Volume 29 (1), pp. 55–73
http://www.orssa.org.za

ORiON
ISSN 0529-191-X
©2013

Metaheuristic approaches to order sequencing on a unidirectional picking line

AP de Villiers[*] J Matthews[*] SE Visagie[†]

Received: 13 September 2012; Revised: 6 December 2012; Accepted: 7 December 2012

Dedication to Emeritus Professor Theodor Stewart

Theo Stewart is one of the founders of OR in South Africa and has since played a long and active role in the OR community, both locally and internationally. His legacy in the South African OR community counts amongst the richest. He has served OR as practitioner, educator, researcher, president and member of ORSSAs executive committee, founding editor of ORiON and mentor to numerous students. It is an honour to dedicate this paper to Theo on his seventieth birthday. Thank you for your contribution. Veels geluk, Theo, mag jy goeie gesondheid behou en nog baie verjaarsdae vier!

Abstract

In this paper the sequencing of orders on a unidirectional picking line is considered. The aim of the order sequencing is to minimise the number of cycles travelled by a picker within the picking line to complete all orders. A tabu search, simulated annealing, genetic algorithm, generalised extremal optimisation and a random local search are presented as possible solution approaches. Computational results based on real life data instances are presented for these metaheuristics and compared to the performance of a lower bound and the solutions used in practise. The random local search exhibits the best overall solution quality, however, the generalised extremal optimisation approach delivers comparable results in considerably shorter computational times.

Key words: Order picking, unidirectional carousels, metaheuristics.

1 Background

In general, order picking systems involves the process of clustering and scheduling customer requests, assigning stock to locations, releasing pick instructions to the floor, picking the stock from the locations as well as the disposal of the picked stock [8]. An order picking system in a distribution centre (DC) operated by Pep Stores LTD (Pep) is considered in this paper. Pep is a major South African retail company selling predominantly apparel. Pep has approximately 1 500 retail outlets in five Southern African countries.

[*]Departments of Logistics, University of Stellenbosch, Private Bag X1, Matieland, 7602, South Africa.
[†]Corresponding author: Departments of Logistics, University of Stellenbosch, Private Bag X1, Matieland, 7602, South Africa, email: svisagie@sun.ac.za

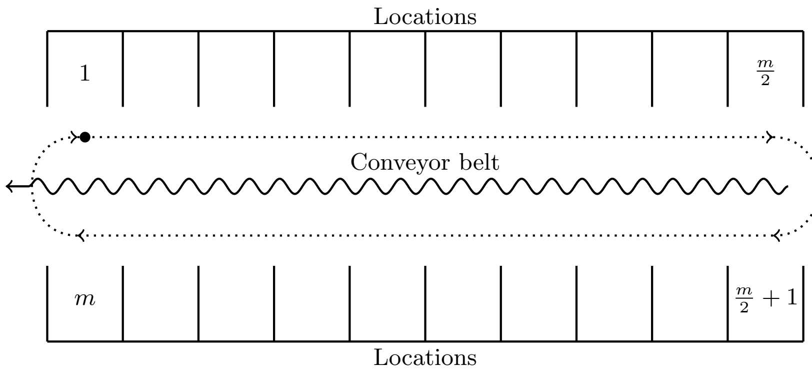


Figure 1: *A schematic representation of the layout of a picking line containing m locations.*

The order picking system considered here functions in waves. A wave is a set of stock keeping units (SKUs) in conjunction with the set of orders (requests from retail outlets). Each order may be viewed as a set of requests, for SKUs in a wave, by a single retail outlet. All the orders in a wave are picked in a single run. All the SKUs in a wave are therefore completely picked for all orders during a single wave. After completion of a wave, a picking line is populated with a new set of SKUs before a new wave of picking commences.

A picking line is the configuration of fixed locations around a conveyor belt. SKUs are placed in these locations for easy access when SKUs are collected for orders. A schematic representation of a picking line containing m locations is displayed in Figure 1. A SKU is stored in a single location and only SKUs within the same wave may be placed on the same picking line. Pickers must travel in a clockwise direction around the conveyor belt while SKUs are collected.

The Order Sequencing Problem (OSP) that is considered here determines the sequence in which orders must be picked to minimise the total distance travelled by the pickers during a single wave of picking. It is assumed that the positions of the SKUs are known, when orders are sequenced.

Each order consists of a set of SKUs together with the number of units required of each SKU. A picker must sequentially visit each location that contains a required SKU and pick the required number of units until all SKUs in that order have been picked. A picker may only start a new order once all the SKUs of the current order are completely picked.

A popular approach to order picking is to make use of a fast-pick (or forward pick area). A fast-pick area is a sub-region of the warehouse in which one concentrates high frequency SKUs (*i.e.* SKUs with a high demand) within a small physical space [4]. The picking line cannot, however, be considered as a fast-pick (or forward pick) area as the planning department in Pep releases a wave as soon as new SKUs have arrived at a DC regardless of SKU demand and size. In addition, locations on a picking line contain a fixed set of SKUs for only one wave, while locations in a forward pick area are permanently stocked with the same SKUs.

The order picking system in the Pep DC cannot be modelled as a typical carousel system either. Carousel systems in literature are typically bi-directional [21], and the composition of the orders are not known in advance. In the DC considered here, the composition of all the orders are known before a wave of order picking commences because a wave of picking only starts once all the requirements for all the branches for the wave's set of SKUs are known. This deterministic nature of the orders distinguishes this order picking system from the carousel systems found in literature.

The remainder of this paper is structured as follows. In the next section an overview of the OSP is presented followed by an insertion heuristic that is used by all of the metaheuristics presented in §3 to obtain initial solutions. In §3.2–3.6 five metaheuristic approaches are presented. The results of the metaheuristics applied to real life data instances follows in §4 and the paper is concluded in §5.

2 An overwiew of the OSP

The OSP is a variant of an equality generalised travelling salesman problem (E-GTSP) [10]. The E-GTSP is an $\mathcal{NP}$-hard problem and exact solution approaches takes too long to solve to be considered for practical implementation [14]. Assignment heuristics for this formulation was presented by De Villiers & Visagie [10], while a tight lower bound for this problem was presented by Matthews & Visagie [15]. A feasible solution within one pick cycle of this lower bound can always be calculated.

The following definitions and notation are required to describe the metaheuristics in this paper. The *span* of an order is the smallest set of locations that must be passed to complete the order, given a starting location.

Let $\mathcal{O}_k$ be the number of different locations that has to be visited to complete order k. A span for an order k starting at location i may be represented by $S_k = \langle i, e_k^i \rangle$, where i is the starting location and e_k^i is the closest ending position of order k.

The *size of a span* is the number of locations that have to be traversed to complete the order if a starting position is specified. Each order may be assigned one starting point from all the possible locations within an order. The size of the span for order k starting at location i may be represented by

$$|S_k^i| = |\langle i, e_k^i \rangle| = \begin{cases} e_k^i - i & \text{if } i < e_k^i \\ m + e_k^i - i & \text{if } i \geq e_k^i, \end{cases}$$

where m is the total number of locations, i is the starting location of the span and e_k^i is the ending location of the span. Let the *minimum span*, $S_k^{\min}$, be a span of smallest size of order k. The concept of a preference span is used to simplify the process of solving the OSP. The *preference span(s)* of an order are the span(s) of an order which start at a location containing a required SKU for that order.

