THE UNCAPACITATED SWAPPING PROBLEM ON A LINE AND ON A CIRCLE

by

S. Anily
A. Pfeffer

Working Paper No 10/2011 July 2011

Research No. 01200100

This paper was partially financed by the Henry Crown Institute of Business Research in Israel.

The Uncapacitated Swapping Problem on a Line and on a Circle

Shoshana Anily[1]and Aharona Pfeffer[2]
Faculty of Management
Tel Aviv University
Tel Aviv 69978, Israel.

Abstract

The *uncapacitated swapping problem* is defined by a graph consisting of n vertices, and m object types. Each vertex of the graph is associated with two object types: the one that it currently holds, and the one it requires. Each vertex holds or requires at most one unit of an object. The problem is balanced in the sense that for each object type, its total supply equals its total demand. A vehicle of unlimited capacity is assumed to ship the objects in order to fulfill the requirements of all vertices. The objective is to find a shortest route along which the vehicle can accomplish the rearrangement of the objects, given designated initial and terminal vertices. The uncapacitated swapping problem on a general graph, including a tree graph, is known to be *NP-Hard*. In this paper we show that for the line and circle graphs, the problem is polynomially solvable: we propose an $O(n)$-time algorithm for a line and an $O(n^2)$-time algorithm for a circle.

Key Words: Swapping Problem, Stacker Crane Problem, Dial-a-Ride Problem.

[1]anily@post.tau.ac.il
[2]pfeffer@post.tau.ac.il

1 Introduction

The *Swapping Problem* (SP) is defined by a graph and a number of object types. More specifically, in the SP, each vertex of a graph may initially hold an object of a certain type, and it may desire an object of a possibly different type. The problem is balanced, meaning that for each object type, its total supply equals its total demand. A vehicle of a given capacity ships the objects among the vertices. The objective is to compute a shortest route along which the vehicle can accomplish the rearrangement of the objects. Anily and Hassin [1], where the SP was first introduced, considers a complete graph satisfying the triangular inequalities, and a vehicle of a unit capacity. The SP has many variations with respect to the underlying graph (line, circle, tree, or general graph), the vehicle's capacity (one unit, finite, or the *uncapacitated* case), number of vehicles, and the mobility of the objects as some objects may be *preemptive*, i.e., they can be dropped at some intermediate vertices before reaching their destination, and others may be *non-preemptive*, meaning that they must be shipped directly to their destination. This last distinction is irrelevant in the uncapacitated case. Applications for the SP include inventory repositioning involving a movement of a robot arm in a factory or a warehouse, especially when dealing with a line or a circle graph, see Attalah and Kosaraju [5].

The SP is a generalization of two well-known routing problems with one unit of each object type and all objects being non-preemptive: In the *Stacker-Crane Problem*, see Frederickson et al. [10], a single vehicle of a unit capacity ships the objects, and in the *Dial-a-Ride Problem*, see Psaraftis [14], the rearrangement of the objects is performed by one or more capapcitated vehicles. In this paper we focus on the single vehicle case.

It is interesting to note that the unit capacity SP on a general graph, considered in Anily and Hassin [1], is NP-hard, where for some special graph structures the problem is polynomially solvable. In particular, the unit capacity SP is polynomial on a line, see Anily, Gendreau and Laporte [2], where on a circle, it is still an open question. For a comprehensive literature review on the unit capacity SP on various graphs, see Anily, Gendreau and Laporte [3]. The uncapacitated Dial-a-Ride Problem on a tree was proved to be NP-hard by de Paepe et al. [7], implying the same complexity result for the uncapacitated SP on a tree.

In this paper we develop low complexity algorithms for the uncapacitated SP on a line and on a circle, each consisting of n vertices: (i) for a linear graph, the algorithm is of linear complexity, i.e., it is $O(n)$; and (ii) for a circular graph, the algorithm is of complexity $O(n^2)$.

The paper is organized as follows: Section 2 contains some general notations and preliminaries for the uncapacitated SP on a line and on a circle. In Section 3, the line case is solved, and in Section 4 the circle case is solved. Section 5 contains some concluding remarks. The detailed code of the data preparation, the algorithms (including all procedures) is deferred to the Appendix.

2 Notations and Preliminaries

2.1 Line and circle

Let $G = (U, E)$ be a graph with n vertices, where $U = \{1, \ldots, n\}$, a is the initial vertex and b is the terminal vertex. The graph G is either a line, or a circle, where each vertex is connected to at most two other vertices. In both types of graphs the vertices $1, \ldots, n$ appear consecutively. In the line, vertex 1 is at the left end-point of the line. In the circle, vertex 1 is also connected to vertex n. Thus, the set of edges E in a line contains $n-1$ edges, and in a circle it contains n edges. More specifically, for a line $E = \{(i, i+1) : 1 \leq i \leq n-1\}$ and for a circle $E = \{(i, i+1) : 1 \leq i \leq n-1\} \cup \{(n, 1)\}$. The edges are rescaled so that the length of the line and the perimeter of the circle is one unit. Let $x[1], ..., x[n]$ denote the locations of the vertices on the graph, where $x[1] = 0$: On the line let $x[n] = 1$, and $x[i]$ represents the distance between vertex 1 and vertex i. W.l.o.g. assume that $x[a] \leq x[b]$. On the circle we consider only the case that the initial vertex coincides with the terminal vertex ($a = b$), and w.l.o.g. we assume $a = b = 1$, and call this vertex the *depot*. Generalization to $a \neq b$ is possible. On the circle let vertex $n+1$ be a dummy vertex that coincides with vertex 1, i.e., $x[n+1] = x[1] = 0$. Vertices $2, \ldots n$ are arranged consecutively in a clockwise direction from vertex 1, where $x[i]$ denotes the length of the clock-wise circular arc, which connects vertices 1 and i. Note that $x[i]$ is not necessarily the shortest distance between vertex 1 and i. W.l.o.g., we assume $x[i] \neq x[h]$ for $1 \leq i < h \leq n$. The next definition applies for both a line and a circle:

Definition 2.1. *A segment between two adjacent vertices is called an* *<u>interval</u>.*

The problem is also defined by m *object types* or simply, *objects*, indexed by $j = 1, \ldots, m$, in addition to the *null object* denoted by $j = 0$. Let $S = \{0, 1, \ldots, m\}$, and $S_{-0} = S \backslash \{0\}$ be the set of real objects. Each vertex

i is associated with two object $(\alpha_i, \beta_i) \in S^2$, which means that the vertex currently holds one unit of object α_i, and it demands one unit of object β_i. W.l.o.g. assume that $\alpha_i + \beta_i > 0$ for any i, $i \notin \{a, b\}$, as otherwise vertex i could be removed from the graph. The total number of units of object j is n_j, where $\sum_{j \in S} n_j = n$. The problem is balanced in the sense that for each object in S, its total supply equals its total demand. A feasible route starts with an empty vehicle leaving vertex a, and terminates at vertex b, after satisfying the demands of the vertices by objects that were loaded along the route. All along we assume that given a route, the vehicle loads the supply of each vertex at the first visit at the vertex, and unloads the demand at the last visit at the vertex. This assumption is legitimate in view of the fact that the vehicle has no capacity limits, and there are no time constraints.

2.2 The line:

In this subsection we present a few more definitions for the line:

Definition 2.2. *A directed path connecting vertex t to vertex h, is called an arrow with tail t and head h. An arrow is denoted by the ordered pair of its tail and its head locations, namely $(x[t], x[h])$.*

Definition 2.3. *An arrow $(x[t], x[h])$ is said to cross any vertex i for which $x[t] < x[i] < x[h]$ or $x[t] > x[i] > x[h]$.*

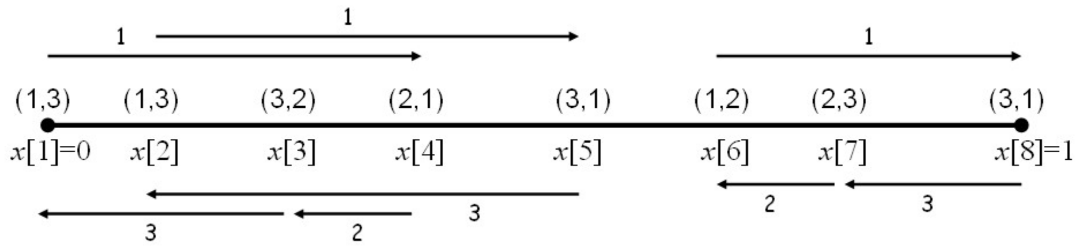
Following Anily et al. [2], which deals with the unit-capacity SP on a line, define for each object $j \in S_{-0}$, a *minimally balanced partition*:

Definition 2.4. *A minimally balanced partition of object $j \in S_{-0}$ is a partition of the set of all j's supply and demand vertices into minimum size consecutive sets so that each set is balanced, i.e., has an equal number of supply and demand vertices of object type j.*

In each set of the minimally balanced partition of object j, number the demand (supply) vertices from left to right, and define *unit-arrows* as follows:

Definition 2.5. *For each set in the minimally balanced partition of object $j \in S_{-0}$, let a j-unit-arrow, or simply a unit-arrow be an arrow connecting the i^{th} supply vertex to the i^{th} demand vertex in that set.*

All unit-arrows of a set in the minimally balanced partition of object j, follow the same direction, and together they cover all the intervals between



The minimally balanced partition of object type 1: $\{1, 2, 4, 5\}, \{6, 8\}$
The minimally balanced partition of object type 2: $\{3, 4\}, \{6, 7\}$
The minimally balanced partition of object type 3: $\{1, 2, 3, 5\}, \{7, 8\}$
$*$ At each vertex $i = 1, ..., 8$ we have its location, $x[i]$, and its supply and demand objects, (α_i, β_i).
$**$ The numbers on the unit-arrows refer to the object type associated with them.

Figure 1: *The unit-arrows of a SP.*

the leftmost and the rightmost vertices in that set. See Figure 1. Anily et al. [2] shows that for the unit capacity SP on a line there exists an optimal solution where the object held by the tail of each unit-arrow satisfies the demand of the head of this unit-arrow. The proof can easily be extended to any capacity, and in particular to the uncapacitated case.

Property 1. *There exists an optimal solution for the SP on a line where the object held by the tail of each unit-arrow satisfies the demand of the head of this unit-arrow.*

It is possible that a set in the minimally balanced partition for object j is a singleton. In such a case, the supply at the vertex is used to satisfy its demand, and the associated unit-arrow $(x[t], x[t])$ is said to be *degenerate.* Let $A = (x[t], x[h])$ represent a unit-arrow of some object, where t and h are the vertices of the tail and the head of the arrow, respectively. Let $h(t)$ denote the vertex index at the head of the unit-arrow whose tail is t. Note that for any tail t of a unit-arrow, the function $h(t)$ is well defined.

