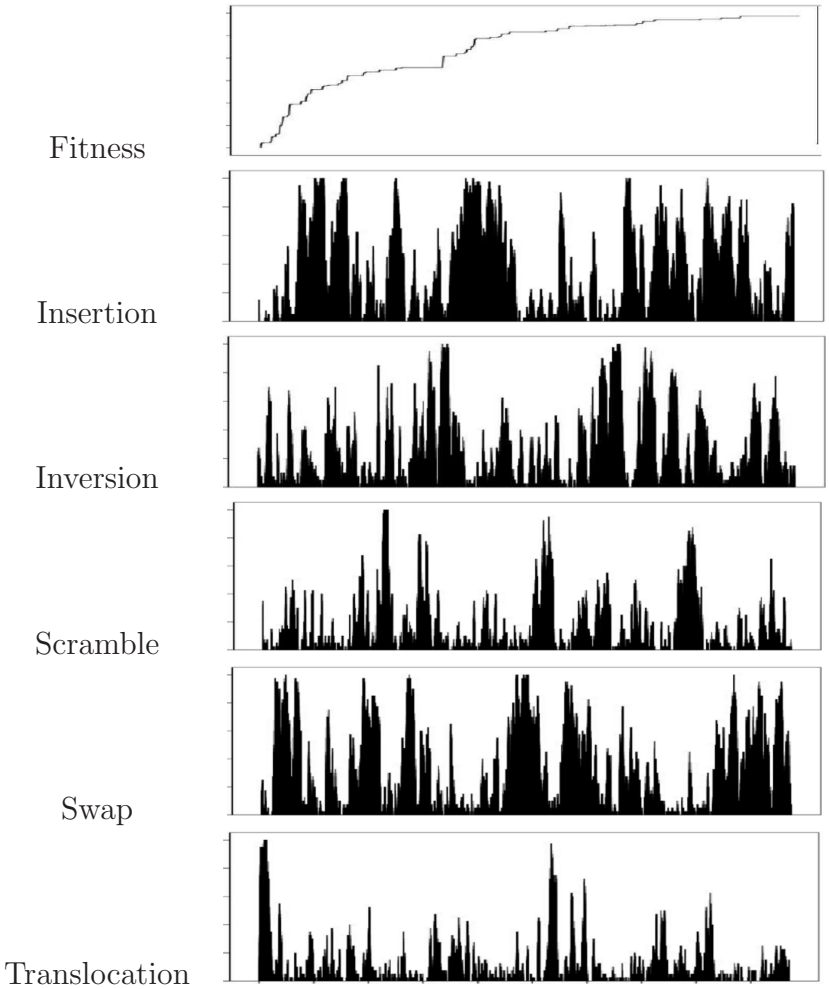

Figure 9: Self-adaption of the Mutation Operator

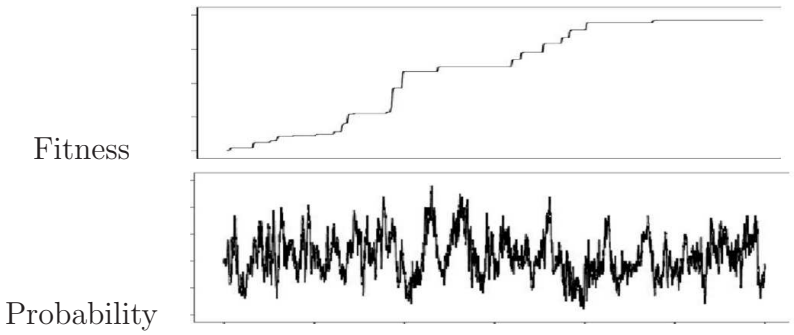

Figure 10: Self-adaption of the Mutation Probability with the SWAP mutation operator

search. The translocation operator comes into play very early on; at the beginning of the search it must be moving good partial solutions around and fixing broken constraints. Swap and insertion dominate improving the quality of the solution with the inversion operator playing a lesser role. Scramble seems to come into play when the other operators are unable to move out of local Optima. It seems to behave in a similar way as allowing immigration [56] of new population members, i.e. adding new blood to the population. The inversion operator may be playing a similar role, but in a much subtler way. It is clear from Figure 9 that the self-adaption of the mutation operator has allowed this search to escape from local Optima. This happens because each of the mutation operators sees a different fitness landscape. Where one mutation operator can see no way of escaping the local Optima in which it is trapped, another mutation operator can see a clear pathway ahead. Figure 10 shows the self-adaption of the mutation probability with a fixed mutation operator. It shows that the mutation probability has drifted up and down between its limits without any underlying pattern. It was expected from prior research that the mutation probability would increase over time, increasing the amount of mutation in the latter stages of the search. The fact that this did not happen may be explained by the shape of the fitness plot. It is made up of numerous plateaus, indicating local Optima, connected by sharp improvements. Whilst the algorithm is stuck in one of the local Optima and no improvements are found no preference for the mutation probability can be inferred by the algorithm. Instead, if the fitness plot had shown a smooth improvement instead of these large steps, then the search algorithm would have learned the benefit of a high mutation probability and that would spread throughout the population. Between them Figures 9 and

10 indicate that the behaviour of the self-adaption is dependent on the shape of the fitness landscape.