Any starting location for an order has a unique span associated with it, since an order must be completed once it is started. The *cut* of a location is the number of spans passing that location. The cut of each location represents a lower bound for the number of cycles needed to pick a set of spans since it represents the minimum number of times a location must be passed to complete the picking of all the spans considered. The *maximal cut* is the largest cut.

Matthews & Visagie [15] showed that the maximal cut represents a lower bound in terms of the number of cycles travelled for a set of spans. They also provided the subtour generation heuristic in Algorithm 1, which supplies a feasible solution within one cycle of the lower bound, when the spans have been identified for each order. By minimising the maximal cut over all combinations of spans, a lower bound to the OSP may be computed.

Algorithm 1: Subtour generation heuristic [15].

```
  Input  : A set of starting positions $\mathcal{S}$ and ending positions $\mathcal{E}$.
  Output : A set $\mathcal{T}$ of subtours that link up all the orders.
1 while All orders have not been allocated to a subtour do
2     Generate a new subtour $s^t$ with the first available unallocated order;
3     Let the current ending position of $s^t$ be location $i$;
4     if An unallocated order exists which has a starting location corresponding to $i+1 \mod m$ then
5         Add this order to the end of the uncompleted subtour;
6     else
7         Close the subtour by connecting the last order to the first order;
```

The size of the problem may be reduced by only considering the preference spans (as opposed to all the spans) of an order. When an order is not picked on a preference span a number of locations are traversed (at the start of the span) that does not contribute to the picking process in the order, resulting in a longer distance travelled for an order than is required.

It is easy to show that only preference spans need to be considered when the OSP is solved, since the subtour generation heuristic will, if necessary, increase the lengths of preference spans to link up spans while constructing a single tour. The subtour generation heuristic can effectively alter the starting locations of the orders. However, each order will still be picked on its selected span. The connectivity between orders may thus be ignored so that the solution structure changes from a $n \times n$ connectivity matrix to a worst case of an $n \times m$ assignment matrix. The sequencing of orders are performed only after each order has been awarded a starting location associated with a preference span.

3 Metaheuristic solution approaches

Let C represent the maximal cut for a given SKU configuration, where the preference spans are awarded to orders in a picking line, and N_C be the number of locations with the maximal cut. Let the solution structure be a vector $\boldsymbol{s}$ of starting locations, where the k^{th} element s_k of the vector $\boldsymbol{s}$ represents the starting location of order k, with respect to a single preference span of order k. From a given solution $\boldsymbol{s}$ (containing only starting locations) both C and N_C can be calculated.

In the next subsection a greedy heuristic is presented that will be used to generate an initial starting solution for all the metaheuristics considered. In the following subsections five different metaheuristic approaches are presented that may be used to solve the OSP.

3.1 A greedy maximal cut(s) insertion heuristic

The greedy heuristic in Algorithm 2 may be used to find good starting solutions for the metaheuristics presented here. It is initialised by iteratively assigning starting locations to all the orders on a picking line. When a starting location is awarded to an order, the maximal cut(s) are calculated. The starting location of an order is chosen in an attempt to minimise the maximal cut and as a secondary objective to minimise the number of maximal cuts.

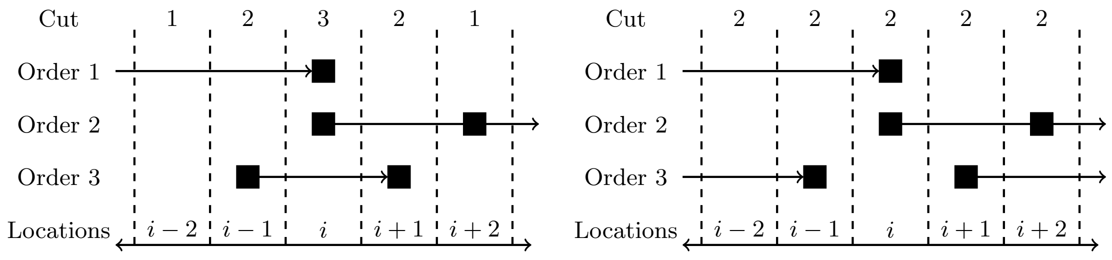
Algorithm 2: Greedy maximal cut insertion heuristic.

```
  Input   : A set of all the order $\mathcal{S}$. All the preference spans for each order.
  Output  : A vector $\boldsymbol{s}$ of starting locations awarded to each individual order.
1 Let $C \leftarrow 0$ and $N_C \leftarrow 0$;
2 while $\mathcal{S} \neq \emptyset$ do
3     Randomly select an order $k$ such that $k \in \mathcal{S}$;
4     if the selected order contains a preference span that will not increase the current value of $C$
      then
5         Assign that starting location to the $k^{\text{th}}$ entry in vector $\boldsymbol{s}$;
6     else
7         Select a preference span of order $k$ that results in the lowest value of $N_C$;
8     Update the value of $C$ and $N_C$;
9     $\mathcal{S} \leftarrow \mathcal{S} \backslash k$;
```

3.2 Tabu search

The fundamental principle that distinguishes tabu search from other search techniques, is that tabu search (TS) incorporates "intelligence." The tabu search directs an iterative search in a "good" direction so that the search for the optimal solution is not determined solely by chance. An initial solution is used as starting solution to the tabu search. The search space of a TS is the space of all possible solutions that can be considered during the search. The neighbourhood of the search space is the subset of solutions obtained by applying a single transformation to the search space of the current solution [12, 13]. A tabu list is a short-term memory structure which contains the solutions that have been visited in the recent past to aid the search to move away from previously visited sections of the search space and thus perform more extensive exploration [11]. The TS repeatedly replaces the current solution by selecting the solution with the best objective function in the neighbourhood of the current solution, that is not in the tabu list.

For the OSP considered here, a number of situations may arise where improvements may be achieved by altering an existing solution, when selecting different preference spans of orders. Orders that pass the maximal cut(s) but do not pick from that/those locations are considered for alterations (moves) to improve a current solution. If the maximal cut is determined by one location only (see Figure 2), an alteration to a span may result in a reduction of the maximal cut. However, if the maximal cut is determined by more than one location (see Figure 3), alterations are considered to reduce the number of locations effecting the maximal cut. If the number of locations effecting the maximal cut can be reduced to one, an alteration that reduces the maximal cut may be attempted. This results in two types of alterations (or moves) namely a *reducing move* (Figure 2) and a *shifting move* (Figure 3).



(a) Initial preference spans before a reducing move

(b) Preference spans after a reducing move was performed by altering the span of order 3.