Definition 2.6. *A unit-arrow $A = (x[t], x[h(t)])$ for which $t < h(t)$ $(t > h(t))$ is referred to as a $\underline{\text{right unit-arrow}}$ ($\underline{\text{left unit-arrow}}$).*

Definition 2.7. *A $\underline{\text{right (left)-covered interval}}$ is an interval that is covered by at least one right (left) unit-arrow. An interval can be both right-covered and left-covered.*

We further define UA to be the set of all non-degenerate unit-arrows for all objects. UA contains at most n unit-arrows. In addition, let UA^R (UA^L) be the subset of right (left) unit-arrows, where $UA = UA^L \cup UA^R$. In order to avoid the need of sorting the sets of unit-arrows, we store them in arrays of n components: The array $\overline{UA}$ is defined such that $\overline{UA}(i) = h(i)$, if i is the tail of a non-degenerate unit-arrow, and $\overline{UA}(i) = 0$ if $\alpha_i = 0$. The arrays $\overline{UA}^R$ and $\overline{UA}^L$ are associated with the right and left unit-arrows, respectively, where $\overline{UA}^R(i) = \overline{UA}(i)$ if $\overline{UA}(i) > i$, and $\overline{UA}^R(i) = 0$, otherwise; $\overline{UA}^L(i) = \overline{UA}(i)$ if $\overline{UA}(i) < i$, and $\overline{UA}^L(i) = 0$, otherwise. In view of Property 1, the set UA is an optimal matching between the supplies and demands. As the capacity of the vehicle is not limited, when having two unit-arrows pointing to the same direction, where one is totally covered by the other (this may occur only if these unit-arrows are associated with different object types), then w.l.o.g. the covered unit-arrow can be removed from UA. This is true as the shorter unit-arrow can be served while traversing the longer. Therefore, before generating $\overline{UA}^R$ and $\overline{UA}^L$, remove from UA all the unit-arrows which are covered by others. For this sake, update the array $\overline{UA}$ by replacing the head of each covered unit-arrow by 0. From now on we assume that all covered unit-arrows have been removed before initiating the data. As will be seen in Section 3, a special attention is paid to unit-arrows that cross either the initial or the terminal vertices. Therefore, let:

Definition 2.8. *A $\underline{\text{right (left) crossing unit-arrow}}$ is a right (left) unit-arrow, which crosses either vertex a or vertex b.*

Definition 2.9. *For each unit-arrow $A = (x[t], x[h(t)])$ define (1) its $\underline{\text{tail-part}}$ as the distance between the tail of A and vertex a, and denote it by $tp(A) = |x[a] - x[t]|$, and (2) its $\underline{\text{head-part}}$ as the distance between the head of A and vertex b, and denote it by $hp(A) = |x[h(t)] - x[b]|$.*

The next property follows immediately from the fact that UA^R and UA^L do not contain any covered unit-arrows:

Property 2. *The right (left) crossing unit-arrows in UA^R (UA^L) satisfy the property that the longer their tail-part the shorter their head-part.*

Finally, we define:

Definition 2.10. *Two unit-arrows such that the head or the tail of one is crossed by the other, are called $\underline{\text{intersecting}}$. Intersecting unit-arrows can be in the same direction, or in opposite directions.*

In Section 3, a polynomial-time algorithm for solving the SP on a line for general initial and terminal vertices, is presented. In addition, simplified algorithms are provided for two special cases, where $a = b$, and $a = 0$, $b = 1$.

3 The SP on a line

Let V^*_{Line} be the optimal cost of the SP on a line. As all the vertices have to be visited by the vehicle, the following lower bound follows:

$$V^*_{Line} \geq 2 - (x[b] - x[a]) \geq 1 \tag{1}$$

It is easy to see that the middle expression in (1) is a tight lower bound. In order to derive an upper bound, note that a feasible tour can be obtained by going from vertex a to one end-point of the line, traverse the entire line back and forth, and then go to the terminal vertex b. If the tour starts by going left, the resulting length is $x[a] + 2 + x[b]$, and if the tour starts by going right, the resulting length is $2 + (1 - x[a]) + (1 - x[b]) = 4 - (x[a] + x[b])$. Thus,

$$V^*_{Line} \leq \min\{2 + x[a] + x[b]; 4 - (x[a] + x[b])\} = 3 - |x[a] + x[b] - 1| \leq 3 \tag{2}$$

The proposed algorithm is based on the following distinction:

Definition 3.1. *A feasible solution to SP on a line is said to be a <u>right (left) solution</u> if after leaving the initial vertex a, the vehicle visits vertex n (1) before visiting vertex 1 (n).*

Each feasible solution is either a right or a left solution. Accordingly, we also define *right (left) basic routes*, which is a path that is contained in any right (left) solution. See Figure 2.

Definition 3.2. *A <u>right basic route</u> consists of three segments: The first emanates from a to n, the second emanates from n to 1, and the third emanates from 1 to b. The total length of this basic route is $2 + x[b] - x[a]$.*

Definition 3.3. *A <u>left basic route</u> consists of three segments: The first emanates from a to 1, the second emanates from 1 to n, and the third emanates from n to b. The total length of this basic route is $2 + x[a] - x[b]$.*

If the right (left) basic route doesn't induce a feasible solution as is, then it must be augmented at a minimum cost in order to make it a feasible right (left) solution. The optimal solution is the best between the optimal left and optimal right solutions. In the next two subsections, the proposed algorithm deals with each of the two solution types separately. It finds the optimal cost for each, and returns the shortest feasible route in complexity $O(n)$. Let V^R (V^L) be the optimal cost of a right (left) solution, implying that $V^*_{Line} = \min\{V^R, V^L\}$.

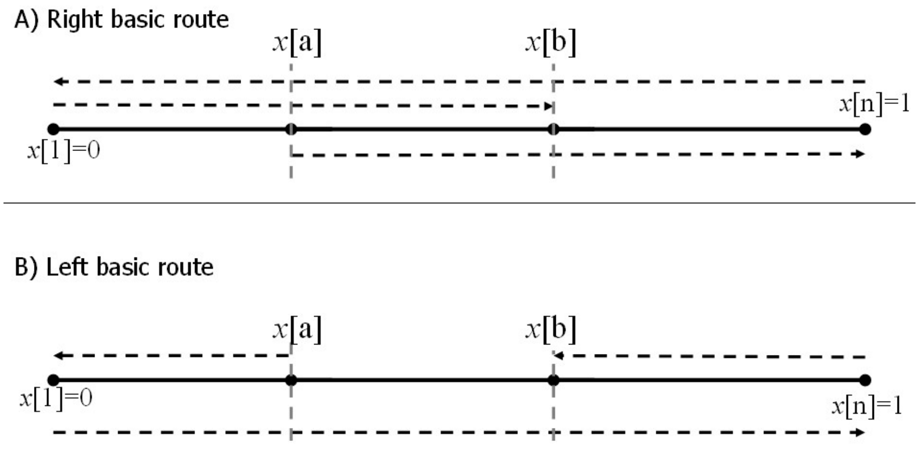
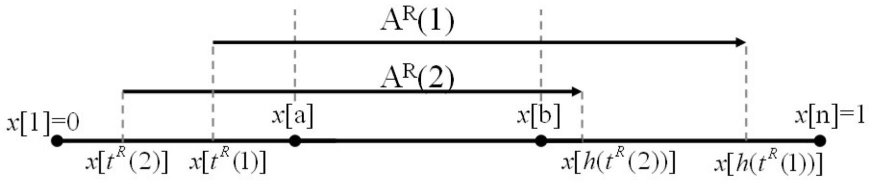


Figure 2: *The two types of basic routes*

3.1 Right solutions:

Following the right basic route allows servicing all the left unit-arrows while driving from vertex n to vertex 1, and also the right unit-arrows $(x[t], x[h(t)])$, which are covered by the arrow $(x[a], 1)$ or by the arrow $(0, x[b])$. However, if there exist right unit-arrows in UA^R, which cross both a and b, then the right basic route needs to be augmented: In such a case, supplies at the tails of these unit-arrows have already been collected along the right basic route, but they have not reached their destinations. Let C^R be the set of right crossing unit-arrows in UA^R that cross both a and b. I.e., $C^R = \{A = (x[t], x[h(t)]) \in UA^R | \ x[t] < x[a] < x[b] < x[h(t)]\}$. Let f^R be the cardinality of C^R. If $f^R = 0$, no augmentation is needed, and the right basic route is feasible as is. Otherwise, C^R is represented by an array $\overline{C^R}$ of size n: for $k = 1, \ldots, f^R$, $\overline{C^R}(k) = (t^R(k), h(t^R(k)))$, and $A^R(k) = [x(t^R(k), x(h(t^R(k))]$, where $t^R(k)$ denotes the vertex index at the tail of the k^{th} unit-arrow in $\overline{C^R}$, such that $1 \le t^R(k) < a \le b < h(t^R(k)) \le n$, and $t^R(1) > t^R(2) \ldots > t^R(f^R)$. In view of Property 2, $h(t^R(1)) > h(t^R(2)) \ldots > h(t^R(f^R))$. That means that $tp(A^R(1) < tp(A^R(2) < \ldots < tp(A^R(f^R))$, (see Figure 3). In order to serve $A^R(k) \in C^R$ along a tour that contains the right basic route, the vehicle, before starting the basic route, needs to go left to its tail, or after completing the basic route (hence already picking-up the supply at $t^R(k)$), it needs to go right to $h(t^R(k))$ to satisfy its demand, before terminating the tour at b. However, as the unit-arrows is C^R are intersecting crossing unit-arrows, the vehicle, if it heads left to $t^R(k)$ before starting the basic route, then on its way to $t^R(k)$, it picks up all the objects supplied between vertices a and $t^R(k)$.



$*$ Tail-parts: $tp(A^R(i)) = x[a] - x[t^R(i)]$, for $i = 1, 2$
$*$ Head-parts: $hp(A^R(i)) = x[h(t^R(i)) - x[b]$, for $i = 1, 2$.

Figure 3: *The right crossing unit-arrows*

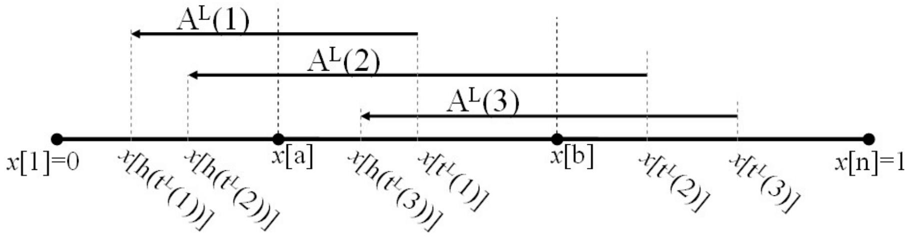
By doing so, it is able to serve all the right crossing unit-arrows $A^R(\ell)$, $\ell \leq k$ on the first segment of the right basic route. The objects at the tails of the other unit-arrows in C^R, i.e., $A^R(\ell)$, $k < \ell \leq f^R$, are picked-up only while traversing the second segment of the right basic route, and after reaching b while completing the right basic route, the vehicle continues to the farthest head of these unit-arrows in order to satisfy all unsatisfied demands. This incurs an additional cost of $2\max_{\ell > k} hp(A^R(\ell))$, which according to Property 2 equals $2hp(A^R(k+1))$.

