Computing 60, 193–215 (1998)

Computing
© Springer-Verlag 1998
Printed in Austria

A Linear Algorithm for the Pos / Neg-Weighted 1-Median Problem on a Cactus*

R. E. Burkard, Graz, and **J. Krarup**, Copenhagen

Received July 2, 1997; revised September 29, 1997

Abstract

The 1-median problem on a network asks for a vertex minimizing the sum of the weighted shortest path distances from itself to all other vertices, each associated with a certain positive weight. We allow for *negative* weights as well and devise an exact algorithm for the resulting 'pos/neg-weighted' problem defined on a cactus. The algorithm visits every vertex just once and runs thus in linear time.

AMS Subject Classifications: 90B80, 90C35, 90C27.

Key words: Location problems, 1-median problem, obnoxious facilities.

1. Introduction

Single-commodity facility location problems deal typically with the location of one or more *facilities* each of which provides the same kind of service to the *users* allocated to it. The facilities to be located are normally regarded as 'friendly' in the sense that 'closeness' is viewed as an attractive property. Location theory, however, does also encompass the counterpart: the location of so-called *obnoxious* facilities where one frequently used criterion is the *maximum* distance between a facility and the closest client. For example, disregarding its workers and subcontractors, who else likes an obnoxious power plant in their backyard? Note that even a friendly facility may well become obnoxious unless 'closeness' is taken with a grain of salt. Thus, optimal closeness to a noisy elementary school is rather 'reachable within a few minutes walk' than 'next door', a feature known as the NIMBY (Not In My Back Yard) syndrome.

The investigation of models with such truly antagonistic criteria capturing both the friendly and the obnoxious aspect of a locational decision problem have attracted several researchers, notably in the 90's, and the field is still gaining further momentum. Our motivation for the present study is a famous problem, allegedly first formulated in the early 1600's by Fermat, and by the so-called

*This research has been supported by the Spezialforschungsbereich F 003 'Optimierung und Kontrolle', Projektbereich Diskrete Optimierung.

'Complementary Problem' proposed in [4]. For three given points A, B, C in a plane, FERMAT asks for a fourth point X such that the sum $AX+BX+CX$ of its (Euclidean) distances to the three given points is minimized. The 'Complementary Problem' CP differs from Fermat in that one of the given points, say point A, is 'obnoxious' which is reflected by the objective function $-AX+BX+CX$; furthermore, $\angle BAC$ exceeds 120°. Both FERMAT and CP can be viewed as special cases of the *Weber Problem* with three users if also negative weights are allowed for. FERMAT represents then the *unweighted* case or the case where the weight associated with each of the given points is $+1$ whereas CP has weights -1, $+1$, $+1$ for A, B, C respectively.

CP appears to be incorrectly solved by Courant and Robbins. This is remedied in Krarup [9] who provides the correct geometrical solution for any set of three given points.

FERMAT and CP are both examples of *continuous* location problems in the sense that the optimizing point sought for can be placed anywhere in the plane. The *discrete* formulations arise when the set of potential sites for the facilities to be placed is finite and often represented by the vertices of a network. For a given graph with positive weights associated with its vertices, Hakimi [6] investigated the so-called *1-median problem* which is to find a vertex such that the sum of the weighted distances to all other vertices becomes as small as possible. The *p-median problem* is to identify a subset P of p vertices minimizing the sum of the weighted shortest path distances from each other vertex in the graph to the *closest* vertex in P. Kariv and Hakimi [8] showed that the problem of finding p medians in a graph is NP-hard.

However, for NP-hard problems defined on a graph, *well-solved special cases* may occur if, for example, the graph possesses certain properties. One such well-solved special case, namely the 1-median problem in a tree, was first considered by Hua et al. [7]. Goldman [5] developed for this problem a simple algorithm which is based on the following observation: If an edge (u,v) is deleted in the tree, we get two trees T_1 and T_2. Let W_i be the weight of all vertices in tree T_i, $i=1, 2$. Then the optimal location lies in the tree with the largest accumulated weight. This observation makes use of the fact that all weights are nonnegative, but does not hold if negative vertex weights are allowed. Thus a new approach is needed in this case. In this paper we shall deal with such an approach which is applicable also to negative weights (and partly also to negative distances). Moreover we consider not only trees, but cacti. A *cactus* is a connected graph where no two cycles have more than one vertex in common. In 1996, Auletta, Parente and Persiano considered a dynamic version of the 1- (and 2-) median problem in trees, where the nonnegative weights of the vertices may change. They developed for the dynamic version of the 1-median problem an algorithm which closely resembles what we shall be doing in the following for the tree structure called graft, and solve the dynamic version in $O(\log n)$ time using preprocessing of $O(n)$ time. Another related work stems from Chen et al. [3]

and deals with block vertex duality and the 1-median problem. In the case of nonnegative weights and distances they identify either a vertex of a subtree as 1-median or a block (in our case: a cycle) which contains the 1-median, whereas we also find the correct vertex within the block (cycle). Their blocking graph resembles our skeleton. Every vertex of a tree, however, forms a single block, whereas we comprise subtrees to 'grafts'. Their duality theory cannot be applied in our case, since we also admit negative distances. In the presence of negative distances the duality between the so-called weight-centroid problem and the 1-median problem does not longer hold.

Another pertinent reference is Bern et al. [2]. Bern et al. provides a general methodology for constructing linear time algorithms for problems of finding optimal subgraphs of graphs defined by certain composition rules (as are trees, series-parallel graphs, and outerplanar graphs); moreover, these subgraphs must possess certain properties.

Section 2 provides the necessary concepts and sets the stage for the ensuing detailed accounts for the data structures employed. In Section 3 we show that a cactus can be decomposed into blocks (that is, tree structures called grafts, and cycles) held together by hinges. Accompanied by numerical examples, Sections 4 and 5 describe the computational procedures for grafts and cycles respectively. The correctness proofs show that the resulting algorithm runs in linear time since every vertex is visited just once. In Section 6 we embed the procedures for the two types of blocks considered into the global optimization routine for cacti.

The prime motivation for this study was, as already said, the interest in location problems with negative weights. As regards realistic applications, however, we believe that models capable of handling arbitrary weights may well provide useful decision support, for example, to companies operating in a competitive environment or to monopolistic companies wishing to increase or decrease the number of existing facilities.