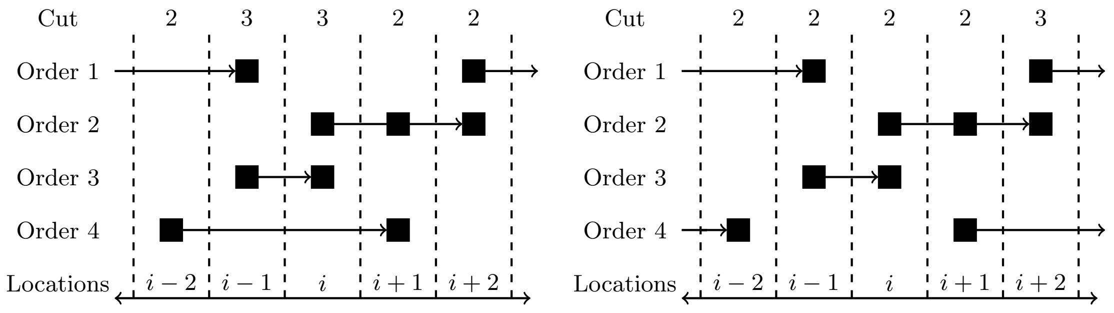
Figure 2: *An example of a reducing move, i.e. the altering of the span of an order to reduce the maximal cut. A filled square represents a location containing a SKU that is required by that order.*

Consider the example in Figure 2(a) where three orders are shown. Order 3 starts at location $i-1$ and ends at location $i+1$. It requires the SKUs in locations $i-1$ and $i+1$, but not any SKU in location i. Order 3 thus passes location i without picking from it. By altering the preference span of order 3 to start at location $i+1$, the maximal cut is reduced by 1. This reassigning of a preference span to reduce the maximal cut is referred to as a reducing move. Figure 2(b) displays the spans after the preference span of order 3 was altered.

If no improvement is possible, by altering preference spans of the orders to minimise the maximal cut(s), an attempt is made to change the location(s) containing the maximal cut(s). Figure 3(a) displays a situation in which the maximal cut cannot be reduced. It is, however, possible to alter the location(s) containing the maximal cut(s). The preference span of order 4 may be reassigned to start at location $i+1$. Before the alteration, locations $i-1$ and i contained the maximal cut and after the change location $i+2$ contains the only maximal cut. This move may be referred to as an shifting move. Figure 3(b) displays the spans after order 4 was altered.

If no improvement is possible, by altering preference spans of the orders to minimise the maximal cut(s), an attempt is made to change the location(s) containing the maximal cut(s). Figure 3(a) displays a situation in which the maximal cut cannot be reduced. It is, however, possible to alter the location(s) containing the maximal cut(s). The preference span of order 4 may be reassigned to start at location $i+1$. Before the alteration, locations $i-1$ and i contained the maximal cut and after the change location $i+2$ contains the only maximal cut. This move may be referred to as an shifting move. Figure 3(b) displays the spans after order 4 was altered.

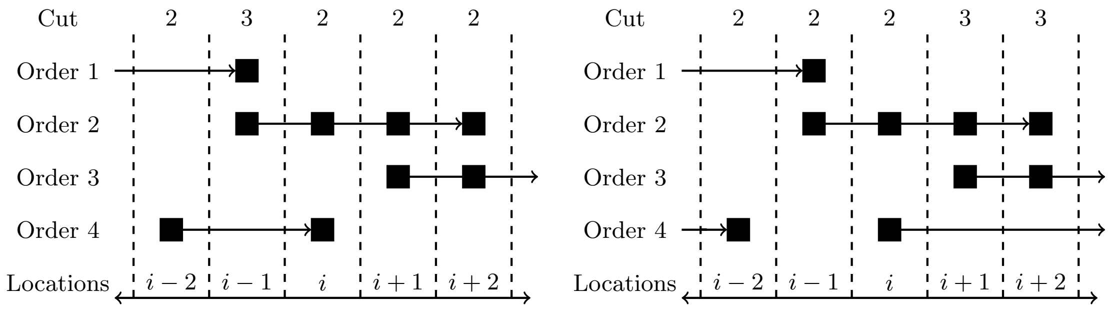
The locations containing a cut of one less than the maximal cut must also be examined. Figure 4(a) displays a situation which may seem similar to the situation in Figure 2. The current maximal cut is at location $i-1$. Orders 1 and 2 have to pick from location $i-1$, while order 4 does not require a SKU from location $i-1$. It may seem beneficial to alter the preference span of order 4 by starting at location i and ending at location $i-2$, decreasing the cut at location $i-1$ by one.



(a) Initial preference spans before a shifting move

(b) Preference spans after a reducing move was performed by altering the span of order 4.

Figure 3: *An example of a shift move i.e. altering the span of an order to shift the maximal cut. A filled square represents a location containing a SKU that is required by that order.*



(a) Initial preference spans before a reducing move

(b) Preference spans after a reducing move was performed by altering the span of order 4.

Figure 4: *An example of altering the span of an order to create new maximal cuts. A filled square represents a location containing a SKU that is required by that order.*

Notice that locations $i+1$ and $i+2$ both contain a cut of 2, one less than the current maximal cut. Figure 4(b) displays the configuration when the preference span of order 4 is altered. Although an attempt was made to decrease the maximal cut, the number of maximal cuts have increased from one to two. These situations are not considered as shifting moves.

The greedy heuristic in Algorithm 2 is utilized to obtain an initial solution. The TS is implemented by including all possible reducing moves in the neighbourhood. If the neighbourhood is not empty, any one of these starting locations is selected (as any one results in the same reduction in the maximal cut). The starting location of the corresponding order is updated and that order becomes tabu for a number of predetermined iterations.

If the neighbourhood is empty, it is increased to include all the starting locations of orders that do not increase the maximal cut(s) but shifts the number of maximal cuts. The neighbourhood is thus increased to contain all shifting moves. A single starting location is then randomly selected which shifts the maximal cut(s). The order corresponding to the starting location then becomes tabu. If no such starting location exists, the algorithm is terminated and the subtour generation heuristic is used to find a feasible tour with the current starting locations of the tabu search. Algorithm 3 contains the outline of the tabu

Algorithm 3: Tabu search.

```
Input  : A set of all the order $\mathcal{S}$, all the preference spans for each order considered and, a tabu
         list length $\ell$, the termination number $t$ and two boolean variables ReducingMove and
         ShiftingMove.
Output : A vector $\boldsymbol{s}$ of starting locations awarded to each individual order.
1  Find an initial solution $\boldsymbol{s}$ by means of the greedy insertion heuristic in Algorithm 2;
2  Calculate the values of $C$ and $N_C$;
3  Create a tabu list $\mathcal{L}$ such that $|\mathcal{L}| = \ell$;
4  repeat
5      ReducingMove ← false;
6      ShiftingMove ← false;
7      repeat
8          Select an order $k$ to be examined at random such that $k \in \mathcal{S}$;
9          if order $k$ contains a preference span that will not increase the current value of $C$ then
10             ReducingMove ← true;
11             Alter the preference span (i.e. starting location) of order $k$;
12     until all orders $k \in \mathcal{S}$ has been examined or ReducingMove is true ;
13     if ReducingMove is false then
14         repeat
15             Select an order $k$ to be examined at random such that $k \in \mathcal{S}$;
16             if order $k$ contains a preference span that results in the lowest value of $N_C^*$
               ($N_C^* \leq N_C$) then
17                 ShiftingMove ← true;
18                 Alter the preference span (i.e. starting location) of order $k$;
19         until all orders $k \in \mathcal{S}$ has been examined or ShiftingMove is true ;
20     Update the value of $C$ and $N_C$;
21     Update the tabu list $\mathcal{L}$ and set $\mathcal{L} \leftarrow \mathcal{L} \cup k$;
22 until no improvement in $t$ iterations or ReducingMove ← false and ShiftingMove ← false ;
```

search implementation used in this paper.