Algorithm Right Basic Route computes V^R by going over the first f^R elements in array $\overline{C^R}(k) = (t^R(k), h(t^R(k)))$ and finds $\min_{k=1}^{f^R}(tp(A^R(k)) + hp(A^R(k+1)))$, which is the minimum augmentation cost. The algorithm returns V^R, which is the sum of the right basic route cost and the minimum augmentation cost. The complexity of the proposed algorithm is $O(n)$. In the next subsection we find the best Left solution.

3.2 Left solutions:

Following the left basic route (recall this structure in Figure 2B) allows a service to all the right unit-arrows in UA^R while driving from vertex 1 to n along the second segment of the basic route, and also a service to the left unit-arrows in UA^L, which are covered by (i) the arrow $(x[a], 0)$ on the first segment of the basic route, or (ii) the arrow $(1, x[b])$ on the last segment of the basic route. However, if there exist left crossing unit-arrows, or left unit-arrows covered by the arrow $(x[b], x[a])$, then the left basic route must be augmented. For this sake define four sets of left unit-arrows:

- $C_a^L = \{(x[t], x[h(t)]) \in UA^L | x[h(t)] < x[a] < x[t] \leq x[b]\}$
- $C_b^L = \{(x[t], x[h(t)]) \in UA^L | x[a] \leq x[h(t)] < x[b] < x[t]\}$
- $C_{ab}^L = \{(x[t], x[h(t)]) \in UA^L | x[h(t)] < x[a] < x[b] < x[t]\}$



$*$ Tail-parts: $tp(A^L(i)) = x[t^L(i)] - x[a]$ for $i = 1, 2, 3$
$*$ Head-parts: $hp(A^L(i)) = x[b] - x[h(t^L(i))]$, for $i = 1, 2, 3$.
$**$ In this example $A^L(2) \in C^L_{ab}$, thus $B = \emptyset$.

Figure 4: *The left crossing unit-arrows in C^L.*

- $B = \{(x[t], x[h(t)]) \in UA^L | x[a] \leq x[h(t)] < x[t] \leq x[b]\}$

By Property 2, $C^L_{ab} \neq \emptyset$ implies $B = \emptyset$. Let $C^L = C^L_a \cup C^L_b \cup C^L_{ab}$, and f^L, f^B, f^L_a, f^L_b, and f^L_{ab} be the cardinalities of C^L, B, C^L_a, C^L_b, and C^L_{ab}, respectively. As the number of unit-arrows is no greater than n, it must hold that $f^L + f^B \leq n$. Define an array $\overline{C^L}$ of size n, whose first f^L_a elements are associated with the crossing unit-arrows in C^L_a, the next f^L_{ab} elements are associated with the crossing unit-arrows in C^L_{ab}, and the last f^L_b elements are associated with the crossing unit-arrows in C^L_b: For $k = 1, \ldots, f^L$, let $\overline{C^L}(k) = (t^L(k), h(t^L(k)))$, where $a < t^L(1) < t^L(2) < \ldots < t^L(f^L) \leq n$. Using Property 2, $1 \leq h(t^L(1)) < h(t^L(2)) \ldots < h(t^L(f^L)) < b$. Let $A^L(k) = (x[t^L(k)], x[h(t^L(k))])$, $k = 1, ..., f^L$, (see Figure 4). Define $t^L(0) = a$, $tp(A^L(0)) = 0$, $h(t^L(f^L + 1)) = b$, and $hp(A^L(f^L + 1)) = 0$. In addition, define an array $\overline{B}$ of size n whose first f^B elements are associated with the unit-arrows in B: For $\ell = 1, \ldots, f^B$, $\overline{B}(\ell) = (t^B(\ell), h(t^B(\ell)))$, where $a < t^B(1) < t^B(2) < \ldots < t^B(f^B) \leq b$. By definition of B and Property 2, $a \leq h(t^B(1)) < h(t^B(2)) \ldots < h(t^B(f^B)) < b$.

Recall the definition of covered intervals, see Definition 2.7:

Definition 3.4. *A coverage of unit-arrows of B is a maximal set of unit-arrows of B, such that the collection of intervals, which are left-covered by these unit-arrows is convex, i.e., it is a consecutive collection of the intervals.*

Definition 3.5. *Each coverage of unit-arrows is associated with a coverage arrow, which is a left arrow that connects the two end points of the coverage.*

A procedure, named Coverage, is run if $f^B > 0$, in order to find all the coverage arrows of B. Let q ($q \leq f^B$) be the number of coverage arrows associated with B, and suppose they are named $AC(1), \ldots, AC(q)$.

Let $t^{AC}(i)$ and $h^{AC}(i)$ be the tail and the head of $AC(i)$, respectively, for $i = 1, \ldots, q$, such that $a < t^{AC}(1) < \ldots < t^{AC}(q) \leq b$. The removal of covered unit-arrows implies that $x[t^{AC}(1)] > x[t^L(f_a^L)]$, if $q \cdot f_a^L > 0$, and $x[h^{AC}(q)] < x[h(t^L(f_a^L + 1))]$ if $q \cdot f_b^L > 0$. Thus, the only coverage arrow that can cross vertex $t^L(f_a^L)$ is $AC(1)$, and similarly, the only coverage arrow that can cross vertex $h(t^L(f_a^L + 1))$ is $AC(q)$. Procedure Coverage uses two indicators, I_a and I_b. If $AC(1)$ crosses $t^L(f_a^L)$, the procedure returns $I_a = 1$, otherwise $I_a = 0$. If $AC(q)$ crosses $h(t^L(f_a^L + 1))$, the procedure returns $I_b = 1$, otherwise $I_b = 0$. The procedure returns, in addition, the value $\Delta(B)$ which is the total length of the coverage arrows, and a value Γ which is needed in computing the augmentation cost, as is shown below. The augmentation is due to the unit-arrows in $C^L \cup B$.

Calculation of the augmentation cost:

Case 1. $B = \emptyset$: Under this case the best Left solution is found similarly to the best right solution: The vehicle starts at a by possibly going right to $t^L(k)$ for some $k = 0, 1, \ldots, f^L$, there it makes a u-turn and goes back to a, and from there it follows the left basic route until its completion at b. Thereafter, the vehicle continues to the left to $h(t^L(k + 1))$, there it makes another u-turn in order to go back to b, where the tour ends. The cheapest resulting total augmentation cost under this case is $2 \min_{k=0}^{f^L} \{tp(A^L(k)) + hp(A^L(k + 1))\}$.

Case 2. $B \neq \emptyset$: Under this case $f_{ab}^L = 0$ and $C^L = C_a^L \cup C_b^L$.

Case 2.1 The vertices $t^L(f_a^L)$ and $h(t^L(f_a^L + 1))$ are not crossed by the same coverage arrow: The conditions in this case imply that $x[t^L(f_a^L)] < x[h(t^L(f_a^L + 1))]$, as otherwise there would either be a single coverage arrow that crosses both $t^L(f_a^L)$ and $h(t^L(f_a^L + 1))$, or $B = \emptyset$, contradicting our assumptions. If $I_a = 1$ then $AC(1)$ is the coverage arrow that crosses $t^L(f_a^L)$. In such a case let $\tilde{t}$ and $\tilde{h}$ be the tail and the head of $AC(1)$, implying that $\tilde{h} < t^L(f_a^L) < \tilde{t}$. If $I_a = 0$, then let $\tilde{t} = \tilde{h} = t^L(f_a^L)$. Similarly, if $I_b = 1$ then $AC(q)$ is the coverage arrow that crosses $h(t^L(f_a^L+1))$. In such a case, let $\hat{t}$ and $\hat{h}$ be the tail and the head of $AC(q)$, implying that $\hat{h} < h(t^L(f_a^L + 1)) < \hat{t}$. If $I_b = 0$, then let $\hat{t} = \hat{h} = h(t^L(f_a^L + 1))$. If $I_a I_b = 1$ then $q > 1$ and $\tilde{t} < \hat{h}$, (see Figure 5). The only augmentation whose cost is bounded from above by $2(x[b] - x[a])$ is to start driving right from a to $\tilde{t}$, make there a

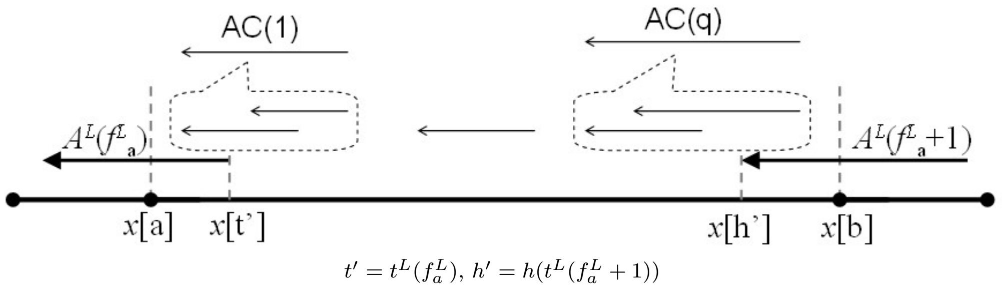


Figure 5: $t(f^L_a) < h(t^L(f^L_a + 1))$, *each of the two vertices is crossed by a different coverage arrow*

u-turn, go back to a, follow the left basic route while serving in loops the coverage arrows lying between $\tilde{t}$ and $\hat{h}$, complete the left basic route at b and then continue to the left to vertex $\hat{h}$, where it makes another u-turn in order to go back to b. The total augmentation cost is $2(tp(A^L(f^L_a)) + hp(A^L(f^L_a + 1)) + \Delta(B) - [(x[t^L(f^L_a)] - x[\tilde{h}]) + (x[\hat{t}] - x[h(t^L(f^L_a + 1))]]))]$.