2. Problem Formulation

Let $N=(V,E,w,c)$ be a given connected network with vertex set V and edge set E. With each vertex $x\in V$ we associate a certain *weight* $w(x)$. Likewise, with each edge $(x,y)\in E$ we associate a certain *cost* or *length* $c(x,y)$.

A *cycle* is a sequence $(x_1,\ldots,x_k,x_{k+1}=x_1)$ of k distinct vertices supplemented by $x_{k+1}=x_1$ such that (x_j,x_{j+1}) is an edge, $j=1,\ldots,k$, $k\geq 3$.

N is assumed to be a *cactus*, that is, no two cycles of N have more than one vertex in common. The weights $w(x)$ can be any real number and likewise for the lengths $c(x,y)$, the only exception being that the length of edges included in a cycle must be nonnegative.

The length of a *path* connecting a pair (x, y) of distinct vertices is the sum of the lengths of its constituent edges.

Let $d(x,y)$ be the *shortest path distance* between vertices x and y in N, $x \neq y$. For all $x \in V$, $d(x,x) = 0$.

The objective function $f(x)$ to be minimized is defined for each vertex x and expresses the sum of the weighted shortest path distances between x and all other vertices, that is

$$f(x) = \sum_{y \in V} w(y) d(x,y)$$

The 'pos/neg-weighted' 1-median problem **PN1** can now be stated as follows: For given $N = (V, E, w, c)$ find a vertex α minimizing $f(x)$,

$$f(\alpha) = \min_{x \in V} f(x).$$

Since negative weights are allowed for, smaller values than $f(\alpha)$ of the objective function can be reached if we relax the requirement that $d(x,y)$ is a shortest path distance.

For an ordinary 1-median problem on a network where all weights are positive, the so-called *Vertex Optimality Property* asserts that the problem has an optimal solution at a vertex. This property does not apply in the present case. For example, for the equilateral triangle abc in Fig. 1 we find

$$f(a) = 0 + 2 + 2 = 4, \qquad f(b) = f(c) = -2 + 2 = 0.$$

m is not a vertex but the midpoint between b and c on the edge (b,c). $f(m) = -3 + 1 + 1 = -1 < 0$. Hence, b and c are optimal vertex solutions whereas better solutions like m can be found unless vertex solutions are explicitly requested in the formulation of the problem.

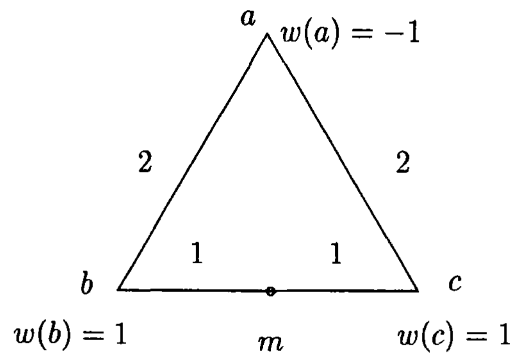

Figure 1. PN1 does not possess the Vertex Optimality Property

To facilitate the overview of the ensuing algorithm **A-PN1** devised for the exact solution of PN1 in linear time, some additional concepts need to be introduced. Initially we partition the vertices of N into three different subsets.

A *C-vertex* is a vertex of degree 2 which is included in exactly one cycle. A *G-vertex* is a vertex not included in any cycle. The remaining vertices, if any, will be referred to as *H-vertices* or *hinges*. It follows from the above that an H-vertex must be included in at least one cycle and be of degree ≥ 3.

A *subtree* is a tree induced by a subset of G- and H-vertices only. A *maximal subtree* is a subtree for which the subset of G- and H-vertices defining it cannot be extended. A *graft* is a maximal subtree with no two H-vertices belonging to the same cycle. Finally, a *block* is a cycle or a graft.

These notions are all illustrated in Fig. 2 where the dotted curves merely are meant to emphasize the maximal subtrees called grafts. Roughly speaking, a cactus does thus consist of blocks which are either cycles or grafts. Note that a graft may be a tree spanning a subset of H-vertices only.

Without knowing the specific values $f(x)$ and $f(y)$ of the objective function for a pair of adjacent vertices, we can, as will be shown, in a simple way calculate

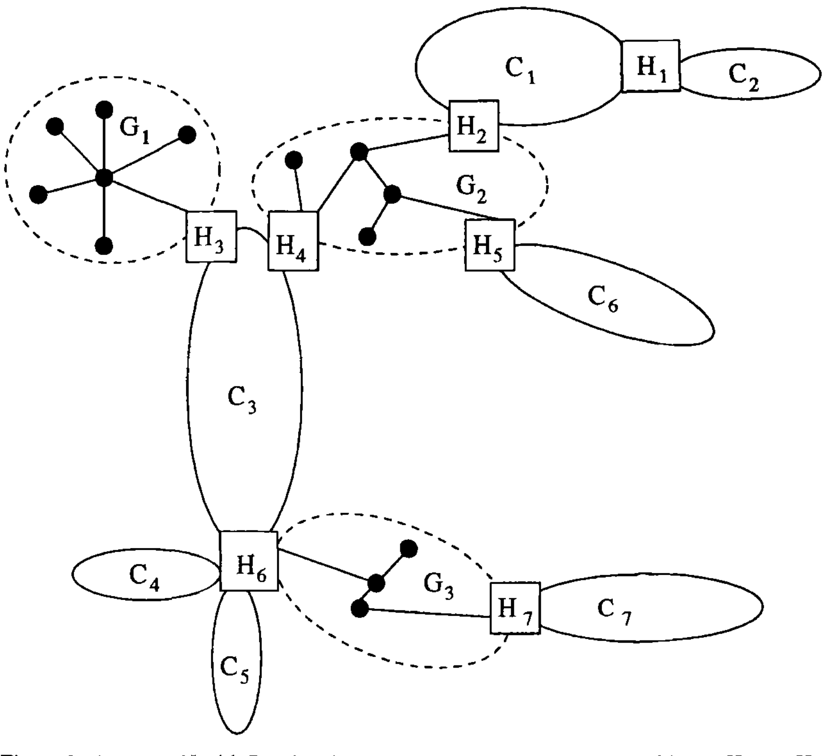

Figure 2. A cactus N with 7 cycles $C_1,\ldots,C_7$, 3 grafts $G_1,\ldots,G_3$ and 7 hinges $H_1,\ldots,H_7$

the difference $\Delta_G(x,y)=f(y)-f(x)$ and so to speak exclude nonoptimal vertices from any further consideration. This operation on a pair (x,y) of adjacent vertices forms the innermost part of A-PN1 and will be referred to as a *primary step*.