A further experiment was run where both the mutation operator and probability were allowed to self-adapt. This experiment was allowed to run for 500,000 iterations and it was expected that given more time the mutation probability might increase towards the end of the search. This however was not the case, the mutation probability behaved in exactly the same way as shown in Figure 10 indicating that self-adaption of the mutation operator plays a greater role in finding better solutions. Given the extra time, however the search produced a much superior solution. This demonstrates the power of self-adaption to avoid searches becoming stuck in local Optima and to do this without the need for the user to tune the search algorithm.

9. Conclusions and Further research

In this paper a rigorous mathematical model was developed to describe the VMSP. Mathematical models that describe route optimisation often cannot guarantee the creation of illegal sub-tours within a solution without the addition of sub-tour elimination constraints. To avoid this problem a reformulation of the mathematical model into a permutation representation was also developed which could then be solved using search techniques from the artificial intelligence domain. This reformulation

- removed the need to introduce sub-tour elimination constraints,
- removed the need for hard constraints that prevented jobs, vehicles and personnel being multiply assigned,
- identified the need for time related soft constraints to be interpreted differently to allow for a graduated penalty when those constraints were broken.

This reformulation of the VMSP created valid solutions which the original mathematical model could not guarantee. However, reformulating the mathematical model as a permutation introduced its own problems; there are many permutation operators from which to choose, and they tend to be more complex than non-permutation operators. To lessen this problem the reformulation added the ability to automatically select which permutation operator and probability to use, via a self-adaption mechanism. The

self-adaption of the mutation operator played a greater role than that of the mutation probability in finding better solutions. In the VMSP a combination of the self-adaption of the mutation operator with a high mutation probability performed best leading to the conclusion that the fitness landscape is made up of numerous local Optima. The addition of self-adaption to the GA produced significantly better solutions with $> 99.9\%$ confidence and removed the need to select the mutation operator or probability.

A comparison of the performance of other search strategies applied to this problem could form a basis for future research. This research identified that changing the paradigm describing a problem can lead to its simplification which could be applied to other problem areas. This research has also touched on how the fitness landscape affects the performance of the search strategy. Examining and predicting the properties of different fitness landscapes and the best ways to search them will make an interesting area of future research.

References

[1] H. Bederina, M. Hifi, A hybrid multi-objective evolutionary optimization approach for the robust vehicle routing problem, Applied Soft Computing 71 (2018) 980–993.

[2] P. Toth, D. Vigo, Models, relaxations and exact approaches for the capacitated vehicle routing problem, Discrete Applied Mathematics 123 (1) (2002) 487–512.

[3] J. E. Bell, P. R. McMullen, Ant colony optimization techniques for the vehicle routing problem, Advanced engineering informatics 18 (1) (2004) 41–48.

[4] J. K. Lenstra, A. Kan, Complexity of vehicle routing and scheduling problems, Networks 11 (2) (1981) 221–227.

[5] A. Subramanian, P. H. V. Penna, E. Uchoa, L. S. Ochi, A hybrid algorithm for the heterogeneous fleet vehicle routing problem, European Journal of Operational Research 221 (2) (2012) 285–295.

[6] F. Ferrucci, S. Bock, M. Gendreau, A pro-active real-time control approach for dynamic vehicle routing problems dealing with the delivery of urgent goods, European Journal of Operational Research 225 (1) (2013) 130–141.

[7] A. Le Bouthillier, T. G. Crainic, A cooperative parallel meta-heuristic for the vehicle routing problem with time windows, Computers & Operations Research 32 (7) (2005) 1685–1708.

[8] D. M. Miranda, J. Branke, S. V. Conceição, Algorithms for the multi-objective vehicle routing problem with hard time windows and stochastic travel time and service time, Applied Soft Computing (2018) (2018) 66–79.

[9] A. Agra, M. Christiansen, R. Figueiredo, L. M. Hvattum, M. Poss, C. Requejo, The robust vehicle routing problem with time windows, Computers & Operations Research 40 (3) (2013) 856–866.

[10] T. Vidal, T. G. Crainic, M. Gendreau, N. Lahrichi, W. Rei, A hybrid genetic algorithm for multidepot and periodic vehicle routing problems, Operations Research 60 (3) (2012) 611–624.