3.3 Random local search

Following the neighbourhood structure of the TS a random local search (RLS) is introduced. The algorithm initially proceeds in the same fashion as the TS. However, the moves that are considered are those that have preference spans which reduce either C or N_C. If no such move can be found the search space is increased to include all the orders. A random order is then selected and a random preference span assigned to it. After such a random move, only orders that affect C and N_C are considered again. The RLS will stop once a predetermined number of consecutive random moves are made. The RLS algorithm is summarised in Algorithm 4.

3.4 Simulated annealing

The simulated annealing algorithm (SA) is based on the analogy between the simulation of the annealing of solids and the problem of solving large combinatorial optimisation problems [20]. Annealing is the process of physically heating a solid material to impart high energy to it. At a high temperature, the solid becomes a liquid. The particles of the solid randomly arrange themselves in the liquid phase. A cooling phase follows where the

Algorithm 4: Random local search.

```
Input  : A set of all the orders $\mathcal{S}$, all the preference spans for each order and a maximum
         number of random moves $R$.
Output : A vector $\boldsymbol{s}$ of starting locations awarded to each individual order.
1  Find an initial solution $\boldsymbol{s}$ by means of the greedy insertion heuristic in Algorithm 2;
2  Calculate the values of $C$ and $N_C$;
3  repeat
4      if An order $k$ contains a preference span that will decrease the current value of $C$ then
5          Alter the preference span (i.e. starting location) of order $k$;
6          $r \leftarrow 0$;
7      else if An order $k$ contains a preference span that will decrease the current value of $N_C$ then
8          Alter the preference span (i.e. starting location) of order $k$;
9          $r \leftarrow 0$;
10     else
11         Increase the search space to include all orders;
12         Select a random order $k$ and a random preference span for order $k$;
13         Alter the preference span (i.e. starting location) of order $k$;
14         $r = r + 1$;
15     Update the value of $C$ and $N_C$;
16 until $r \geq R$ or no reducing or shifting move is possible ;
```

temperature is slowly decreased. By decreasing the temperature in a controlled manner the particle settle in a crystalised solid state at a minimum energy configuration.

In the annealing process, when thermodynamic balance is reached at a given temperature, the relative probability of a physical system to have an energy E in a multi-state system in thermodynamic equilibrium is preportional to the Boltzmann factor of $\exp\left(\frac{-E}{k_B T}\right)$, where k_B denotes the Boltzmann constant and T denotes the temperature [1].

The Metropolis algorithm is used to enable the system (solid) to reach a thermal equilibrium whenever the temperature is lowered. Given the structure of the system (the current layout/state of the components of the solid), the system is subject to an alteration (by exchanging components of this solid). If this alteration improves the objective function (advancing to a thermal equilibrium) of the system, it is accepted. If the alteration reduces the solution quality by ΔE of the objective function, it may be accepted with a probability of $\exp\left(\frac{-\Delta E}{T}\right)$. Following this criterion the system eventually evolves into a thermal equilibrium at the temperature considered — this leads to a Boltzmann distribution of energy states [1, 11, 20].

The annealing process may be terminated when the system is *solidified* (the temperature reaches 0 or no moves are able to improve the solution quality). The system may also be reheated, essentially applying the annealing process again if the solidified state of the current system is not desirable.

The choice of valueing the exchange of starting locations is measured by a combination of the maximal cut and the number of locations containing the maximal cut. The energy E_x for a given state x is

$$E_x = C_x + \frac{N_C^x}{m}, \tag{1}$$

where C_x denotes the maximal cut, N_C^x denotes the number of locations containing the maximal cut for a given state x and m is the total number of locations in the picking line.

The initial temperature T_0 is calculated by assigning a random starting location to each order. A total of 100 alterations are made at random calculating the average, $\overline{\Delta E}$, of the corresponding ΔE variations. The initial rate of acceptance τ_0 that is associated with the quality of the initial configuration must be selected. According to Dréo *et al.* [11] poor quality solutions are considered if $\tau_0 = 0.5$ (starting at a higher temperature), while better quality solutions are considered when $\tau_0 = 0.2$ (starting at a lower temperature). The value of T_0 is deduced from the relation

$$\exp\left(\frac{-\overline{\Delta E}}{T_0}\right) = \tau_0. \tag{2}$$

The Metropolis acceptance rule is used to address exchanges that decreases the solution quality. If $\Delta E > 0$, a number $r \in [0, 1]$ is randomly generated. The exchange is accepted if $r < \exp\left(\frac{-\Delta E+\alpha}{T}\right)$, where T indicates the current temperature, and α is the current number of iterations at a given temperature. The Metropolis rule is adapted in this way, since the greedy insertion heuristic in Algorithm 2 serves as a good general starting point and few moves are able to improve the solution quality. If too many moves are selected that do not improve the solution quality, the algorithm tends towards a random search.

Algorithm 5: Simulated annealing.

```
   Input  : A set of all the order $\mathcal{S}$, all the preference spans for each order considered and the
            number of orders to be altered during each iteration $\ell$ and the value of $\tau_0$.
   Output : A vector $\boldsymbol{s}$ of starting locations awarded to each individual order.
 1 Find an initial solution $\boldsymbol{s}$ by means of the greedy insertion heuristic in Algorithm 2;
 2 Calculate the values of $C$ and $N_C$;
 3 Evaluate $\overline{\Delta E}$;
 4 Calculate $T_0$ according to (2);
 5 repeat
 6     Select $\ell$ orders at random and an alternative preference span for each order based on (3);
 7     for $i \leftarrow 0$ to $\ell$ do
 8         Use the adapted Metropolis rule of acceptance to possibly alter solution;
 9         Update the fitness $E_x$;
10         if 100 perturbations attempted then
11             $T_{y+1} = 0.9 \cdot T_y$;
12 until no improvement is found in 5 iterations or until 100 iterations have been completed ;
```

During an iteration a number of orders are randomly selected. The starting locations of these orders may be altered. An order k requests $|\mathcal{O}_k|$ different SKUs resulting in $|\mathcal{O}_k|$ possible starting locations associated with these preference spans. Each of these starting positions is awarded a probability relative to the minimum span of order k. The score $c_k^i = |S_k^{\min}|/|S_k^i|$ is then used to award a probability

$$p_k^i = c_k^i / \sum_{i \in \mathcal{O}_k} c_k^i \quad i = 1, 2, \ldots, n \tag{3}$$

for each starting location i of an order k. When an order k is selected for an alteration of its starting location, the probability p_k^i constitutes the probabilty of starting order k on location i.