Primary steps are only executed on pairs of vertices belonging to the same block. We deal with one block at a time and transfer all information on the block, needed for further computations, to one of its hinges, if any left at all. Unless the algorithm terminates with a *globally optimal solution*, a vertex representing a *locally optimal solution* for the block under consideration has thereby been identified.

The sequence of primary steps needed to eliminate a block constitutes a *secondary step*. Two different types are distinguished: CYCLE(.,.) and GRAFT(.,.) will eliminate a cycle and graft respectively.

In the next section we describe the skeleton of a cactus, a concept which will enable us to perform the secondary step. In Sections 4 and 5, respectively, we describe how grafts and cycles can be handled in linear time. Finally, in Section 6 the interplay between the secondary steps and the primary step is described.

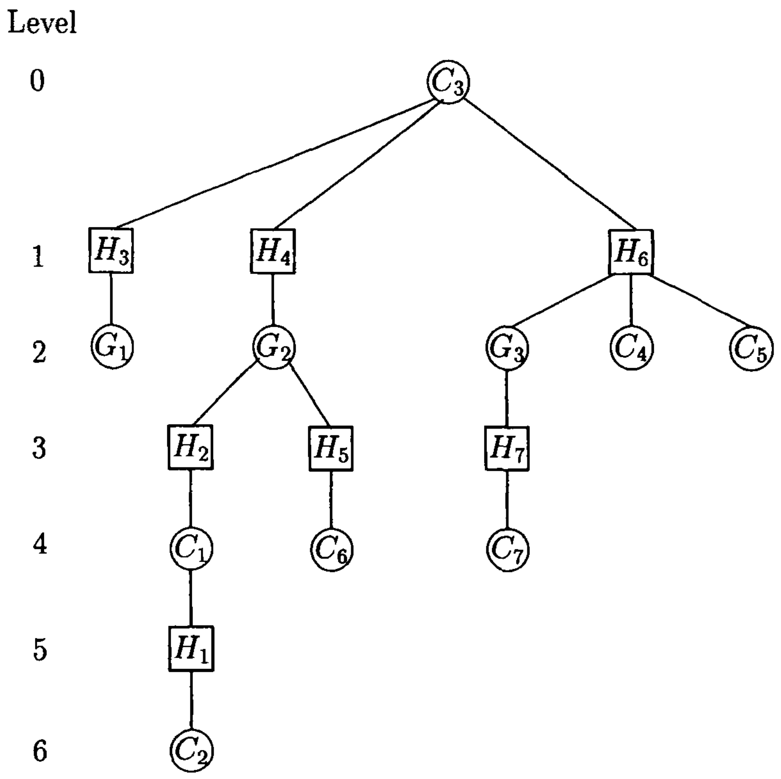

Figure 3. The skeleton S rooted at a block

3. The Skeleton of a Cactus

To establish the processing order of the blocks, we shall define the skeleton of a cactus. Let $C_1, C_2, \ldots, G_1, G_2, \ldots,$ and $H_1, H_2 \ldots$ be the cycles, the grafts, and the hinges of N respectively. Each block (cycle or graft) and each hinge is represented by a vertex in the *skeleton* $S = (V_S, E_S)$ of N.

For each block we join the vertex representing the block itself by an edge to each of its hinges. The graph obtained in this way constitutes the *skeleton* S of N, cf. Fig. 3.

Since N is a connected cactus, the skeleton so defined will become a tree with all end vertices or *leaves* representing blocks.

Two degenerate cases may occur: N itself is either a tree or a single cycle. In both cases, N will comprise a single block only and the skeleton of N reduces accordingly to a single vertex.

Without ambiguity, we can conveniently refer to the vertices of S as blocks and hinges. Since no hinge can be a leaf of S, the degree $deg(H)$ of a hinge H must be at least 2 whereas the degree of a block in S can be a positive integer or even zero in the degenerate cases mentioned above.

To make the skeleton ready for use as intended, we shall first represent S as a *rooted tree* where the root in principle can be any block. The *level number* associated with each vertex in Fig. 3 is the number of edges along the path between that vertex and the root.

For a given block B at level $lev(B) = i$ the *successors* of B form the subset $suc(B) \subset V_S$ of vertices X, $lev(X) > i$, in the subtree of S rooted at B. Thus, for example, $suc(G_2) = \{H_2, H_5, C_1, C_6, H_1, C_2\}$, $suc(G_3) = \{H_7, C_7\}$ and $suc(C_7) = \emptyset$, cf. Figure 3.

If it exists, the *father* of a block is always a hinge-vertex, called its *companion hinge*. For example in Fig. 3, the companion hinge of G_2 is H_4.

If $|V_S| > 1$, the processing order of the blocks is defined as follows. A *live* block is a block which has not been processed as yet. Among the live blocks we select in each secondary step a block B at the highest possible level together with its companion hinge H. Ties in the selection of B can be resolved if we, for example, among the live blocks at the highest level choose the *rightmost* one (although blocks on the same level can in principle be processed in any order). The block is then processed via a call of the procedure CYCLE(B, H) if B is a cycle and GRAFT(B, H) otherwise. In either case we identify a solution which is locally optimal for the subset $\{B\} \cup suc(B)$ of vertices of S. All information such as the optimizing vertex, the total weight of all vertices in $\{B\} \cup suc(B)$, and the data needed for finding $f(\alpha)$, the objective function value corresponding to a global optimal solution α, is passed to the companion hinge H. When no live blocks remain, the last secondary step will process the root itself whereby we are done. For the example exhibited in Fig. 3, the processing order of the blocks will thus become: C_2, C_7, C_6, C_1, C_5, C_4, G_3, G_2 (containing the two hinges H_2 and H_5), G_1 and C_3.

4. Optimization on Grafts

In the following we describe the algorithm in the case where the block to be eliminated is a graft G.

PROCEDURE GRAFT *(G, H)*

Input: A graft G rooted at its companion hinge H. If $G = N$ itself or the root of the skeleton, the root of G can be any vertex of G.