[11] A. S. Tasan, M. Gen, A genetic algorithm based approach to vehicle routing problem with simultaneous pick-up and deliveries, Computers & Industrial Engineering 62 (3) (2012) 755–761.

[12] M. A. Figliozzi, An iterative route construction and improvement algorithm for the vehicle routing problem with soft time windows, Transportation Research Part C: Emerging Technologies 18 (5) (2010) 668–679.

[13] Z. Zhu, J. Xiao, S. He, Z. Ji, Y. Sun, A multi-objective memetic algorithm based on locality-sensitive hashing for one-to-many-to-one dynamic pickup-and-delivery problem, Information Sciences 329 (2016) 73–89.

[14] L. Pradenas, B. Oportus, V. Parada, Mitigation of greenhouse gas emissions in vehicle routing problems with backhauling, Expert Systems with Applications 40 (8) (2013) 2985–2991.

[15] R. Dondo, J. Cerdá, A cluster-based optimization approach for the multi-depot heterogeneous fleet vehicle routing problem with time windows, European Journal of Operational Research 176 (3) (2007) 1478–1507.

[16] E.-G. Talbi, Metaheuristics: from design to implementation, Vol. 74, John Wiley & Sons, 2009.

[17] S. H. Rubin, T. Bouabana-Tebibel, Y. Hoadjli, Z. Ghalem, Reusing the np-hard traveling-salesman problem to demonstrate that p˜ np, in: Information Reuse and Integration (IRI), 2016 IEEE 17th International Conference on, IEEE, 2016, pp. 574–581.

[18] R. Matai, S. P. Singh, M. L. Mittal, Traveling salesman problem: An overview of applications, formulations, and solution approaches, Traveling Salesman Problem, Theory and Applications (2010) 1–24.

[19] C. Chauhan, R. Gupta, K. Pathak, Survey of methods of solving tsp along with its implementation using dynamic programming approach, International Journal of Computer Applications 52 (4) (2012) 12–19.

[20] A. H. Land, A. G. Doig, An automatic method for solving discrete programming problems, in: 50 Years of Integer Programming 1958-2008, Springer, 2010, pp. 105–132.

[21] D. S. Johnson, L. A. McGeoch, The traveling salesman problem: A case study in local optimization, Local search in combinatorial optimization 1 (1997) 215–310.

[22] P. Tian, J. Ma, D.-M. Zhang, Application of the simulated annealing algorithm to the combinatorial optimisation problem with permutation property: An investigation of generation mechanism, European Journal of Operational Research 118 (1) (1999) 81–94.

[23] M. Dorigo, L. M. Gambardella, Ant colony system: a cooperative learning approach to the traveling salesman problem, IEEE Transactions on evolutionary computation 1 (1) (1997) 53–66.

[24] J. Yang, C. Wu, H. P. Lee, Y. Liang, Solving traveling salesman problems using generalized chromosome genetic algorithm, Progress in Natural Science 18 (7) (2008) 887–892.

[25] J. Lu, M. Xie, Immune-genetic algorithm for traveling salesman problem, Traveling Salesman Problem, Theory and Applications (2010) 81–96.

[26] H. Sengoku, I. Yoshihara, A fast tsp solver using ga on java, in: Third International Symposium on Artificial Life, and Robotics (AROB III98), 1998, pp. 283–288.

[27] D. Kaur, M. Murugappan, Performance enhancement in solving traveling salesman problem using hybrid genetic algorithm, in: Fuzzy Information Processing Society, 2008. NAFIPS 2008. Annual Meeting of the North American, IEEE, 2008, pp. 1–6.

[28] X. Chen, Y. Zhou, Z. Tang, Q. Luo, A hybrid algorithm combining glowworm swarm optimization and complete 2-opt algorithm for spherical travelling salesman problems, Applied Soft Computing 58 (2017) 104–114.

[29] S. Hore, A. Chatterjee, A. Dewanji, Improving variable neighborhood search to solve the traveling salesman problem, Applied Soft Computing 68 (2018) 83–91.

[30] E. Osaba, J. Del Ser, A. Sadollah, M. N. Bilbao, D. Camacho, A discrete water cycle algorithm for solving the symmetric and asymmetric traveling salesman problem, Applied Soft Computing 71 (2018) 277–290.

[31] J. Wang, O. K. Ersoy, M. He, F. Wang, Multi-offspring genetic algorithm and its application to the traveling salesman problem, Applied Soft Computing 43 (2016) 415–423.

[32] A. Eiben, J. Smith, Introduction to Evolutionary Computation, Springer, 2003.

[33] R. L. Haupt, S. E. Haupt, Practical genetic algorithms, John Wiley & Sons, 2004.