A change in temperature occurs at the end of an iteration when either a number of exchanges are accepted or when a number of exchanges are attempted. The change in temperature is altered according to the geometric law $T_{y+1} = 0.9 \cdot T_y$. The algorithm is terminated after 5 successive iterations without an improvement in the solution quality or after a total of 100 iterations. Algorithm 5 displays the pseudo code for the simulated annealing process.

3.5 Genetic algorithm

Genetic Algorithms (GAs) are search techniques inspired by the biological evolution of species in nature [2, 3]. A GA commences with an *initial population*, where each individual has a certain fitness level, which measures the level of *adaptation* to the given objective. Adaptation is the evolutionary process whereby a population (or individual within the population) becomes better suited to its habitat [5, 11]. The population gradually evolves in successive *generations* due to selection pressure.

An initial population of possible solutions (chromosomes) must be generated. Chromosomes may be any solution to the problem and is usually expressed as a binary string. A chromosome contains *genes*. The genes are the elements that described the chromosome in its entirety. The chromosomes are observed and evaluated according to their fitness. A fitness function specifies the goodness of a solution (chromosome).

All the chromosomes in a population with a good fitness, or even the chromosomes with weaker fitnesses, are then allowed to create offspring [18]. This process is called the *crossover*. This reproduction process generates diversity in the gene pool [17]. The existing chromosomes breed to create new chromosomes (offspring), which will enter the following generation.

Mutation is a random change to a chromosome. This is usually used to achieve *genetic diversity*. Genetic diversity ensures that the population varies from one generation to another.

The starting locations of orders are modelled as genes, with a chromosome being a collection of starting locations for all orders. Initially starting locations for all the orders (genes) are iteratively awarded to each chromosome according to the heuristic in Algorithm 2. The sequence in which orders are awarded preference spans according to the heuristic in Algorithm 2 is altered for each chromosome to obtain a distinct range of starting solutions in an attempt to diversify the gene pool.

The fitness of each chromosome z is calculated as

$$f_z = C_z + \frac{N_C^z}{m}, \tag{4}$$

where m is the total number of locations, C_z denotes the maximal cut and N_C^z denotes the number of locations containing the maximal cut for chromosome z.

In this paper three types of crossovers are used to create offspring between two parents.

All the chromosomes are allowed to create offspring according to the ranked-based fitness selection criteria [11].

The first type of crossover consist of a random swop of genes (starting locations of orders) between two chromosomes and is called the *50/50 crossover*. A second type of crossover is to award a higher probability to one offspring receiving the shorter preference span. This crossover is referred to as the *ranked crossover*. When considering two parents, the probability of swopping two genes, i and j for an order k is determined by generating a random number $r \in [0, 1]$. If $r < |S_k^i|/(|S_k^i| + |S_k^j|)$ the two starting locations are swopped. With this rule a higher likelihood exists of creating one offspring containing the majority of the shorter preference spans and another offspring containing the longer preference spans. A final crossover is considered where the maximal cut(s) of the first parent are identified, called the *maximal cut traverse crossover*. A crossover is only made between two starting locations if the starting location of an order in the first parent currently crosses all the current maximal cut(s) associated with the fitness of the first parent.

Elitism is also implemented in the genetic algorithm. The best 2% of the solutions in a generation will directly enter the next generation. Furthermore, 1% of the best performing chromosomes during an iteration are selected for a mutation phase, where each gene is individually examined for any alteration in starting location (preference span) that may improve the solution quality. During each iteration a total of 30% of the worst performing chromosomes are replaced with new chromosomes generated by Algorithm 2. Algorithm 6 shows the pseudo code for the implementation of the genetic algorithm.

Algorithm 6: Genetic algorithm.

```
   Input   : A set of all the order $\mathcal{S}$, all the preference spans for each order considered and
             population size $P$ and the desired crossover approach.
   Output  : A vector $\boldsymbol{s}$ of starting locations awarded to each individual order.
1  Find a set of initial solutions by means of the greedy insertion heuristic in Algorithm 2. Calculate
   with the values of $C^z$ and $N_C^z$ for each chromosome $z = 1, 2, \ldots, P$;
2  Evaluate $f_z$ for each chromosome $z = 1, 2, \ldots, P$;
3  repeat
4      Rank the chromosomes in a decreasing manner in terms of their fitness;
5      for $z \leftarrow 0$ to $P - 1$ (with step size 2) do
6          Perform a crossover between chromosomes $z$ and $z + 1$ according to the specified crossover
           approach in the input;
7          Update the fitness $f_z$ and $f_{z+1}$;
8      Sort the chromosomes in decreasing order of fitness;
9      Perform mutation on the best 1% of the chromosomes;
10     Sort the chromosomes in decreasing order of fitness;
11     Perform elitism of the top 2% of chromosomes;
12     Replace 30% of the worst performing chromosomes with new solutions by means of the greedy
       insertion heuristic in Algorithm 2;
13 until no improvement in the fitness is found during 5 iterations ;
```

3.6 Extremal optimisation

Extremal optimization (EO) is an optimisation heuristic inspired by the proporty of self-organized criticality (SOC) from the field of statistical physics [22]. EO is based on the dynamics of non-equilibrium processes and in particular those exhibiting SOC, where better solutions emerge dynamically without the need for parameter tuning [7]. The Bak-Sneppens model of co-evolution of species exhibits SOC and was the inspiration for EO [22].

In contrast to a genetic algorithm which considers the entire gene pool, EO improves a single candidate solution by considering each component as co-evolving according to Darwinian principles [6]. Unlike simulated annealing, EO uses a non-equilibrium approach which requires few parameters that have to be altered. It may be modelled by having a single parameter, while maintaining competitive, and even superior, results with respect to elaborate optimisation algorithms [22]. In contrast to simulated annealing, EO takes the system far from equilibrium as it applies no decision criteria, and all new configurations are accepted indiscriminately.

A single solution is considered and is examined to determine the contributions of its genes to the overall fitness. Analogously to the Bak-Sneppens model, each gene g_x has its own fitness contribution $\lambda(g_x)$. The higher the value of $\lambda(g_x)$, the better the contribution.

Instead of always picking the weakest part of $\boldsymbol{g}$, Boettcher & Percus [6] selected a gene to be modified randomly in order to ensure that the algorithm does not become stuck in a local optima. The probability

$$P(y) \propto y^{-\tau} \tag{5}$$

is awarded to a gene g_x with a fitness rank of y_x. The higher the value of τ, the less prone it is to alter genes that are ranked lower.