Output: GRAFT(G, H) returns vertex β_G which is a locally optimal solution to G together with $f_G^* = f_G(\beta_G)$, the corresponding (local) value of the objective function restricted to the vertices of G, the sum of all weights of vertices in G and the local function values for all hinge vertices in G.

We shall, for ease of exposition, present GRAFT(G, H) via an illustrative example concurrently with the general remarks. To avoid a too complicated notation involving subscripts on subscripts and the like, we designate the vertices of G by single letters $a, b, \ldots$ not used for other purposes.

Figure 4 shows a graft as a rooted tree.

To each vertex in G we attach its weight $w(x)$ and unless x is the root of G, the length $c(x, y)$ of the edge (x, y) between x and the father y. *Live* vertices are vertices not processed as yet. The root is not considered to be a live vertex. The processing order is identical to that of the blocks of a skeleton: In each primary step we choose the rightmost live vertex among the live vertices at the highest level. It follows from the processing order thus defined that whenever a vertex x is selected, we have already processed the subset $suc(x)$ of all vertices at a higher level than x in the subtree of G rooted at x.

For any vertex x of G, let

$$w^+(x) = \sum_{j \in suc(x)} w(j) \qquad \text{and} \qquad w^-(x) = -w(x) + \sum_{j \notin suc(x)} w(j).$$

By which amount $\Delta_G(x, y) = f(y) - f(x)$ does the objective function value change if we move from vertex x to its father y? Since each pair (u, v) of vertices in a graft is connected by exactly one path, the shortest path distance $d(u, v)$ between u and v must equal the length of that path. For any vertex k, $k \in suc(x)$, $d(k, x) + c(x, y) = d(k, y)$ and the contribution of vertex k to

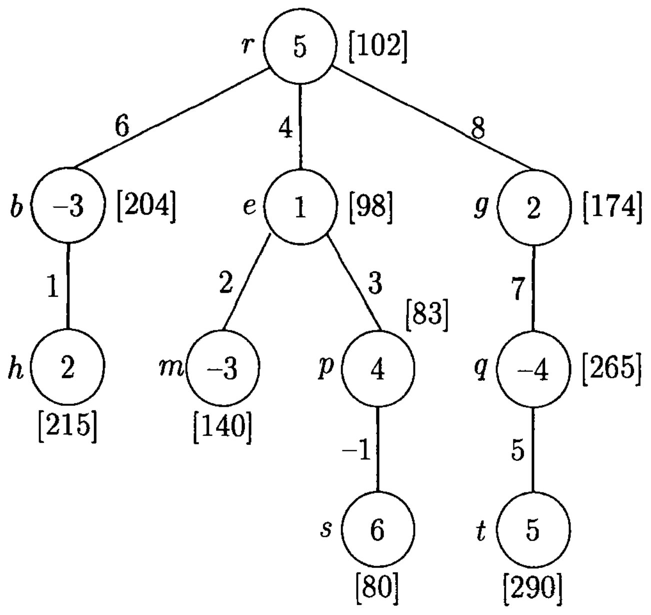

Figure 4. A graft G rooted at r. Circled numbers are the vertex-weights $w(x)$. The other numbers are the edge-lengths $c(x,y)$. The numbers in brackets show the value $f_G(x)$ for each vertex x

$\Delta_G(x,y)$ is accordingly $w(k)c(x,y)$. Similarly, any vertex k, $k \notin suc(x) \cup \{x\}$, will contribute $-w(k)c(x,y)$ to $\Delta_G(x,y)$. Finally, the contribution of x itself is $w(x)c(x,y)$. Hence,

$$\Delta_G(x,y) = f(y) - f(x) = (w^+(x) + w(x) - w^-(x))c(x,y) \qquad (1)$$

Noting that W_G, the total weight of G equals $w^+(x) + w(x) + w^-(x)$ for any vertex x of G, Eq. (1) can be written as

$$\Delta_G(x,y) = [2(w^+(x) + w(x)) - W_G]c(x,y). \qquad (2)$$

For every vertex x in G we require the following information. (We adopt the convention that a sum with an empty index set equals 0.)

- $S(x)$: set of sons of x. If x is a leaf, $S(x) = \emptyset$.
- $w(x)$: weight of the vertex x.
- $w^+(x)$: weight of all successor vertices of x. For $w^+(x)$ we have the following obvious recursion

$$w^+(x) = \sum_{s \in S(x)} [w^+(s) + w(s)].$$

- $c(x,y)$: length of the edge (x,y) from x to its uniquely determined father y, provided that x is different from the root r.
- $\Delta_G(x,y)$: change of the objective value by moving from x to its father y, provided that x is different from the root r. $\Delta_G(x,y)$ is defined by (2), i.e. by

$$\Delta_G(x,y) = [2(w^+(x) + w(x)) - W_G]c(x,y).$$

- $\Delta_G^*(x)$: difference of the objective function between y, the father of x, and a vertex $z \in \{x\} \cup suc(x)$ with a minimum objective function value. If x has only one son s, we have the recursion

$$\Delta_G^*(x) := \Delta_G(x,y) + \max(0, \Delta_G^*(s)).$$

If x has multiple sons, we have to be careful: since we only want a vertex with minimum value, we get

$$\Delta_G^*(x) := \Delta_G(x,y) + \max_{s \in S(x)}(0, \Delta_G^*(s)). \tag{3}$$

If x is the root r, we have

$$\Delta_G^*(r) := \max_{s \in S(r)}(0, \Delta_G^*(s)). \tag{4}$$

- $\beta_G(x)$: name of a vertex with minimum objective function value in the set $suc(x) \cup \{x\}$. Obviously, for every leaf e we get $\beta_G(e) = e$. Otherwise we have the recursion

$$\beta_G(x) = \begin{cases} x, & \text{if } \Delta_G^*(s) \le 0 \text{ for all } s \in S(x) \\ \beta_G(s^*), & \text{if } \Delta_G^*(s^*) = \max_{s \in S(x)}\{\Delta_G^*(s) : \Delta_G^*(s) > 0\} \end{cases} \tag{5}$$

- $\varphi(x)$: contribution of the subtree rooted at x to the objective function value in the root of the graft.

$$\varphi(x) := \sum_{s \in S(x)} \varphi(s) + [w^+(x) + w(x)]c(x,y)$$

where y is the father of x. If x is the root, we have

$$\varphi(x) := \sum_{s \in S(x)} \varphi(s).$$

Let $f_G(x)$ be that part of the global objective function value which is induced by the vertices of G and their weights. We refer to $f_G(x)$ as the local objective function. For the root r of G we have

$$f_G(r) = \varphi(r).$$

In order to get the optimum objective function value for G, we compute

$$f_G^* = \varphi(r) - \Delta_G^*(r), \tag{6}$$

since this amount states the difference between the current and the optimum value found so far. This optimum value is attained in $\beta_G(r)$. The values $f_G(h_i)$ for other hinge vertices h_i of G are determined by $\varphi(r)$ and the changes $\Delta_G(x,y)$ along the uniquely defined path from the hinge vertex h_i to the root. Thus they can easily be determined by the same pass through the vertices of G.