Case 2.2 The two vertices $t^L(f^L_a)$ and $h(t^L(f^L_a + 1))$ are crossed by the same coverage arrow: Under this case any augmentation costs at least $2(x[b] - x[a])$ as all the intervals in between a and b are covered by left unit-arrows. In view of the removal of covered unit-arrows, it must hold that $q = 1$ and $AC(1)$ covers the f^B unit-arrows of B, and $h_1 < t^L(f^L_a) \leq h(t^L(f^L_a + 1)) < t_1$, where t_1 and h_1, respectively, are the tail and head of $AC(1)$ (see Figure 6). Consider first the case that the vehicle starts the tour by going right to $t^L(\ell)$, for some $\ell \neq f^L_a$ and $0 \leq \ell \leq f^L$, there it makes a u-turn in order to go back to a and start the left basic route. Then, after completing the left basic route at b, it continues to $h(t^L(\ell+1))$, where it makes a second u-turn in order to return to b. Note that such a route ensures the coverage of all the unit-arrows in B, as either before starting the left basic route ($\ell > f^L_a$) or after its completion ($\ell < f^L_a$), the vehicle traverses all the way from b to a. In overall there are f^l such options. More caution should be paid if the vehicle starts the tour by going right to $t^L(f^L_a)$: Note that here a replication of the form of the tour for $\ell \neq f^L_a$ may miss the service to some unit arrows in B. Indeed here one needs to consider the f^B unit-arrows of B, which are covered by $AC(1)$, as it may be beneficial continuing right from $t^L(f^L_a)$ to the tail of

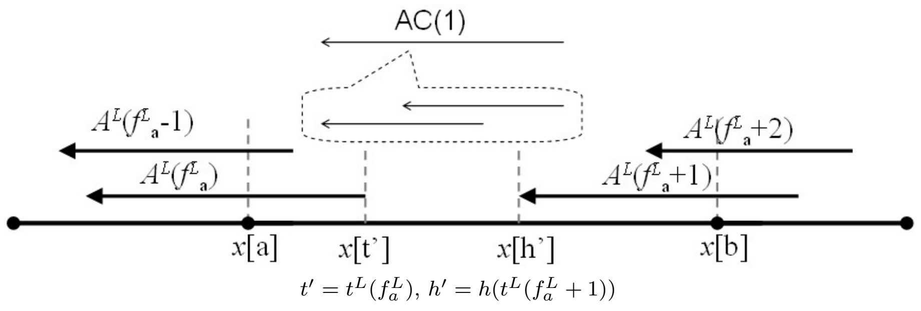


Figure 6: $t^L(f_a^L) < h(t^L(f_a^L + 1))$; *both are crossed by* $AC(1)$

the $k-$th unit arrow in array $\overline{B}$ making there a u-turn and then after completing the left basic route at b, continue left to the head of the $(k + 1)^{st}$ unit-arrow in $\overline{B}$, where another u-turn is made in order to return to b. Thus, starting by going right to $t^L(f_a^L)$ necessitates the need to compare $f^B + 1$ options regarding the additional right ride before making the first u-turn in order to start the left basic route. For this sake let $t^B(0) = t^L(f_a^L)$, and $t^B(f^B + 1) = t^L(f_a^L + 1)$. In overall, the best augmentation cost for the $f^L + f^B + 1$ options in this case, calculated by Procedure Coverage, is $2\min\{\min_{k=0,k\neq f_a^L}^{f^L}\{(tp(A^L(k)) + hp(A^L(k+1)))\}; (tp(A^L(f_a^L)) + hp(A^L(f_a^L + 1)) + \Gamma)\}$, where $\Gamma = \min_{j=0}^{f^B}\{x[t^B(j)] - (x[t^L(f_a^L)]) + (x[h(t^L(f_a^L + 1))]) - x[h(t^B(j + 1))]\}$.

The Algorithm: Left Basic Route returns the best left solution V^L. The complexity for computing V^L is $O(f^L)$. Based on the results of the last two subsections we conclude:

Theorem 1. *The complexity of the algorithm for computing the optimal solution for SP on a line with* n *vertices is* $O(n)$.

3.3 Two special cases

Now we consider two special cases: In the first, $a = b$, meaning that the initial and terminal points coincide. The second is the problem where the initial and terminal points are at the two extreme end-points of the line. For both special cases the proposed algorithm boils down to simpler versions, though the complexity is obviously the same.

3.3.1 $a = b$

In this case the crossing arrows are the right and left unit-arrows that cross vertex a. Both the left and right basic routes are of length of 2. As $B = \emptyset$, the best left solution is calculated similarly to the best right solution.

3.3.2 $x[a] = 0$ and $x[b] = 1$

In this case, any right solution requires a traversal of the whole line three times, i.e., $V^R = 3$, which is the worst possible, see (2). In any left solution, the vehicle traverses the line from its left end-point to its right end-point, but each time it reaches the tail of a coverage arrow, it makes there a u-turn and rides back up to the head of the coverage arrow, where it makes again a u-turn in order to continue to the right end-point. The cost of the optimal solution is therefore $V^*_{Line} = V^L = 1 + 2\Delta(B) \leq 3$.

4 The SP on a Circle

Recall that here we assume that the depot is located at vertex 1, and it serves as both the initial and terminal vertex ($a = b = 1$). Let V^*_{Circle} be the optimal solution of the SP on a circle, and $l(i) = x[i+1] - x[i]$, for $i = 1, ..., n$, is the length of the i-th interval. V^*_{Circle} is bounded from below by the minimum between the cost of a full traversal of the circle, and the cost of covering the whole circle twice except of one interval:

$$V^*_{Circle} \geq min\{1; \min\{2(1 - l(i)) : i = 1, ..., n\}\}. \tag{3}$$

In order to obtain an upper bound we consider the cheapest among two feasible tours, which are obtained by going clockwise (hereinafter c.w.) from vertex 1 to n and back, or by going counter-clockwise (hereinafter c.c.w.) from vertex 1 to 2 and back. Thus,

$$V^*_{Circle} \leq 2(1 - max\{l(1), l(n)\}). \tag{4}$$

Both lower and upper bounds in (3) and (4) can be shown to be tight.

Definition 4.1. *A solution to the SP on a circle is said to contain a complete encirclement (hereinafter c.e.) if the vehicle traverses all n intervals of the circle in the same direction at least once.*

The next property is straightforward:

Property 3. *Any feasible solution either contains a c.e. (c.w. or c.c.w.), or it leaves an uncovered interval, while all the others are covered by at least one ride to each direction.*

Let V^{UC} be the optimal solution with an uncovered interval, and V^{CE} be the optimal solution with a c.e. We calculate these two values in the next two subsections. $V^*_{Circle} = \min\{V^{UC}, V^{CE}\}$.

4.1 V^{UC}

Let $V^{UC}(i)$,$i = 1, \dots, n$, be the best solution that does not traverse interval $(i, i+1)$ in neither direction. As observed above , for $i = 1$ and $i = n$, $V^{UC}(i) = 2(1 - l(i))$, see the explanation of (4). For $1 < i < n$, computing $V^{UC}(i)$ boils down to solving an appropriate *uncapacitated SP on a line*, where the initial and terminal points coincide at vertex 1, the left end-point of the line is vertex $i+1$, and the right end-point is vertex i, implying that the length of the line is $1 - l(i)$. The problem can be solved by invoking the algorithm for the line proposed in Section 3 in its simplified version for $a = b$, see Subsection 3.3.1. In view of (4), a further simplification is obtained by noting that there is no need to calculate $V^{UC}(i)$ for $1 < i < n$ if $l(i) \leq \max\{l(1), l(n)\}$. *Algorithm SP-Circle(UC)* returns $V^{UC} = \min\{V^{UC}(i) | i = 1, \dots, n\}$, in complexity $O(n^2)$.

4.2 V^{CE}

The problem is solved twice, once for c.w. c.e., whose value is denoted by $V^{CE(c.w.)}$ and once for a c.c.w. c.e., whose value is denoted by $V^{CE(c.c.w.)}$. $V^{CE} = \min\{V^{CE(c.w.)}, V^{CE(c.c.w.)}\}$. Clearly, if a c.e. is feasible then $V^{CE} = 1$. Otherwise, it is necessary to augment it. W.l.o.g., we describe the algorithm for $V^{CE(c.w.)}$. For simplicity, if it is not said otherwise, we use c.e. for a c.w. c.e. As we are going to see, it is sufficient to look at augmentations of the c.e. that are associated with a pair of vertices (p, f), such that $1 \leq f < p \leq n+1$, and consist of three, possibly empty, disjoint subsets of vertices:

1. $PL1(p) = \{\ell : p \leq \ell \leq n\}$;
2. $PL2(f) = \{\ell : 1 \leq \ell \leq f\}$;
3. $R(p, f) = \{\ell : f < \ell < p\}$.

The tour associated with this partition starts by driving in a c.c.w. direction to p while picking-up the loads supplied by the vertices in $PL1(p)$; at vertex p the vehicle makes a u-turn and returns to the depot. This initial segment of the tour is called the *first pre-loading segment.* If $p = n{+}1$, then $PL1(n{+}1) = \emptyset$ and the first pre-loading option is not used. After completion of the first pre-loading segment, the vehicle goes c.w. from the depot to vertex f while picking-up the supplies of the vertices in $PL2(f)$. This second segment of the tour is called the *second pre-loading segment.* Note that along the two pre-loading segments no object is unloaded from the vehicle, as the vertices in $PL1 \cup PL2$ will be visited again later at the end of the tour. By definition, $f \geq 1$, which means that the load of vertex 1 is picked-up at the beginning of the pre-loading segments, but its demand is satisfied later when the vehicle returns to the depot after picking-up all objects. The remaining vertices in $U \backslash (PL1(p) \cup PL2(f)) = R(p, f)$ are served as follows: From vertex f the vehicle goes c.w. and whenever it reaches a vertex for the first time, it loads its supply, and if possible it also unloads its demand. Vertices in $R(p, f)$ whose demand could not been served at the first visit, are served in loops: Suppose that $j \in R(p, f)$ is the first such vertex, i.e., when j is first visited, the demands of all vertices in $\{f + 1, \ldots, j - 1\}$ have already been satisfied, but object β_j is not available on the vehicle. The vehicle then continues in a c.w. direction, until the first time it reaches a vertex k, $p > k > j > f$, where the current accumulated cargo on the vehicle together with object α_k, is sufficient to satisfy the demands of all the yet unserved vertices in $\{j, j + 1 \ldots, k\}$. When reaching k, the vehicle makes there a u-turn and drives back to j while satisfying the demands of the yet unserved vertices. At j the vehicle makes again a u-turn in order to go back to k. The ride from k to j in order to satisfy the yet unserved demands, and back to k, is called a *loop.* The vehicle then continues the c.w. ride from k to $p - 1$ while making loops if necessary. The set $LOOP(p, f) \subset R(p, f)$ consists of all the vertices covered by such loops. In particular, $\{j, j{+}1, \ldots, k\} \subset LOOP(p, f)$. By definition, the sets $PL1(p)$,$PL2(p)$, and $LOOP(p, f)$ are disjoint. When reaching p for the second time, the vehicle has completed to pick-up all the objects, and by riding c.w. to f it can satisfy the demands of the vertices in $PL1 \cup PL2$. At vertex f the vehicle makes a u-turn and it returns to the depot. Figure 7 demonstrates a c.w. c.e. and the possible components of an augmentation.