The major concern with EO is the manner of determining the fitness contributions $\lambda(h_x \cdot g_x)$ for the elements $h_x \cdot g_x$. De Sousa *et al.* [9] extended EO to the Generalised Extremal Optimisation (GEO), by examining each gene in the search space, according to the following procedure:

1. Determine an initial solution (individual) $\boldsymbol{h}$ with a random gene construction $\boldsymbol{g}$ and set the current best known solution candidate f^*, *i.e.* $f^* = \boldsymbol{h} \cdot \boldsymbol{g}$.
2. For each gene $g_x \in \boldsymbol{g}$:
   (a) Toggle the value of each gene g_x to retrieve a solution $f'_x = \boldsymbol{h} \cdot \boldsymbol{g}'$.
   (b) Set the change in fitness at $\Delta f_x = f^* - f'_x$.
   (c) Return the gene to its original form.
3. Rank the genes in decreasing order in terms of Δf_x for $x = 1, 2, \ldots, X$.
4. Use (5) as *mutating probability* to determine which gene will be mutated. A random number r is uniformly generated in the range $[0, 1]$. If r is smaller than the mutating probability, the mutation is performed, otherwise the process is repeated until a gene is found to mutate, that yields a fitness of f'.
5. If f' is better than f^*, set $f^* = \boldsymbol{h} \cdot \boldsymbol{g}'$.
6. If the termination criterion has not yet been met, continue at Step 2.

The EO algorithm is implemented for the OSP by considering the starting locations of each order. Once again, to reduce the size of the problem, only preference spans of orders are considered. An initial solution to the problem is determined by means of Algorithm 2. This initial solution enables the EO algorithm to reach better solutions in fewer iterations. The implementation of the EO algorithm is based on the GEO approach used by De Souse *et al.* [9]. Each gene (starting location of an order) is considered and "toggled." An attempt is made (by toggling the starting location of an order) to find a more suitable starting location that will benefit the solution quality by considering implications that may arise as the algorithm progresses.

Before genes are altered, the maximal cut C of the current configuration is determined together with the number of maximal cuts in the current configuration N_C. For each gene the current starting location is compared to all other starting locations (of preference spans) of that order. An alternative starting location for an order is identified that may improve the solution quality or slightly reduce the solution quality based on a number of considerations.

During an iteration genes that will be considered for alteration are identified. The orders that do not traverse a maximal cut cannot improve the solution quality. Only genes that currently traverse maximal cut(s) are considered. Each of these genes g_x are awarded with a change in performance Δf_x, where the current preference span of the corresponding order g_x is compared with all its other preference spans and the best alternative preference span is identified and becomes the proposed preference span.

Once the change in performance is calculated for each considered gene g_x, the genes are ranked in decreasing order of Δf_x. Each gene g_x is awarded a probability of

$$P(y_x) = y_x^{-\tau}, \tag{6}$$

where y_x denotes the rank of gene g_x. These probabilities are then normalised to calculate a probability distribution where a higher likelihood exists of altering a gene that will improve the solution quality. A random number is used to determine the alteration. When the alteration is made, the cuts at each location is updated.

Algorithm 7: Generalised extremal optimisation.

```
Input  : All the preference spans for each order considered and a set of all the order $\mathcal{S}$, the value
         of $\tau$ and the stopping criteria $\ell$.
Output : A vector $\boldsymbol{s}$ of starting locations awarded to each individual order.
1  Find an initial solution $\boldsymbol{s}$ by means of the greedy insertion heuristic in Algorithm 2 and $f_x^*$;
2  repeat
3      for $x \leftarrow 0$ to $n$ do
4          Toggle the value of each gene $g_x$ and retrieve the solution $f_x' = \boldsymbol{h} \cdot \boldsymbol{g}'$;
5          Calculate $\Delta f_x$;
6      Rank the genes in decreasing order of $\Delta f_x$ for $x = 1, 2, \ldots, n$;
7      Use the mutating probability distribution in (6) to determine a gene for mutation;
8      if $f^* < f'$ then
9          Let $f^* = \boldsymbol{h} \cdot \boldsymbol{g}'$ be the new best solution;
10 until no improvement in the fitness is found in $\ell$ iterations ;
```

When an alteration is made with a positive fitness, the solution may remain unchanged. In some cases the solution quality may even decrease. These situations exist when a picking line contains multiple maximal cuts. Some maximal cuts are decreased but some are increased (the GEO algorithm will ensure that more maximal cuts are decreased than the number of maximal cuts increased). Algorithm 7 displays the pseudo code implementation of the GEO implementation used.

4 Results

All the algorithms were coded in Java [16] and executed on an Intel(R) Core(TM)2 Duo 3GHz with 3.7 GB RAM running Linux Ubuntu [19]. Initial parameter selection were based on suggestions from literature by Dréo *et al.* [11]. The parameters for the algorithms were then improved by means of sensitivity analysis around these suggested values. The specific parameter values of the algorithms are discussed in the following paragraphs.

The TS was tested by varying the tabu list length at 10%, 20%, 30% and 40% of the number of orders considered. A second parameter was introduced which terminated the algorithm if the maximal cut was not reduced after 50, 100 or 200 iterations. The best results are realised when the tabu list is fixed at 30% of the number of order considered. Insignificant savings are achieved when the algorithm is terminated at 200 iterations without improvement or 100 iterations without improvement. However, the solution quality decreases substantially when the termination criteria is set at 50. Thus only 100 iterations without a reduction of the maximal cut is considered, which requires significantly less computational time.

The SA is implemented by altering the percentage of preference spans during each iteration at 2%, 5%, 10% and 11%. The value of τ_0 is varied at 0.02, 0.05 and 0.1. The best results are produced when τ_0 is equal to 0.02 and when 10% of the preference spans are considered for alteration during every iteration. Considerable increases in compuational time are incurred when the number of preference spans that are altered are increased from 5% to 10% and even higher increases are incurred for an increase from 10% to 11%. Neither of these time increases justify the minor or no increase in solution quality associated with it.

The GA is tested for the three crossover approaches namely the *50/50 crossover*, *ranked crossover* and the *maximal cut traverse crossover*. The population size is also varied to be either 20, 40, 60 or 80. The population is relatively small compared to the number suggested in literature. These parameters are, however, selected since increasing the number of chromosomes does not yield significant improvements in the solution quality. The best solutions for the GA are achieved when the number of chromosomes are set to 60 and the *ranked crossover* selection rule is used. Solution quality does not differ considerably when the parameters are altered.

The GEO algorithm is tested by varying the value of τ over 1, 1.25, 1.5 and 1.75. The stopping criteria is tested for situations where no improvement was found in 100, 200 and 300 iterations. The best results are found for the GEO when $\tau = 1.5$ and when the stopping criteria is set to 200 iterations without improvement.