Summarizing we have first to fix the sequence of vertices beginning with the leaves such that the sons of a vertex x are all processed prior to x. Then we compute the total weight W_G of all vertices of G. Afterwards starting with the first vertex in our sequence we compute $w^+(x)$, $\Delta_G(x,y)$, $\Delta_G^*(x)$, $\beta_G(x)$ and $\varphi(x)$ until the root is reached. Finally we compute for the root r the optimum local objective function value according to (6).

For the example shown in Fig. 4 the total weight W_G of all vertices is 15. We furthermore obtain:

x	$S(x)$	$w(x)$	$w^+(x)$	$c(x,y)$	$\Delta_G(x,y)$	$\Delta_G^*(x)$	$\beta_G(x)$	$\varphi(x)$
t	—	5	0	5	−25	−25	t	25
s	—	6	0	−1	3	3	s	−6
m	—	−3	0	2	−42	−42	m	−6
h	—	2	0	1	−11	−11	h	2
q	t	−4	5	7	−91	−91	q	32
p	s	4	6	3	15	18	s	24
g	q	2	1	8	−72	−72	g	56
e	m,p	1	7	4	4	22	s	50
b	h	−3	2	6	−102	−102	b	−4
r	b,e,g					22	s	102

The optimal solution is obtained at vertex s with $f_G^* = \varphi(a) - \Delta_G^*(a) = 80$.

It follows immediately from the updating scheme that the number of elementary operations (additions, multiplications, comparisons) needed to find a (locally) optimal solution for a graft is proportional to the number of its vertices, since every vertex is visited just once and the computational amount for fathoming a vertex is constant. Thus GRAFT(G,H) runs in linear time.

5. Optimization on Cycles

The bookkeeping as regards the shortest paths whenever a vertex is killed is slightly more complicated for cycles than for grafts. As we shall see, however, it is possible to devise a data structure such that a cycle can be eliminated in linear time.

Let C be a cycle with k vertices $x_1, x_2, \ldots, x_k$ and edges (x_i, x_{i+1}) of length $c(i,i+1)$ for $i = 1,2,\ldots,k$, where $x_{k+1} = x_1$. The weights $w(i)$ are positive or negative reals, whereas the lengths $c(i,i+1)$ must be nonnegative. If two or more vertices are connected by edges of length $c(i,i+1) = 0$, we can replace them by a single vertex whose weight is the sum of the single weights. Thus we can assume in the following that all lengths $c(i,i+1)$ are positive.

Let $L = \sum_{i=1}^{k} c(i,i+1)$ be the total length of C. If C has an edge (x_i, x_{i+1}) with length $c(i,i+1) > \frac{1}{2}L$, no shortest path between any pair of vertices of C will include (x_i, x_{i+1}). Thus the edge can be deleted and we can apply procedure GRAFT to the remaining graph (which is just a path). Disregarding ties, the same applies for edges (x_i, x_{i+1}) with $c(i, i+1) = \frac{1}{2}L$. Therefore we assume in the following that $c(i, i+1) < \frac{1}{2}L$ for all edges (x_i, x_{i+1}).

The procedure CYCLE(C, H) investigates the vertices of a cycle C one by one and transfers all information to its companion hinge H.

PROCEDURE CYCLE (C, x_1)

Input: A cycle C, where x_1 is assumed to correspond to the companion hinge of the cycle in the skeleton S, unless the cactus N comprises only this single cycle or C is the root of the skeleton.

Output: CYCLE(C, x_1) returns a locally optimal vertex β_C together with f_C^*, the corresponding value of the objective function restricted to the vertices of C. Moreover, it returns the sum of all weights of vertices in C and the local function values for all hinge vertices in C.

The following description of CYCLE will be made concurrently with the modest-sized data instance as shown in Fig. 5.

The ensuing description of CYCLE will to some extent resemble that of GRAFT. Computationally, however, there are two significant differences between these two procedures:

For a graft, the shortest path tree is the graft itself. For a cycle, however, the shortest path tree may vary from one vertex to another, since we may reach

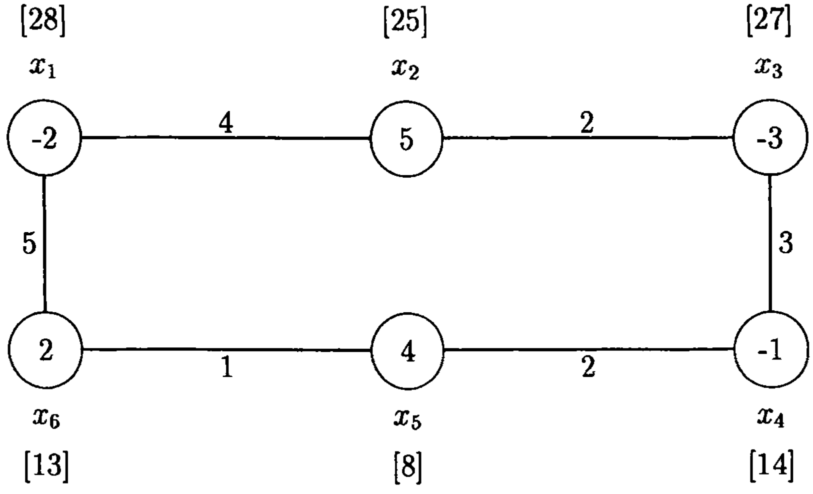

Figure 5. A cycle C with vertices $x_1, \ldots, x_6$. Circled numbers: weights $w(x)$. Numbers in brackets [ ]: f_C, the corresponding local function values. Other numbers: lengths $c(x, y)$

certain vertices either clockwise or counterclockwise. This fact complicates the updating whenever a vertex is being killed. On the other hand, a cycle has no junctions, which simplifies the bookkeeping.