Thus, for given p and f, the problem reduces to solving the SP on a line with $n + f + (n - p + 1)$ vertices, denoted by $L(p, f)$: The left part of $L(p, f)$

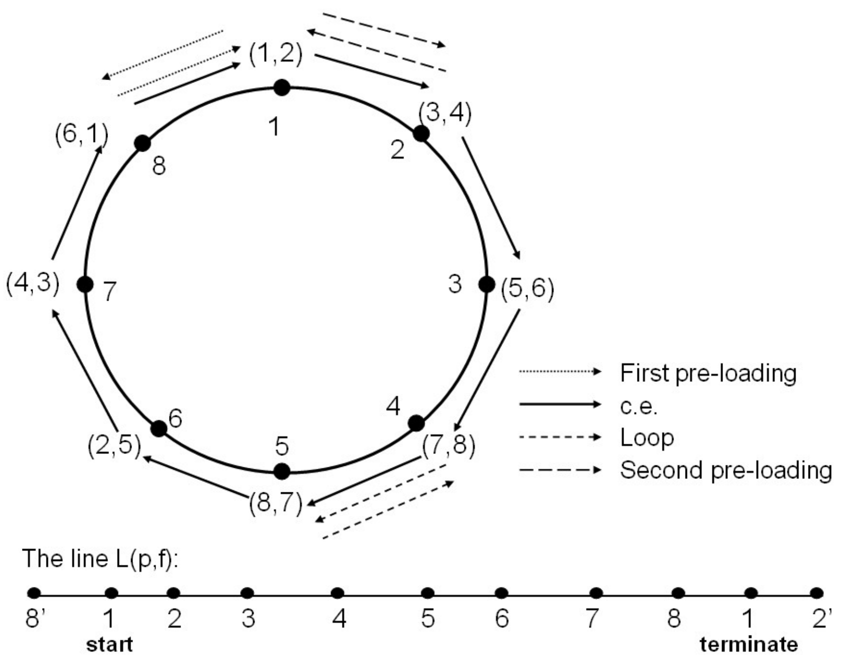


Figure 7: $(p, f) = (8, 2)$, $PL1(8) = \{8\}$, $PL2(2) = \{1, 2\}$, $R(8, 2) = \{3, 4, 5, 6, 7\}$, $LOOP(8, 2) = \{4, 5\}$.

is associated with the c.w. circular arc from vertex p to 1, attached to it is the middle part of $L(p, f)$ that consists of a c.e. of the circle, and thereafter attached to it is the last part which is associated with a c.w. circular arc from vertex 1 to f. Number the vertices on $L(p, f)$ from left to right by $p', \ldots, n'$,1$, \ldots, f$,$f + 1, \ldots, p - 1$, $p, \ldots, n, n + 1$,$2', 3', \ldots, f'$. Let $U(p, f)$ denote the set of vertices of $L(p, f)$. The initial and terminal vertices are $a = 1$ and $b = n + 1$, respectively. The supply of $q' \in \{p', \ldots, n'\}$ and of $q \in \{1, \ldots, f\}$ is α_q, where its demand is the null object. The supply and demand of $q \in \{f + 1, \ldots, p - 1\}$ are α_q and β_q, respectively. The supply of $q \in \{p, \ldots, n + 1\}$ and of $q' \in \{2', \ldots, f'\}$ is the null object, and its demand is β_q. Thus, in the new problem the total supply equals the total demand, for each object in S_{-0}, and it is exactly the same as in the original problem on the circle. See Figure 8.

Observation 1. *The SP on line $L(p, f)$ is equivalent to the problem of finding the best SP tour on a circle with a c.w. c.e., and a pre-loading component that consists of the sets of vertices $PL1(p) \cup PL2(f)$.*

As shown in Section 3, the complexity of the algorithm for solving the SP on a line is linear in the number of vertices, and as in the worst case there are

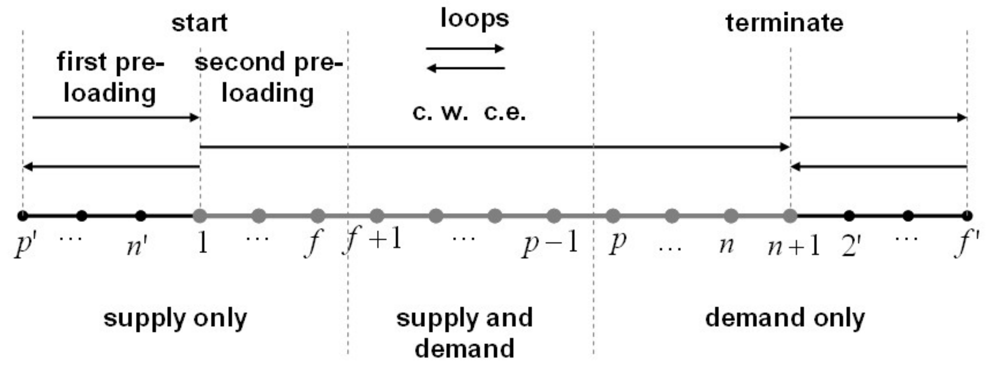


Figure 8: *Line*(p, f)

$O(n^2)$ different pairs of (p, f) that need to be verified, a direct application of the above described algorithm requires a complexity of $O(n^3)$, even if we use the special structure of $L(p, f)$. The special structure of $L(p, f)$ is due to the fact that vertices $\{p', \ldots, n'\} \cup \{1, \ldots, f\}$ have no demand, and vertices $\{p, \ldots, n + 1\} \cup \{2', \ldots, f'\}$ have no supply, implying that right solutions cost more than left solutions. Therefore, it suffices to focus on left solutions only. Furthermore, the algorithm that generates left solutions boils down to a simple version as the structure of $L(p, f)$ is such that all its left unit-arrows are in the set B, denoted here by $B(p, f)$. In particular, it means that $L(p, f)$ has no left crossing unit arrows. All unit-arrows in $B(p, f)$ and the corresponding coverage arrows have their tail at t and their head at h such that $f < h < t < p$. The best left solution starts at vertex $a = 1$, goes to the left end-point of the line, namely vertex p', makes there a u-turn and goes right, while making loops on the coverage arrows in between vertices $f+1$ and $p-1$; thereafter the vehicle continues to the right end-point of the line, namely to vertex f', where it makes a u-turn to go back to the terminal point $b = n+1$. Thus, $V^{CE(c.w.)} = \min_{1 \le f < p \le n+1}\{1+2(x[f]+(1-x[p])+\Delta(B(p, f)))\}$, where $\Delta(B(p, f))$ is the corresponding length of the coverage arrows.

In view of the upper bound in (4), it is sufficient to restrict our search to pairs of (p, f) for which $x[f]+\Delta(B(p, f))+(1-x[p]) < 0.5-\max\{\ell(1), \ell(n)\}$, which implies that $x[p] - x[f] > 0.5 + \max\{\ell(1), \ell(n)\}$. Let $\underline{p}$ and $\overline{f}$ be defined such that $x[\underline{p} - 1] \le 0.5 + \max\{\ell(1), \ell(n)\} < x[\underline{p}]$, and $x[\overline{f}] < 0.5 - \max\{\ell(1), \ell(n)\} \le x[\overline{f} + 1]$. In addition, let $\overline{f}(p)$ for $p \ge \underline{p}$, be the maximum index f such that $x[p] - x[f] > 0.5 + \max\{\ell(1), \ell(n)\}$. Clearly, $\overline{f}(p) \le \overline{f}(p + 1) \le \overline{f}(n + 1) = \overline{f}$. Thus, for any $p \ge \underline{p}$, we need to consider $L(p, f)$ for $1 \le f \le \overline{f}(p)$. The calculation of $\underline{p}$, $\overline{f}$, and the sequence $\overline{f}(p)$ for $p \ge \underline{p}$, takes $O(n\log(n))$ time. Unfortunately, all these insights do not reduce the magnitude of the complexity, which remains $O(n^3)$. In the next subsection we show that the problem has further properties that make it relatively easy to solve it iteratively for all possible pairs of (p, f) without invoking each time the algorithm for a general line. The new method enables us to find V^{CE} in complexity of $O(n^2)$.

4.2.1 Improvement: computing V^{CE} in $O(n^2)$

Below we describe how to calculate $\Delta(B(p, f))$ for all possible pairs (p, f) in $O(n^2)$ time only. The reduced complexity is achieved by updating both $\Delta(B(p, f + 1))$ and $\Delta(B(p - 1, f))$ from $\Delta(B(p, f))$ in constant time on average. This saving is based on the following claim:

Claim 1. *The set $LOOP(p, f)$ as well as $\Delta(B(p, f))$ are (i) non-decreasing in p for any given f; and (ii) non-increasing in f for any given p. In particular, $\Delta(B(p, f)) \leq \Delta(B(n + 1, 1))$.*

Proof. As $LOOP(p, f) \subset R(p, f) = \{f + 1, \ldots, p - 1\}$, the smaller is p and the larger is f $(f < p)$, the load on the vehicle after the pre-loading segments is larger, and therefore more vertices in $R(p, f)$ get served at the first time they are visited. As a result the set of vertices served by loops is shrunk. □

We present now the recursive method that enables the reduction of the complexity. Initialization Procedure $L(n + 1, 1)$-I starts by initializing the data for line $L(n + 1, 1)$. For $1 \leq i \leq n$, and $j \in S_{-0}$ let:

- $W = (\omega(i, j))$ is a matrix, where $\omega(i, j)$ is the number of left j-unit-arrows that cover interval $(i, i + 1)$.
- $S(i) = \sum_{j\in S_{-0}} \omega(i, j)$ is the number of left unit-arrows that cover the interval;
- Indicator $I(i) = 1$ if $S(i) > 0$, and 0 otherwise. Thus, $I(i) = 1$ implies that interval $(i, i + 1)$ is covered by a coverage arrow.
- Indicator $Y(j)$ assumes the value 1 if there are no any left j-unit-arrows in $B(n + 1, 1)$, and 0 otherwise.

Note that $Y(j) = 1$ implies that along the c.e. all vertices demanding object j get the object at the first visit of the vehicle there. It is easy to see that in such a case $Y(j) = 1$ for any $L(p, f)$. The matrix W, the indicators

$I(i)$, the values $S(i)$ for $i = 1, \dots, n$, and $Y(j)$ for $j \in S_0$ that are found for $L(n+1,1)$ by Initialization Procedure $L(n+1,1)$-I, need to be updated for each possible pair (p, f) and its respective line $L(p, f)$. Updating the matrix W for $L(p, f)$ from $L(p, f-1)$ if $f > 1$, or from $L(p-1, f)$ if $f = 1$, requires $O(n)$ operations, as the number of intervals between $f+1$ and $p-1$ is $O(n)$, meaning that such a procedure is not going to help in reducing the complexity.