Table 1 contains the results for the metaheuristics considered. A total of 22 real life data sets provided by Pep are considered. Each data set is categorised as either large, medium or small for reporting purposes. Large data sets have a lower bound in excess of 900 orders, medium data sets require between 40 and 900 cycles when considering the lower bound, and the rest of the data sets are considered as small data sets. The lower bound

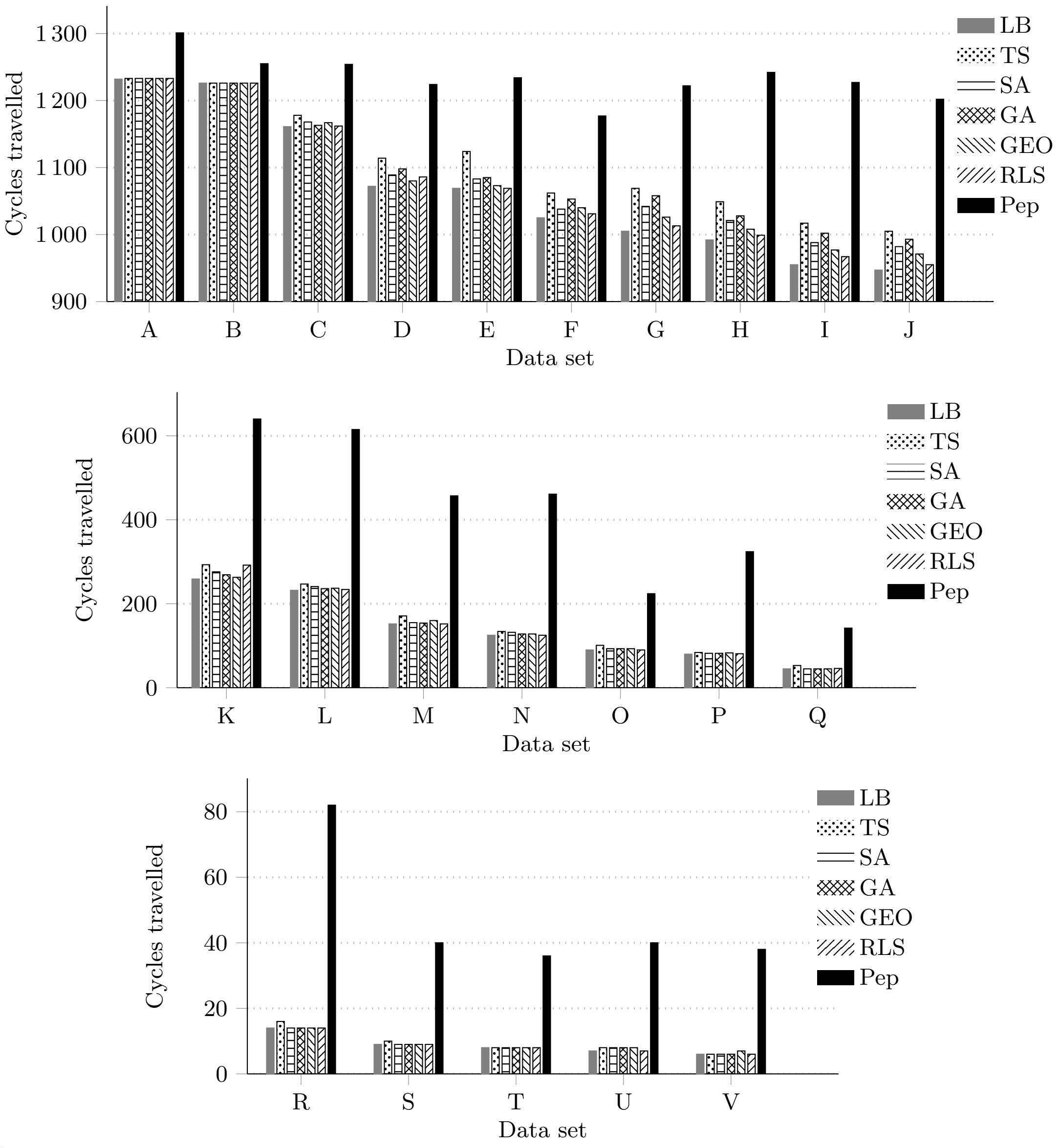


Figure 5: *A bar chart displaying the results (cycles travelled) obtained in Table 1. The number of cycles travelled for the lower bound (LB), tabu search (TS), simulated annealing (SA), genetic algorithm (GA), generalised extremal optimisation (GEO) and random local search (RLS) for each data set are compared to the historical results from Pep.*

	Data set	Size (O, L)	Lower bound	TS	SA	GA	GEO	RLS	Pep
Large	A	(1 262, 49)	1 232	**1 233**	**1 233**	**1 233**	**1 233**	**1 233**	1 301
	B	(1 264, 54)	1 226	**1 226**	**1 226**	**1 226**	**1 226**	**1 226**	1 255
	C	(1 265, 51)	1 161	1 178	1 168	1 163	1 167	**1162**	1 254
	D	(1263, 56)	1 072	1 114	1 089	1 098	**1 080**	1 086	1 224
	E	(1 264, 51)	1 069	1 124	1 083	1 085	1 073	**1 069**	1 234
	F	(1 258, 55)	1 025	1 062	1 038	1 053	1 040	**1 031**	1 177
	G	(1 258, 53)	1 005	1 069	1 042	1 058	1026	**1013**	1 222
	H	(1 244, 54)	992	1 049	1 021	1 028	1 008	**999**	1 242
	I	(1 260, 56)	955	1 017	988	1 002	977	**967**	1 227
	J	(1 264, 56)	947	1 005	982	993	971	**955**	1202
Medium	K	(943, 63)	259	293	276	269	**263**	292	640
	L	(846, 56)	232	247	241	236	237	**234**	615
	M	(728, 51)	152	171	155	154	160	**152**	457
	N	(733, 55)	125	134	132	128	128	**125**	461
	O	(396, 63)	90	101	93	93	93	**90**	224
	P	(574, 48)	80	84	82	82	83	**81**	324
	Q	(242, 64)	45	53	**45**	**45**	**45**	46	142
Small	R	(158, 55)	14	16	**14**	**14**	**14**	**14**	82
	S	(89, 42)	9	10	**9**	**9**	**9**	**9**	40
	T	(82, 51)	8	**8**	**8**	**8**	**8**	**8**	36
	U	(90, 48)	7	8	8	8	8	**7**	40
	V	(80, 56)	6	**6**	**6**	**6**	7	**6**	38
	Total		11 711	12 208	11 939	11 991	11 856	11 805	15 437

Table 1: *The solutions (as number of cycles travelled) for the tabu search (TS), simulated annealing (SA), genetic algorithm (GA), generalised extremal optimisation (GEO) and random local search (RLS) for each data set. The historical results from Pep are listed in the last column. The number of orders (O) and the number of locations (L) are indicated for each data set. The best solution quality for each data set is indicated in boldface.*

is calculated by means of the maximal cut formulation by Matthews & Visagie [15]. The best performing metaheuristic is the RLS which obtains on average 0.80% more cycles than the lower bound over all data sets. The GEO approach is in a close second place and delivers results that is on average within 1.24% of the lower bound. The TS delivers the worst performance of all the metaheuristics considered and is on average 4.24% from the lower bound over all the data sets considered.