In the following we denote the direction given by $x_1, x_2, \dots, x_k, x_1$ as *clockwise*, whereas $x_1, x_k, \dots, x_2, x_1$ represents the *counterclockwise* direction.

In an initialization step we compute the cycle length, the total weight W_C of all vertices of the cycle, the function value $f_C(x_1)$ and we determine — up to ties— if the shortest path from x_1 to x_j ($j = 2, 3, \dots, k$) is clockwise or counterclockwise. Let $D(p)$ be the length of a *clockwise* path from x_1 to x_p. Thus $D(k+1)$ becomes the total length L of the cycle. We define $\lambda := \frac{1}{2}D(k+1)$. If $D(j) \le \lambda$, a shortest path from x_1 to x_j is clockwise. If $D(j) > \lambda$, the shortest path from x_1 to x_j is counterclockwise. Let x_q be the first among the vertices $x_2, x_3, \dots, x_k$ for which $D(q) > \lambda$ holds. Then the shortest path lengths from x_1 to x_j are given by

$$d(1,j) := \begin{cases} D(j) & \text{for } j < q, \\ D(k+1) - D(j) & \text{for } j \ge q \end{cases} \qquad (j = 2, 3, \dots, k).$$

Thus the function value $f_C(x_1)$ becomes

$$f_C(x_1) = \sum_{j=2}^{q-1} w(j)D(j) + \sum_{j=q}^{k} w(j)[D(k+1) - D(j)].$$

For purposes to become clear a little later, we shall also compute

$$w^* := \sum_{j=2}^{q-1} w(j).$$

Summarizing, we perform the following initialization

```
Procedure INITIALIZE
$p := 1$ ($x_p$ denotes the actual vertex)
$D(P) := 0$, $w_{\cdot}^{+} := 0$, $f_C := 0$, $W_C := 0$
**for** $j := 2$ **to** $k+1$ (mod $k$) **do** $D(j) := D(j-1) + c(j-1, j)$
L := D(1), $\lambda := \frac{1}{2}L$ ($\lambda$ denotes the half cycle length)
**for** $j := 2$ **to** $k$ **do**
  **if** $D(j) \le \lambda$
  **begin**
    $q := j + 1$ (mod $k$)
    $W_C := w^* := w^* + w(j)$
    $f_C := f_C + w(j)D(j)$
  **end**
  **else** $f_C := f_C + w(j)[D(1) - D(j)]$, $W_C := W_C + w(j)$
$W_C := W_C + w(1)$
$\beta_C := p$, $f_C^* := f_C$
```

Thus the procedure INITIALIZE returns the objective function value $f_C(x_1) = f_C^*$ with $\beta_c = 1$, the first vertex x_q, for which the length of a clockwise path from x_1 to x_q is larger than the length of a counterclockwise path and the weight sums w^* and W_C. Note that $q \neq 2$ always holds, since from the beginning we have assumed that $c(i, i+1) < \lambda$ and $q = 2$ would imply $c(1,2) > \lambda$.

Applied to the data instance of Fig. 5 we get

$$D(2) = 4,\ D(3) = 6,\ D(4) = 9,\ D(5) = 11,\ D(6) = 12,\ D(1) = 17,$$
$$\lambda = 8.5,\ q = 4,\ w^* = 2,\ W_C = 5,\ L = 17,$$
$$f_C(x_1) = 5 \cdot 4 + (-3)6 + (-1)[17-9] + 4[17-11] + 2[17-12] = 28.$$

How does the objective function value change, if we proceed from vertex x_p to vertex x_{p+1}?

Let $x_{q(p)}$ be the first among the vertices $x_{p+2}, x_{p+3}, \ldots, x_1, \ldots, x_{p-1}$ for which the length of a clockwise path starting in x_p is greater than λ. Obviously, all vertices $x_{p+2}, \ldots$ which lie on a clockwise shortest path starting in x_p will also lie on a clockwise shortest path starting in x_{p+1}. This implies $q(p+1) \geq q(p)$. Since $c(p, p+1) < \lambda$, we get otherwise $q(p+1) \leq k + p$. (Recall that all indices are to be taken mod k.)

In order to compute $\Delta_C(p, p+1) := f_C(x_{p+1}) - f_C(x_p)$, we distinguish between three classes of vertices and their contribution to $\Delta_C(p, p+1)$, cf. Fig. 6.

First we consider the vertices x_j with $j = p+1, \ldots, q(p) - 1$. All these vertices lie on a clockwise path from x_p as well as from x_{p+1}. Since the clockwise path starting in x_{p+1} does not contain the edge $(p, p+1)$, the objective function decreases by

$$\sum_{j=p+1}^{q(p)-1} w(j) c(p, p+1)$$

Let us define

$$w^* := \sum_{j=p+1}^{q(p)-1} w(j).$$

Thus the contribution to $\Delta_C(p, p+1)$ is $-w^* c(p, p+1)$.

Next we consider the vertices x_j with $j = q(p+1), \ldots, p-1,\ p$. All these vertices lie on a counterclockwise path with respect to x_p as well as with respect to x_{p+1}. Thus their contribution to $\Delta_C(p, p+1)$ is just