In order to present an $O(n^2)$ algorithm we need a more efficient data structure. Considering $L(n+1,1)$, note that for each $j \in S_{-0}$, $\omega(1, j) = 0$, as the demand of vertex 1 is satisfied only at the end of the c.e. In general, the values $\omega(i, j)$ for $i = 2, \dots, n-1$ satisfy the following properties:

- If $j \notin \{\alpha_{i+1}, \beta_{i+1}\}$ then $\omega(i+1, j) = \omega(i, j)$.
- If $j = \alpha_{i+1}$, then $\omega(i+1, j) = (\omega(i, j) - 1)^+$, where $x^+ = \max\{x, 0\}$.
- If $j = \beta_{i+1}$ and $\omega(i, j) > 0$, then $\omega(i+1, j) = \omega(i, j) + 1$.
- If $j = \beta_{i+1}$, and $\omega(i, j) = 0$, then $\omega(i+1, j) \in \{0, 1\}$, depending on the current load on the vehicle.

Thus, for a given j, $\omega(i, j)$ as a function of i, is a step function that assumes non-negative integer values, starting at 0 for $i = 1$, and changing in steps of a size of at most 1 unit. These properties continue to hold for any $L(p, f)$.

Next, the proposed data structure for $L(p, f)$ is described. For illustration purposes, assume that matrix W, and the indicators $I(i)$ and $Y(j)$ for $i = 1, \dots, n$ and $j \in S_{-0}$, which are associated with $L(p, f)$, are given. (In practice, these values are calculated by Initialization Procedure-I only for $L(n+1, 1)$, and these are used as input by initialization Procedure $L(n+1, 1)$-II to produce the proposed data structure). Suppose that in the solution for $L(p, f)$, for some $k > f$, $\omega(k-1, j) = 0$, $\omega(k, j) = 1$, $\omega(i, j) \geq 1$ for $i = k+1, \dots, \ell-2 < p-2$, $\omega(\ell-1, j) = 1$, and $\omega(\ell, j) = 0$. Then, $[k, \ell]$ is said to be a *1-block of j*, vertex k is said to be the *starting vertex of the block*, and vertex ℓ is said to be the *ending vertex of the block.* This means that in the SP solution of $L(p, f)$, each of the intervals inbetween vertices k and ℓ is covered by at least one left j-unit-arrow. Note that object j can be associated by a number of non-overlapping 1-blocks. In general, define:

Definition 4.2. *An $\underline{h\text{-block of } j}$ is a maximal set of consecutive intervals where each is covered by at least h left j-unit-arrows. h is said to be the level of an h-block of object j.*

Clearly, the blocks are nested in the sense that any h-block of j is also an h'-block of j for $h' \leq h$. Storing and updating blocks instead of matrix W consume less memory and time. More precisely, we show that the blocks require $O(n)$ space, where the matrix - $O(n^2)$. For this sake, note that the number of blocks for any line $L(p, f)$ is bounded from above by n and that the number of blocks is non-increasing as p gets smaller and f gets larger. In order to see this note that given a certain h-block $[k, \ell]$, $h \geq 1$, of object j, $j \in S_{-0}$, then at the interval $(k, k+1)$ the demand for j has increased by 1, i.e., $\beta_k = j$, and at the interval $(\ell, \ell+1)$ the demand for j has decreased by 1, i.e., $\alpha_\ell = j$. As each vertex is associated with at most a single unit of supply and a single unit of demand, then each vertex may be the initial (terminal) vertex of at most one block, implying that the total number of blocks is bounded by the number of vertices in between $f+1$ and $p-1$, namely $p - f - 1 < n$. Moreover, the number of levels for each object is bounded by n_j, where n_j is the number of units of object j in the problem, and $\sum_{j \in S_{-0}} n_j \leq n$.

The information about the left unit-arrows associated with $L(p, f)$, and, in particular, the output of Initialization Procedure $L(n+1, 1)$-I, is stored by using pointers rather than a matrix. Next we describe the pointers:

1. For each $j \in S_{-0}$, satisfying $Y(j) = 0$, i.e., an object that is associated with at least one left j- unit-arrow, define pointer $first(j)$ to point to the minimum index vertex k, $f < k < p$, with $\beta_k = j$, which is served by a loop in $L(p, f)$. This means that k is the starting vertex of the first 1-block of j.

2. For any k which is the starting vertex of an h-block of j, define the following pointers:

   (a) Pointer $end(k)$ points to the ending vertex of the block that stars at k. This means that for some $h \geq 1$, $\omega(i, j) \geq h$ for $k \leq i < end(k)$ and $\omega(end(k), j) = h - 1$.

   (b) Pointer $nxtblck(k)$ points to the starting vertex of the next h-block of j, if such one exists, i.e., $\omega(i, j) \leq h-1$ for $i = end(k), \ldots, nxtblck(k)-1$ and $\omega(nxtblck(k), j) = h$. If k is the starting vertex of the last h-block of j, then let $nxtblck(k) = 0$.

   (c) Pointer $nxtlvl(k)$ points to the starting vertex of the next $(h+1^{st})$-block of j, if such one exists. For example, $nxtlvl(first(j))$ points to the starting vertex of the first 2-block of j, $nxtlvl^2(first(j))$ points to the starting vertex of the first 3-block of j, etc. If h is the highest block level for object j, and k is the starting vertex an h-block of j, then let $nxtlvl(k) = n + 1$.

(d) Let pointer $prvlvl(\ell) = k$ if and only if $nxtlvl(k) = \ell$.

Initially, $first(j)$ for $j \in S_{-0}$, $nxtlvl(i), prvlvl(i), end(i), nxtblck(i)$ are set to 0, for $i = 1, \dots, n$. Initialization Procedure $L(n+1,1)$-II uses as input matrix W found by Initialization Procedure $L(n+1,1)$-I, and scans it for setting the blocks and the pointers described above. More specifically, the second Initialization Procedure uses stacks of maximum size bounded by n, in order to facilitate the initialization of the pointers. Let SK be a stack of current size ns. The following operations on stacks are common:

1. Function $TOP[SK]$ returns the last element inserted to the stack, without removing it from the stack.
2. Function $POP[SK]$ removes the last element from the stack.
3. Function $PUSH[SK, e]$ inserts a new element e as the last element of the stack.

For each column $j \in S_{-0}$ with $Y(j) = 0$, the procedure opens an empty stack SK. After scanning the first i elements of column j, the stack contains at its h position, if it is not empty, the starting vertex of the h-block of j, where the ending vertex of this block has not yet been scanned, i.e., its ending vertex is in $\{i+1, \dots, n\}$. A block whose starting vertex has been scanned but its ending vertex has not been scanned is said to be an *open block*. In addition, we use an auxiliary function $g : \{1, \dots, n\} \to \{0, 1, \dots, n\}$, where $g(h)$ indicates the starting vertex of the block of j at level h that has been opened. By construction, the update of $g(h)$ is upwards. $g(h) = 0$ means that no block of j at level h has been opened. Initially, set $g(h) = 0$, for $h = 1, \dots, n$.

Initialization Procedure $L(n+1,1)$-II performs the following steps in order to set the pointers for object j: $first(j) = \min\{i : \omega(i,j) > 0\}$, and $nxtlvl(first(j)) = n+1$, where $nxtlvl(i) = n+1$ means that i is currently at the highest indexed open block for object j. Add $first(j)$ to the stack by applying $PUSH[SK, first(j)]$, and increase ns by 1. Set also $g(1) = first(j)$. For $i = first(j) + 1, \dots, n$ do the following:

1. If $\omega(i,j) \neq \omega(i-1,j)$ and $ns \neq 0$, then let $\ell = TOP[SK]$.
2. If $\omega(i,j) > \omega(i-1,j)$ do the following:

- Increase ns by 1;
- If $g(ns) > 0$ then let $nxtblck(g(ns)) = i$;
- If $g(ns) = 0$ then let $nxtlvl(\ell) = i$; $prvlvl(i) = \ell$; $nxtlvl(i) = n+1$ and $prvlvl(n+1) = i$;
- Insert i to the end of the stack by applying $PUSH[SK, i]$;
- Let $g(ns) = i$;

3. If $\omega(i, j) < \omega(i-1, j)$ do the following:
   - Reduce ns by 1;
   - Remove ℓ from the stack by applying $POP[SK]$;
   - Let $end(\ell) = i$;
   - If $nxtlvl(\ell) = n+1$ then let $nxtlvl(prvlvl(\ell)) = n+1$ and $prvlvl(n+1) = prvlvl(\ell)$;

For any $L(p, f)$, Δ denotes the length of the respective coverage arrows. Initially, for $L(n+1, 1)$, $\Delta := \Delta(B(n+1, 1))$ and the cost $V := 1 + 2\Delta$ are calculated by Initialization Procedure $L(n+1, 1)$-I. Let (p^*, f^*) be the currently optimal (p, f)-pair and V is the currently optimal cost. Initially, set $(p^*, f^*) = (n+1, 1)$ and $V = 1 + 2\Delta(B(n+1, 1))$. Next the recursive update of the data is described:

- Update of data of $L(p, 1)$ based on $L(p+1, 1)$ for $p = n, n-1, \dots, \underline{p}$: Let $\Delta \leftarrow \Delta(B(p+1, 1))$. If $\alpha_p = 0$ or $Y(\alpha_p) = 1$, then the data for $L(p, 1)$ is the same as for $L(p+1, 1)$. Otherwise, the only data that we need to update is with respect to object α_p. A comparison between the load of the vehicle when it starts the tour at vertex 2, after completing the pre-loading components, reveals that in $L(p, 1)$ the vehicle contains the extra object α_p relative to its load in $L(p+1, 1)$. Therefore, the update of the $S(i)$ values for $L(p, 1)$ is by reducing their respective values in $L(p+1, 1)$ by 1, for each interval $(i, i+1)$, $2 \leq i < p-1$, which is covered by a 1-block of α_p. For all $i \in \{2, \dots, p-2\}$, for which $S(i)$ has dropped to 0, set $I(i) := 0$ and $\Delta \leftarrow \Delta - l_i$, as the the coverage arrows in $L(p, 1)$ do not cover interval $(i, i+1)$ that was covered in $L(p+1, 1)$. Set $\Delta(B(p, 1)) \leftarrow \Delta$. If $1 + 2(\Delta + 1 - x[p]) < V$, then let $(p^*, f^*) \leftarrow (p, 1)$, and $V \leftarrow 1 + 2(\Delta + 1 - x[p])$. In addition, if $L(p+1, 1)$ does not contain any 2-blocks of object α_p, i.e., in $L(p+1, 1)$ $nxtlvl(first(\alpha_p)) = n+1$, then delete the 1-blocks of α_p and set $Y(\alpha_p) = 1$ and $first(\alpha_p) = 0$.

Otherwise, update the data for $L(p, 1)$ from the data of $L(p + 1, 1)$ by deleting the 1-blocks of α_p, which are not 2-blocks of α_p, and making the h-blocks of α_p to be the $(h-1)$-blocks of α_p, for $h \geq 2$. This is done by setting $first(\alpha_p) \leftarrow nxtlvl(first(\alpha_p))$. Up to here, the amount of data that has been stored for $L(p, 1)$, $p \in \{\underline{p}, \ldots, n + 1\}$, is $O(n^2)$.