[34] J. H. Holland, Adaptation in natural and artificial systems. an introductory analysis with application to biology, control, and artificial intelligence, Ann Arbor, MI: University of Michigan Press, 1975.

[35] J. H. Holland, et al., Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence, MIT press, 1992.

[36] C. Darwin, On the origin of species by means of natural selection: or the preservation of favoured races in the struggle for life., John Murray, Albemarle Street, 1880.

[37] A. Eiben, Z. Michalewicz, M. Schoenauer, J. Smith, Parameter Control in Evolutionary Algorithms, in: Fernando G. Lobo, Cláudio F. Lima, Zbigniew Michalewicz (Eds.), Parameter Setting in Evolutionary Algorithms, Vol. 54 of Studies in Computational Intelligence, Springer Verlag, 2007, pp. 19–46.

[38] S. Meyer-Nieberg, H.-G. Beyer, Self adaptation in evolutionary algorithms, in: L. Lobo, Michalewicz (Eds.), Parameter Setting in Evolutionary Algorithms, Springer, 2007, pp. 47–76.

[39] H.-G. Beyer, The Theory of Evolution Strategies, Springer, Berlin, Heidelberg, New York, 2001.

[40] H.-P. Schwefel, Numerical Optimisation of Computer Models, Wiley, New York, 1981.

[41] T. Bäck, Self adaptation in genetic algorithms, in: F. Varela, P. Bourgine (Eds.), Toward a Practice of Autonomous Systems: Proceedings of the 1st European Conference on Artificial Life, MIT Press, Cambridge, MA, 1992, pp. 263–271.

[42] M. Glickman, K. Sycara, Reasons for premature convergence of self-adaptating mutation rates, in: 2000 Congress on Evolutionary Computation (CEC'2000), IEEE Press, Piscataway, NJ, 2000, pp. 62–69.

[43] M. Preuss, T. Bartz-Beielstein, Sequential parameter optimization applied to self-adaptation for binary-coded evolutionary algorithms, in: Parameter setting in evolutionary algorithms, Springer, 2007, pp. 91–119.

[44] J. Smith, T. Fogarty, Self adaptation of mutation rates in a steady state genetic algorithm, in: ICEC-96 [57], pp. 318–323.

[45] M. Serpell, J. E. Smith, Self-adaptation of mutation operator and probability for permutation representations in genetic algorithms, Evolutionary Computation 18 (3) (2010) 491–514.

[46] B. Julstrom, Adaptive operator probabilities in a genetic algorithm that applies three operators, in: Proceedings of the 1997 ACM Symposium on Applied Computing, ACM Press, New York, NY, 1997, pp. 233–238.

[47] D. Thierens, An adaptive pursuit strategy for allocating operator probabilities, in: Proceedings of the 2005 Conference on Genetic and Evolutionary Computation, ACM Press, New York, NY, 2005, pp. 1539–1546.

[48] C. Reeves, Landscapes, operators and heuristic search, Annals of Operations Research 86 (1999) 473–490.

[49] T. Schiavinotto, T. Stutzle, A review of metrics on permutations for search landscape analysis, Computers and Operations Research 34 (10) (2007) 3143–3153.

[50] C. Simons, Interactive evolutionary computing in early lifecycle software engineering design, Ph.D. thesis, University of the West of England (2011).

[51] J. E. Smith, A. R. Clark, A. T. Staggemeier, M. C. Serpell, A genetic approach to statistical disclosure control, IEEE Transactions on Evolutionary Computation 16 (3) (2012) 431–441.

[52] Z. Ren, H. Jiang, J. Xuan, Z. Luo, Hyper-heuristics with low level parameter adaptation, Evolutionary computation 20 (2) (2012) 189–227.

[53] M. Zaefferer, J. Stork, T. Bartz-Beielstein, Distance measures for permutations in combinatorial efficient global optimization, in: International Conference on Parallel Problem Solving from Nature, Springer, 2014, pp. 373–383.

[54] J. Smith, On the interaction between self-adaptive mutation and memetic learning, in: The Proceedings of The 50th Annual Convention Of The Society For The Study Of Artificial Intelligence And The Simulation Of Behaviour (AISB), 2014.

[55] V. A. Cicirello, The permutation in a haystack problem and the calculus of search landscapes, IEEE Transactions on Evolutionary Computation 20 (3) (2016) 434–446.

[56] M. C. Moed, C. V. Stewart, R. B. Kelly, Reducing the search time of a steady state genetic algorithm using the immigration operator, in: 1991 Third International Conference on Tools for Artificial Intelligence, IEEE, 1991, pp. 500–501.

[57] Proceedings of the 1996 IEEE Conference on Evolutionary Computation, IEEE Press, Piscataway, NJ, 1996.