$$\sum_{j=q(p+1)}^{p} w(j) c(p, p+1).$$

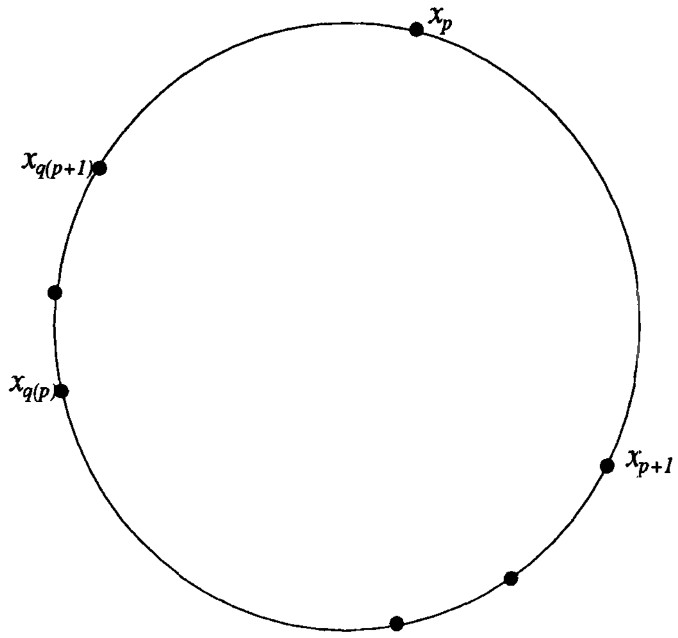

Figure 6. Vertices x_p, x_{p+1}, $x_{q(p)}$ and $x_{q(p+1)}$ along the cycle

Now we consider the vertices x_j with $j = q(p), \dots, q(p+1) - 1$. These vertices lie on a counterclockwise path with respect to x_p, but on a clockwise path with respect to x_{p+1}. Therefore any such vertex x_j contributes the following amount to $\Delta_C(p, p+1)$:

$$\begin{aligned} &w(j) \sum_{r=p+1}^{j-1} c(r, r+1) - w(j)\left[2\lambda - \sum_{r=p}^{j-1} c(r, r+1)\right] \\ &= w(j)\left[c(p, p+1) + 2\sum_{r=p+1}^{j-1} c(r, r+1) - 2\lambda\right] \\ &= w(j)c(p, p+1) + 2w(j)[D(j) - D(p+1) - \lambda]. \end{aligned}$$

Therefore their accumulated contribution to $\Delta_C(p, p+1)$ is

$$\sum_{j=q(p)}^{q(p+1)-1} w(j)c(p, p+1) + 2\sum_{j=q(p)}^{q(p+1)-1} w(j)[D(j) - D(p+1) - \lambda].$$

Summarizing, using that $\sum_{j=q(p)}^{k+p} w(j) = W_C - w^*$, we get for $\Delta_C(p, p+1)$:

$$\begin{aligned} \Delta_C(p, p+1) &= c(p, p+1)(W_C - 2w^*) \\ &\quad + 2\sum_{j=q(p)}^{q(p+1)-1} w(j)[D(j) - D(p+1) - \lambda]. \end{aligned}$$

For example, for $p = 1$, we get

$$\Delta_C(1,2) = c(1,2)(W_C - 2w^*) + 2 \sum_{j=q}^{q(2)-1} w(j)[D(j) - D(2) - \lambda].$$

In our example we get for $\Delta_C(1,2)$ with $q(2) = 7 \equiv 1 \pmod 6$:

$$\Delta_C(1,2) = 4(5 - 2\cdot 2) + 2[(-1)(9 - 4 - 8.5) + 4(11 - 4 - 8.5) + 2(12 - 4 - 8.5)]$$

$$= 4 + 2[3.5 - 6 - 1] = -3.$$

In order to perform the step from p to $p + 1$ we can use the following

```
Procedure FATHOM $p$
$\Delta_C := c(p, p+1)(W_C - 2w^*)$
$\lambda := \lambda + c(p, p+1)$
$D(p) := D(p) + L$
$j := q$
**while** $D(j) < \lambda$ **do**
**begin**
  $q := j + 1 \pmod k$
  $\Delta_C := \Delta_C + 2w(j)[D(j) - \lambda]$
  $w^* := w^* + w(j)$
**end**
$f_C := f_C + \Delta_C$
$w^* := w^* - w(p+1)$
**if** $f_C < f_C^*$ **then**
**begin**
  $f_C^* := f_C$
  $\beta_C := p + 1$
**end**
```

The routine CYCLE consists of the initialization step INITIALIZE and an application of the procedure FATHOM p for $p = 1, 2, \ldots, k - 1$. It returns the optimal vertex x_{β_C} together with the optimal objective function value f_C^*. Moreover, for all hinge vertices x_p of C we need to store the value $f_C(x_p)$. The whole routine CYCLE, applied to our data instance shown in Fig. 5 can be visualized in the following tables:

Given data

j	1	2	3	4	5	6
$w(x_j)$	-2	5	-3	-1	4	2
$c(j, j+1)$	4	2	3	2	1	5

INITIALIZE computes the distances

j	1	2	3	4	5	6
$D(j)$	17	4	6	9	11	12

and $W_C = 5$, $L = 17$.

Starting with vertex x_1 ($p = 1$), we get the following values

	λ	$D(p)$	$q(p)$	$\Delta_C(p-1,p)$	$w^*(p)$	$f_C(p)$	$f_C^*(p)$	β_C
INITIALIZE $p = 1$	8.5	17	4	·	2	28	28	1
FATHOM p: $p = 2$	12.5	21	$7 \equiv 1$	-3	2	25	25	2
$p = 3$	14.5	23	1	2	5	27	25	2
$p = 4$	17.5	26	2	-13	4	14	14	4
$p = 5$	19.5	28	2	-6	0	8	8	5
$p = 6$	20.5	·	·	5	·	13	8	5

Thus the optimal solution is obtained at vertex x_5 and yields the objective function value 8.

As regards the time complexity of the routine CYCLE$(\cdot,\cdot)$ we first note that the initialization procedure runs in $O(k)$ time. It is less apparent that the same actually applies for the total time of all $k-1$ applications of the routine FATHOM p. However, a close look to the **while**-loop reveals that the $q(p)$ values are ascending since the inequalities $1 < q(1) \leq q(2) \leq \ldots \leq q(k) < k + q(1)$ must hold. Thus the total time for executing all **while**-loops is $O(k)$. This shows that the overall time complexity of CYCLE $(\cdot,\cdot)$ is just $O(k)$ as was the case for GRAFT $(\cdot,\cdot)$.

6. Global Optimization on a Cactus

In order to find the global optimum of the objective function, we use the skeleton of the cactus. As already mentioned, we start with processing a block which is a leaf at the highest level. After processing a block we shall see how the global objective function changes when we pass to the immediate predecessing block in the skeleton. So, let us assume that B is a leaf of the skeleton which is connected by its companion hinge h_B to the predecessing block A (cf. Fig. 7).