- Update of data of $L(p, f)$ based on $L(p, f - 1)$ for $p = n + 1, n, \ldots, \underline{p}$ and $f = 2, \ldots, \overline{f}(p)$: The procedure is similar to the one described above. Let $\Delta \leftarrow \Delta(B(p, f - 1))$. If $\alpha_f = 0$ or $Y(\alpha_f) = 1$, then the data for $L(p, f)$ is the same as for $L(p, f - 1)$. Otherwise, the data with respect to object α_f need to be updated. A comparison between the load of the vehicle when it starts the tour after completing the pre-loading components reveals that in $L(p, f)$ the vehicle contains the additional object α_f relative to its load in $L(p, f - 1)$. Therefore, the update of the $S(i)$ values for $L(p, f)$ is by reducing their values in $L(p, f - 1)$ by 1, for each interval $(i, i + 1)$, $f < i < p - 1$, which is covered by a 1-block of α_f. For all $i \in \{f + 1, \ldots, p - 2\}$, for which $S(i)$ has dropped to 0, set $I(i) := 0$ and $\Delta \leftarrow \Delta - l_i$, as the coverage arrows in $L(p, f)$ do not cover interval $(i, i + 1)$ that was covered in $L(p, f - 1)$. Set $\Delta(B(p, f)) \leftarrow \Delta$. If $1 + 2(\Delta + x[f] + 1 - x[p]) < V$, then let $(p^*, f^*) \leftarrow (p, f)$, and $V \leftarrow 1 + 2(\Delta + x[f] + 1 - x[p])$. In addition. if $L(p, f - 1)$ does not contain any 2-blocks of object α_f, i.e., if in $L(p, f - 1)$ $nxtlvl(first(\alpha_f)) = n + 1$, then update the data for $L(p, f)$ by deleting the 1-blocks of α_f and setting $Y(\alpha_f) = 1$ and $first(\alpha_f) = 0$. Otherwise, update the data for $L(p, f)$ from the data of $L(p, f - 1)$ by deleting the 1-blocks of α_f, which are not 2-blocks of α_f, and making the h-blocks of α_f to be the $(h-1)$-blocks of α_f, for $h \geq 2$. This is done by setting $first(\alpha_f) \leftarrow nxtlvl(first(\alpha_f))$. At the end of this process, the algorithm returns the optimal pair (p^*, f^*), and the optimal c.w. c.e. solution $V^{CE(c.w.)}(p^*, f^*) = 1 + 2(\Delta(B(p^*, f^*)) + x[f^*] + 1 - x[p^*])$.

The only concern regarding the complexity of the proposed algorithm is with respect to the number of updates of the $S(i)$ values for $i = 1, \ldots, n$. The total number of updates of these values is at most $\sum_{i=1}^{n} S(i)$, because the values are integer and at each update one of them is reduced by one unit. The $S(i)$ values are initially determined by Initialization Procedure $L(n+1, 1)$-I. Note that at its maximum, which is also its initial value, $S(i) \leq n - i$, as each of the $n - i$ vertices in $\{i + 1, \ldots, n\}$ can be the tail of no more than one left unit-arrow that covers interval $(i, i + 1)$. Thus, $\sum_{i=1}^{n} S(i) \leq \sum_{i=1}^{n} (n - i) = \sum_{i=1}^{n-1} i = 0.5n(n - 1)$, meaning that at most $O(n^2)$ such updates are needed.

Therefore, the complexity of Algorithm $SP-Circle(CE)(c.w.)$ is $O(n^2)$. In an analogous way also $V^{CE(c.c.w.)}$ is calculated, and the best of the two is returned as V^{CE}.

5 Concluding Remarks

The uncapacitated SP on general graphs, including the same problem on a tree de Paepe at al. [7], is known to be NP-hard, see the Introduction. In this paper, we investigate the problem on two simple graphs, line and circle. In both cases we propose algorithms of low complexity to solve them to optimality. For the line, the algorithm is linear in the number of vertices, the best complexity that can be expected. For the circle, the best algorithm we could generate has a complexity which is the square of the number of vertices.

Similarly to the uncapacitated SP, also the unit capacity SP is known to be polynomial on a line, see Anily, Gendreau and Laporte [2], and NP-hard on general graphs. In particular, the unit capacity SP on a tree, for both the preemptive and non-preemptive cases, see Anily, Gendreau and Laporte [3], and Frederickson and Guan [9], respectively, are NP-hard like the uncapacitated SP on a tree. However, it is still an open question whether the unit-capacity SP on a circle is polynomially solvable like its uncapacitated version on a circle, or whether there is a further inherent complexity in the problem that makes it NP-hard. We do hope that this research will encourage researchers in the field to investigate this open problem.

Acknowledgement

The research of the first author was partially funded by the Israeli Institute for Business Research.

References

[1] Anily, S. and R. Hassin (1992), The Swapping Problem. *Networks* 22: 419-433.

[2] Anily, S., M. Gendreau and G. Laporte (1999), The Swapping Problem on a Line. *SIAM Journal on Computing* 29: 327-335.

[3] Anily, S., M. Gendreau and G. Laporte (2006), The Preemptive Swapping Problem on a Tree.

[4] Anily, S. and G. Mosheiov (1994), The Traveling Salesman Problem with Delivery and Backhauls. *Operations Research Letters 16*: 11-18.

[5] Attalah, M.J. and S.R. Kosaraju (1988), Efficient Solutions to Some Transportation Problems with Applications to Minimizing Robot Arm Travel. *SIAM Journal on Computing* 17: 849-869.

[6] Charikar, M. and B. Raghavachari (1998), The Finite Capacity Dial-a-Ride Problem. *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*: 458-467.

[7] de Paepe, W., J.K. Lenstra, J. Sgall, R. Sitters, and L. Stougie (2004), Computer-Aided Complexity Classification of Dial-a-Ride Problems. *INFORMS Journal on Computing* 16: 120-132.

[8] Frederickson, G.N. and D.J. Guan (1992), Preemptive Ensemble Motion Planning on a Tree. *SIAM Journal on Computing* 22: 1130-1152.

[9] Frederickson, G.N. and D.J. Guan (1993), Non-preemptive Ensemble Motion Planning on a Tree. *Journal of Algorithms* 15: 29-60.

[10] Frederickson, G.N., M.S. Hecht and C.E. Kim (1978), Approximation Algorithms for some Routing Problems. *SIAM Journal on Computing* 7: 178-193.

[11] Guan, D.J. (1998), Routing a Vehicle of Capacity Greater than One. *Discrete Applied Mathematics* 81: 41-57.

[12] Kubo, M. and H. Kasugai (1990), Heuristic Algorithms for the Single Vehicle Dial-a- Ride Problem. *Journal of the Operations Research Society of Japan* 33: 354-365.

[13] Pfeffer, A. (2004), Uncapacitated Swapping Problems on a Line. M.Sc. Thesis (in Hebrew), The Recanati Graduate School of Business, Tel-Aviv University.

[14] Psaraftis, H. (1983), Analysis of an $O(n^2)$ Heuristic for the Single Vehicle Many-to-Many Euclidean Dial-a-Ride Problem. *Transportation Research* 17B: 133-145.

6 Appendix: The algorithms

This Appendix contains the algorithms.

Subsection 3.1

```
Procedure $\overline{C^R}$
1:set $f^R = 0$, $\overline{C^R}(\ell) = (0,0)$ for $\ell = 1$ to $n$, $k = 1$, $i = a - 1$.
2:    while $i > 0$ do begin:
3:        if $\overline{UA}^R(i) > b$ then $f^R \leftarrow f^R + 1$, $\overline{C^R}(k) = (i, \overline{UA}^R(i))$, $k \leftarrow k + 1$
4:        $i \leftarrow i - 1$.
5:    endwhile.
6: return $f^R$; $\overline{C^R}$.
```

```
Algorithm: Right Basic Route($\overline{C^R}$;$f^R$;$tp(A^R(k))$;$hp(A^R(k))$;$k = 1, \ldots, f^R$)
1: if $f^R = 0$ then do begin
2:   $V^R = 0$
3:otherwise
4:   set $V^R = hp(A^R(1))$
5:   for $k = 1$ to $f^R - 1$ do begin
6:       if $tp(A^R(k)) + hp(A^R(k+1)) < V^R$ then $V^R = tp(A^R(k)) + hp(A^R(k+1))$.
7:   endfor
8:   if $tp(A^R(f^R)) < V^R$ then $V^R = tp(A^R(f^R))$.
9:endif
10: return $V^R \leftarrow (2 + x[b] - x[a]) + 2V^R$.
```

Subsection 3.2

```
Procedure $\overline{C^L}$ and $\overline{B}$
1: set: $f^L = 0$, $f_a^L = 0$, $f_b^L = 0$, $f_{ab}^L = 0$, $f^B = 0$. $k^L = 1$, $k^B = 1$, $i = a + 1$.
   for $\ell = 1$ to $n$, $\overline{C^L}(\ell) = (0,0)$ and $= \overline{B}(\ell) = (0,0)$.
2: while $i \leq n$ if $0 < \overline{UA}^L(i) < a$ do begin:
3:     $f^L \leftarrow f^L + 1$,
4:     if $i \leq b$ then $f_a^L \leftarrow f_a^L + 1$; otherwise $f_{ab}^L \leftarrow f_{ab}^L + 1$.
5:     $\overline{C^L}(k^L) = (i, \overline{UA}^L(i))$, $k^L \leftarrow k^L + 1$ and $i \leftarrow i + 1$.
6: endwhile.
7: while $i \leq b$ if $\overline{UA}^L(i) \geq a$ do begin:
8:     $f^B \leftarrow f^B + 1$, $\overline{B}(k^B) = (i, \overline{UA}^L(i))$, $k^B \leftarrow k^B + 1$ and $i \leftarrow i + 1$.
9: endwhile.
10: while $i \leq n$ and $\overline{UA}^L(i) < b$ do begin:
11:     $f^L \leftarrow f^L + 1$, $f_b^L \leftarrow f_b^L + 1$,
12:     $\overline{C^L}(k^L) = (i, \overline{UA}^L(i))$, $k^L \leftarrow k^L + 1$ and $i \leftarrow i + 1$.
13: endwhile.
14: return $f^L$,$f^B$, $f_a^L$, $f_b^L$, $f_{ab}^L$, $\overline{C^L}$; $\overline{B}$.
```

```
Procedure: Coverage($\overline{B}$; $f^B$; $f_a^L$; $f_b^L$; $t^L(f_a^L)$; $h(t^L(f_a^L + 1))$)
1: set $t^B(0) = t^L(f_a^L)$; $h(t^B(f^B + 1)) = h(t^L(f_a^L + 1))$; $\Gamma = 0$; $\Delta(B) = 0$; $I_a = 0$; $I_b = 0$;
2: for $k = 1$ to $f^B$ do begin
3:     $t^{AC}(k) = 0$, $h^{AC}(k) = 0$, $AC(k) = (0,0)$
4: endfor.
5: $q = 1$; $h' = h(t^B(1))$; $t' = t^B(f^B)$
6: if $x[h'] < x[t^L(f_a^L)]$ then $I_a \leftarrow I_a + 1$.
7: if $x[t'] > x[h(t^L(f_a^L + 1))]$ then $I_b \leftarrow I_b + 1$.
8: for $k = 2$ to $f^B + 1$ do begin
9:     if $x[h(t^B(k))] > x[t^B(k-1)]$ then do begin
10:         $t^{AC}(q) = t^B(k-1)$; $h^{AC}(q) = h'$; $AC(q) = (x[t^{AC}(q)], x[h^{AC}(q)])$,
11:         if $k \leq f^B$ then $h' = h(t^B(k))$;
12:         $q \leftarrow q + 1$;
13:     endif.
14:     $k \leftarrow k + 1$.
15: endfor.
16: $q \leftarrow q - 1$
17: $\Delta(B) = \sum_{k=1}^{q}(x[t^{AC}(k)] - x[h^{AC}(k)])$
18: if $q * I_a * I_b = 1$ then do begin
19:     $\Gamma = x[h(t^L(f_a^L + 1))]] - x[t^L(f_a^L)]$;
20:     $\delta = x[t^B(0)] - x[h(t^B(1))]$
21:     for $j = 1$ to $f^B$ do
22:         if $x[t^B(j)] - x[h(t^B(j+1))] < \delta$
          then $\delta = x[t^B(j)] - x[h(t^B(j+1))]$
23:     endfor.
```

```
24: $\Gamma \leftarrow \Gamma + \delta$;
25: endif
26: return for $k = 1$ to $q$   $t^{AC}(k)$; $h^{AC}(k)$; $AC(k) = (t^{AC}(k), h^{AC}(k))$
27: return $q$; $I_a$; $I_b$; $\Delta(B)$; $\Gamma$;
```