Table 2 displays the computational times in milliseconds for the results in Table 1. The TS is the fastest metaheuristic on average over all data sets considered. The TS reaches a local optimum quickly whereafter no reducing move is found that reduces the maximal cut. The RLS requires on average the most computational time to solve an instance provided by Pep. The GEO displays favourable computational times, especially for medium and small data sets.

Although the RLS provides on average the best quality solutions for the data considered, the GEO delivers on average competitive solutions with respect to the RLS in much shorter computational times. Figure 5 displays bar charts of the solution quality for the large, medium and small data sets, respectively as given in Table 1. For medium and small data

	Data set	Size (O, L)	TS	SA	GA	GEO	RLS
Large	A	(1 262, 49)	672	5 380	2 547	1 594	6 376
	B	(1 264, 54)	333	5 625	2 012	1 232	9 931
	C	(1 265, 51)	273	16 754	6 478	1 308	5 812
	D	(1 263, 56)	148	24 692	2 312	6 178	17 282
	E	(1 264, 51)	147	33 093	6 679	2 473	22 290
	F	(1 258, 55)	173	29 430	2 331	5 075	15 259
	G	(1 258, 53)	144	27 299	4 497	7 903	25 513
	H	(1 244, 54)	150	48 500	1 954	11 486	24 188
	I	(1 260, 56)	169	42 712	2 267	9 584	13 412
	J	(1 264, 56)	124	26 845	5 490	10 943	10 628
Medium	K	(943, 63)	89	5 110	1 521	718	1 099
	L	(846, 56)	57	969	2 339	581	541
	M	(728, 51)	41	619	414	220	751
	N	(733, 55)	53	416	812	213	325
	O	(396, 63)	43	152	381	80	715
	P	(574, 48)	44	143	888	130	176
	Q	(242, 64)	21	67	145	35	168
Small	R	(158, 55)	11	24	42	27	54
	S	(89, 42)	9	14	18	10	12
	T	(82, 51)	7	13	21	8	24
	U	(90, 48)	7	12	17	9	17
	V	(80, 56)	8	13	22	10	33
	Total		2 723	267 882	43 187	59 817	154 606

Table 2: *Computational times in milliseconds for the results in Table 1. The number of orders (O) and the number of locations (L) are indicated for each data set. The results of the tabu search (TS), simulated annealing (SA), genetic algorithm (GA), generalised extremal optimisation (GEO) and random local search (RLS) are dispalyed.*

sets, the various approaches deliver approximately the same results, in some cases finding solutions equal to the lower bound. All the approaches outperforms the number of cycles travelled by Pep for all the data sets.

5 Conclusion

An order picking system in a DC owned by Pep stores was investigated. The problem was categorised into three tiers. Metaheuristic approaches are considered to achieve close-to-optimal solutions in considerably shorter time frames than the approach used by Matthews & Visagie [15] and better solution quality than the algorithms presented by De Villiers & Visagie [10]. A greedy heuristic is used to obtain a good initial solution for all the meta-heuristics considered. A tabu search, simulated annealing, genetic algorithm, extremal optimisation approach and a random local seacrh were developed to solve the OSP. The random local search delivers the best results, while the tabu search was able to solve each instance considered in the shortest time frame, but the method that produces the best solution quality relative to solution times is the GEO approach.

References

[1] Aarts E & Korst J, 1990, *Simulated annealing and Boltzmann machines: A stochastic approach to combinatorial optimization and neural computing*, John Wiley & Sons, New York (NY).

[2] Alba E, Blum C & Roli A, 2005, *An introduction to metaheuristic techniques*, pp. 3–42 in *Parallel metaheuristics: A new class of algorithms*, John Wiley & Sons, Hoboken (NJ).

[3] Bäck T & Hoffmeister F, 1991, *Extended selection mechanisms in genetic algorithms*, Proceedings of the Fourth International Conference on Genetic Algorithms, San Mateo (CA), pp. 710–719.

[4] Bartholdi JJ, Hackman ST, 2010, *Warehouse & distribution science*, Release 0.92, [Online], [Cited March 24th, 2010], Available from `http://www.tsp.gatech.edu/problem/index.html`.

[5] Blum C & Roli A, 2003, *Metaheuristics in combinatorial optimization: Overview and conceptual comparison*, ACM Computing Surveys, **35(3)**, pp. 268–308.

[6] Boettcher S & Percus AG, 1999, *Extremal optimization: Methods derived from co-evolution*, Proceedings of the Genetic and Evolutionary Computation Conference, San Francisco (CA).

[7] Boettcher S & Percus AG, 2001, *Extremal optimization for graph partitioning* , Physical Review E, **64(2)**, pp. 1–13.

[8] De Koster R, Le-Duc T & Roodbergen K J, 2007, *Design and control of warehouse order picking: A literature review*, European Journal of Operations Research, **182(2)**, pp. 481–501.

[9] De Sousa FL, Vlassov V & Ramos FM, 2004, *Generalized extremal optimization: An applicationin heat pipe design*, Applied Mathematical Modelling, **28(10)**, pp. 911–931.

[10] De Villiers AP & Visagie SE, 2011, *Toewysingsheuristieke om die volgorde van bestellings vir 'n uitsoeklyn te bepaal*, LitNet Akademies (Natuurwetenskappe), **9(1)**, pp. 1–22.

[11] Dréo J, Pétrowski A, Siarry P & Taillard E, 2006, *Metaheuristics for hard optimisation: Methods and case studies*, Springer-Verlag, Berlin.

[12] Glover F, 1986, *Future paths for integer programming and links to artificial intelligence*, Computers and Operations Research, **13(5)**, pp. 533–549.

[13] Glover F, 1990, *Tabu search — Part II*, Operations Research Society of America, **2(1)**, pp. 4–32.

[14] Gutin G & Yeo A, 2003, *Assignment problem based algorithms are impractical for the generalized TSP*, Australasian Journal of Combinatorics, **27(1)**, pp. 149–153.

[15] Matthews J & Visagie SE, 2011, *Order sequencing on a unidirectional cyclical picking line*, [Submitted].

[16] Sun Mircosystems, *Java*, [Online], [Cited March 26th, 2010], Available from `http://java.sun.com`.

[17] Patnaik LM, 1994, *Genetic algorithms: A survey*, Computer, **27(6)**, pp. 17–26.

[18] Qi X, 1994, *Theoretical analysis of evolutionary algorithms with an infinite population size in continuous space part I: Basic properties of selection and mutation*, IEEE Transactions of Neural Networks, **5(1)**, pp. 102–119.

[19] Ubuntu, [Online], [Cited March 15th, 2011], Available from `http://www.ubuntu.com`.

[20] Van Laarhoven PJM & Aart EHL, 1988, *Simulated annealing: Theory and applications*, D. Reidel Publishing Company, Dordrecht.

[21] Vickson RG & Fujimoto A, 1996, *Optimal storage locations in a carousel storage and retrieval system*, Location Science, **4(4)**, pp. 237–245.

[22] Wiese T, 2011, *Global optimization algorithms: Theory and application*, 2nd Edition, [Online], [Cited May 31st, 2011], Available from `http://www.it-weise.de/`.