Algorithm: Left Basic Route($\overline{C^L}$; f^L; f_a^L; f_b^L; $\overline{B}$; f^B; $t^L(f_a^L)$; $h(t^L(f_a^L+1))$; $tp(A^L(k))$, $hp(A^L(k))$ for $k = 1$ to f^L)

```
1: set $V^L = 0$, $tp(A^L(0)) = 0$, $hp(A^L(f^L+1)) = 0$; $I_a = 0$; $I_b = 0$;
2: if $f^B = 0$ then do begin     (Case 1)
3:    if $f^L > 0$ then do begin
4:      $V^L = 2\{tp(A^L(0)) + hp(A^L(1))\}$
5:      for $k = 1$ to $f^L$ do begin
6:          if $2\{tp(A^L(k)) + hp(A^L(k+1))\} < V^L$ then
7:            $V^L = 2\{tp(A^L(k)) + hp(A^L(k+1))\}$
8:      endfor.
9: endif
10:else do begin      (Case 2)
11:    run Procedure:Coverage($\overline{B}$; $f^B$; $f_a^L$; $f_b^L$; $t^L(f_a^L)$; $h(t^L(f_a^L+1))$)
12:    output $q$; $I_a$; $I_b$; $\Delta(B)$; $\Gamma$; $t^{AC}(k)$ and $h^{AC}(k)$ for $k = 1, \ldots, q$
13:    if $q * I_a * I_b \neq 1$ then do begin      (Case 2.1)
14:        $V^L = 2\Big\{ tp(A^L(f_a^L)) + hp(A^L(f_a^L+1)) + \Delta(B) -$
           $I_a\Big( x[t^L(f_a^L)] - x[h^{AC}(1)]\Big) - I_b\Big( x[t^{AC}(q)] - x[h(t^L(f_a^L+1))]\Big)\Big\}.$
15:    endif
16:    else do begin $V^L = 2\{tp(A^L(f_a^L)) + hp(A^L(f_a^L+1)) + \Gamma\}$   (Case 2.2)
17:        for $k = 1$ to $f^L$ do begin
18:          if $k \neq f_a^L$ then
19:              if $2\{tp(A^L(k)) + hp(A^L(k+1))\} < V^L$ then
20:              $V^L = 2\{tp(A^L(k)) + hp(A^L(k+1))\}$
21:        endfor.
22:     endif
23:    $V^L \leftarrow (2 + x[a] - x[b]) + V^L$
24: return $V^L$.
```

Subsection 4.1

Algorithm SP-Circle(UC)

```
1: input: for $i = 1$ to $n$, $x[i]$; $x[n+1] = 1$;
2: for $i = 1, \ldots, n$ do begin $l(i) = x[i+1] - x[i]$; $x'[i] = 0$; endfor
3: if $l(1) > l(n)$ then $I = 1$ otherwise $I = n$.
4: set $V^{UC} = 2(1 - l(I))$
5: for $i = 2$ to $n - 1$ do begin
6:    if $l(i) > l(I)$ then do begin
7:        for $j = i + 1$ to $n$ do $x'[j] = \frac{x[j]-x[i+1]}{1-l(i)}$
8:        for $j = 1$ to $i$ do $x'[j] = \frac{1-x[i+1]+x[j]}{1-l(i)}$
9:        solve problem SP-Line($x'[1], x'[1]$) let $V$ be its solution
10:        if $(1 - l(i))V < V^{UC}$ then do begin
11:          $V^{UC} = (1 - l(i))V$; $I = i$;
12:        endif
13:    endif
14: endfor
15: return $I$; $V^{UC}$
```

Subsection 4.2

Initialization Procedure $L(n + 1, 1)$-I

```
1: Input: for $j = 1$ to $m$, $\omega(1, j) = 0$; and $Y(j) = 0$; $\Delta(B(n+1, 1)) = 0$;
   for $j = 0$ to $m$, $\sigma(j) = 0$ and for $i = 1$ to $n$, $I(i) = 0$; $S(i) = 0$.
2: $i = 1$; $\sigma(\alpha_1) = 1$.
3: for $i = 2$ to $n$ do begin
4:    for $j = 1$ to $m$ do $\omega(i, j) = \omega(i - 1, j)$
5:    $\sigma(\alpha_i) \leftarrow \sigma(\alpha_i) + 1$
6:    $\sigma(\beta_i) \leftarrow \sigma(\beta_i) - 1$
7:    if $\sigma(\alpha_i) < 0$ then $\omega(i, \alpha_i) \leftarrow |\sigma(\alpha_i)|$
8:    if $\sigma(\beta_i) < 0$ then $\omega(i, \beta_i) \leftarrow |\sigma(\beta_i)|$
9:    $S(i) = \sum_{j=1}^{m} \omega(i, j)$;
10: if $S(i) > 0$ then $I(i) = 1$;
11: endfor
12: $\Delta(B(n+1, 1)) = \sum_{i=1}^{n} I(i)l(i)$;; $V = 1 + 2\Delta(B(n+1, 1))$;
13: if $\sum_{i=1}^{n} \omega(i, j) = 0$, then $Y(j) = 1$.
```

Initialization Procedure $L(n + 1, 1)$-II

```
1: **Input**: $\omega(i,j)$ **for** $i = 1$ to $n$, $j = 1$ to $m$; $Y(j)$ **for** $j = 1$ to $m$; **stack** $SK$;
2: **for** $j = 1$ to $m$ **do begin**
3:    **if** $Y(j) = 0$ **then do begin**
4:       $first(j) = 0$; $ns = 0$;
5:       **for** $i = 1$ to $n$ **do begin**
6:         $g(i) = 0$; $end(i) = 0$; $nxtlvl(i) = 0$; $prvlvl(i) = 0$;
          $nxtblck(i) = 0$;
7:       **endfor**
8:       $i = 2$
9:       **while** $\omega(i,j) = 0$ **do** $i \leftarrow i + 1$ **endwhile**
10:        $first(j) = i$;
11:        $PUSH[SK, first(j)]$; $ns \leftarrow ns + 1$; $g(ns) = first(j)$;
12:        **for** $i = first(j) + 1$ to $n$ **do begin**
13:           **if** $ns \neq 0$ **then** $\ell = TOP[SK]$;
14:           **if** $\omega(i,j) > \omega(i-1,j)$ **do begin**
15:              $ns \leftarrow ns + 1$;
16:              **if** $g(ns) > 0$ **then** $nxtblck(g(ns)) = i$;
17:              **if** $g(ns) = 0$ **then do begin**
18:                 $nxtlvl(\ell) = i$; $prvlvl(i) = \ell$; $nxtlvl(i) = n + 1$;
               $prvlvl(n + 1) = i$;
19:              **endif**
20:              $PUSH[SK, i]$; $g(ns) = i$;
21:           **endif**
22:           **if** $\omega(i,j) < \omega(i-1,j)$ **do begin**
23:              $ns \leftarrow ns - 1$; $POP[SK]$; $end(\ell) = i$;
24:              **if** $nxtlvl(\ell) = n + 1$ **then do begin**
25:                 $nxtlvl(prvlvl(\ell)) = n + 1$; $prvlvl(n + 1) = prvlvl(\ell)$;
26:              **endif**
27:           **endif**
28:        **endfor**
29:     **endif**
30: **endfor**
```

Algorithm SP-Circle(CE)(c.w)
Input: $p^* = n + 1$, $f^* = 1$; $\underline{p}$;

```
1: **for** $p = n$ downto $\underline{p}$ **do begin**
2:    **for** $f = 1 \dots \overline{f}(p)$ **do begin**
3:       **if** $f = 1$ **then do begin**
4:          **work on a copy of the data for** $L(p + 1, f)$;
5:          $\Delta = \Delta(B(p + 1, f))$; $j = \alpha_p$;
6:       **endif**
7:       **if** $f > 1$ **then do begin**
8:          $\Delta = \Delta(B(p, f - 1))$; $j = \alpha_f$;
9:       **endif**
10:        **if** $j > 0$ **and** $Y(j) = 0$ **then do begin**
11:           $i = first(j)$;
12:           **while** $i > 0$ **do begin**
13:              **for** $t = i$ **to** $end(i) - 1$ **do begin**
14:                 $S(t) = S(t) - 1$;
15:                 **if** $S(t) = 0$ **then do begin**
16:                    $I(t) = 0$; $\Delta = \Delta - l(t)$;
17:                 **endif**
18:              **endfor**
19:              $i \leftarrow nxtblck(i)$;
20:           **endwhile**
21:           $\Delta(B(p, f)) = \Delta$;
22:           **if** $1 + 2(\Delta + x[f] + 1 - x[p]) < V$ **then do begin**
23:              $p^* = p$; $f^* = f$; $V = 1 + 2(\Delta + 1 - x[p])$;
24:           **endif**
25:           **if** $nxtlvl(first(j)) = n + 1$ **then do begin**
26:              $Y(j) = 1$; $first(j) = 0$;
27:           **endif**
28:           **else** $first(j) \leftarrow nxtlvl(first(j))$;
29:        **endif**
30:        **endfor**
31: **endfor**
32: **Return** $V$, $p^*$, $f^*$
```