Let us evaluate the objective function value $f(z)$ for a vertex z of B. For any vertex x belonging to neither A nor B, the shortest path between x and z must have both hinge vertices h_A and h_B as intermediate vertices. Thus such a vertex contributes the following amount to $f(z)$:

$$w(x)[d(x,h_A) + d(h_A,h_B) + d(h_B,z)] \tag{7}$$

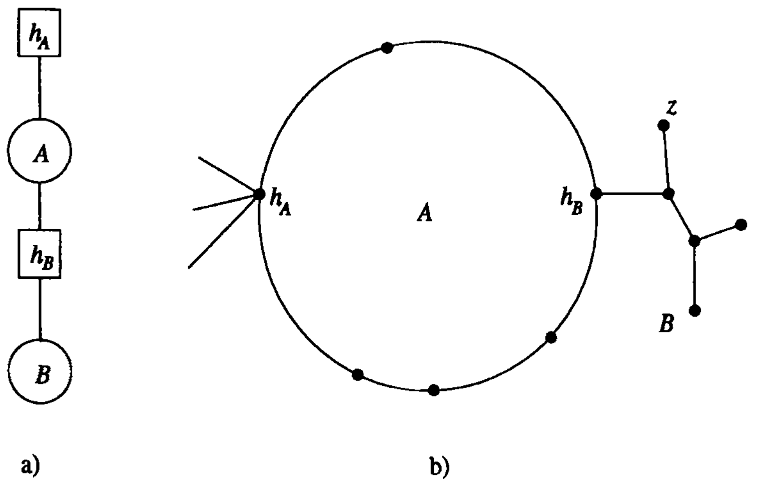

Figure 7. **a** shows a part of the skeleton as exemplified in **b** where A is a cycle and B is a graft

Summing over all such vertices we get for the first summand a constant

$$K = \sum_{x \notin A \cup B} w(x) d(x, h_A),$$

which does not depend on what is done in A or B. If we now replace the given weight of h_A by

$$w^*(h_A) := W - \sum_{y \in A \cup B, y \neq h_A} w(y) \tag{8}$$

where W is the total weight of all vertices of the cactus, and compute the local objective function value $f_A(h_B)$ with respect to the updated weight of hinge vertex h_A, we take care of the second summand in (7) and of the contribution of all vertices except h_B in A to $f(z)$. A similar observation shows that by redefining

$$w^*(h_B) := W - \sum_{y \in B, y \neq h_B} w(y) \tag{9}$$

the local objective function value $f_B(z)$ with respect to the updated weight of hinge vertex h_B takes care of the third summand in (7), as well as of the contribution of all other vertices in A or B. Thus $f(z)$ becomes

$$f(z) = K + f_A(h_B) + f_B(z). \tag{10}$$

This shows how we can compute a locally optimal solution for block B: We replace the given weight $w(h_B)$ by $w^*(h_B)$ according to (9) and apply GRAFT(B, h_B) or CYCLE(B, h_B), whatever is applicable. These procedures yield a locally optimum value

$$f_B^* := \min_{z \in B} f_B(z) \tag{11}$$

which is attained in vertex β_B.

But also a locally optimum solution for block A can now easily be computed. We replace

$$w(h_B) := W_B, \text{ the total weight of all vertices in } B$$

and

$$w(h_A) := w^*(h_A) \text{ according to (8).}$$

Now we apply the procedure GRAFT(A, h_A) or CYCLE(A, h_A) and obtain a locally optimum solution value

$$f_A^* = \min_{z \in A} f_A(z)$$

for the vertices in block A, which is attained in vertex β_A. The corresponding global objective function values for β_A and β_B are

$$f(\beta_A) = K + f_A^* + f_B(h_B)$$
$$f(\beta_B) = K + f_B^* + f_A(h_B)$$

Thus we define

$$\Delta(A,B) := f(\beta_A) - f(\beta_B) = f_A^* - f_B^* + f_B(h_B) - f_A(h_B) \tag{12}$$

If $\Delta(A,B) > 0$, then vertex β_B yields a smaller objective function value than vertex β_A. Thus the optimum within the vertices of A and B is reached at β_B. If $\Delta(A,B) < 0$, then β_A leads to the better value. Note that for evaluating $\Delta(A,B)$ only the outcome of the local procedures GRAFT(A, h_A), GRAFT(B, h_B), CYCLE(A, h_A) and CYCLE(B, h_B), respectively, is needed.

Now let us consider a block A in the skeleton which has several successor blocks $B_1, B_2, \ldots, B_r$ with companion hinges $h_{B_1}, h_{B_2}, \ldots, h_{B_r}$. These companion hinges may not necessarily be distinct. For example, block C_3 in Fig. 3 has five successor blocks G_1, G_2, G_3, C_4 and C_5, but only three different hinges. First we note that all blocks $B_1, B_2, \ldots, B_r$ are processed prior to A. As soon as block B_i is processed we know $f_{B_i}^*$, $f_{B_i}(h_{B_i})$ and we replace $w(h_{B_i})$ by W_{B_i}, the total weight of all vertices in B_i. Thus we take care of all vertices in B_i and possible successor blocks of B_i when processing block A. Finally we process block A. If A has a companion hinge h_A, we define

$$w^*(h_A) := W - \sum_{x \in A,\, x \neq h_A} w(x).$$

If A is the root of the skeleton, any vertex of A can be chosen as its companion hinge. Processing A yields f_A^* taken in the vertex β_A, as well as $f_A(h_{B_i})$ $(i = 1, 2, \ldots, r)$. Thus we can evaluate the differences

$$\Delta(A, B_i) = f_A^* - f_{B_i}^* + f_{B_i}(h_{B_i}) - f_A(h_{B_i}) \qquad (i = 1, 2, \ldots, r).$$

If all these differences are nonpositive, then the optimal solution among the vertices of the blocks A, $B_1, B_2, \ldots, B_r$ is obtained in β_A. Otherwise we must consider

$$\Delta(A, B_s) = \max_{1 \le i \le r} \Delta(A, B_i) > 0$$

and the optimum is attained in β_s.

Let $\Delta^*(A)$ be the difference between $f(\beta_A)$ and the locally optimum objective function value among the successors of the block A. $\Delta^*(A)$ can be computed via the following recursion:

If A is a leaf of the skeleton, define $\Delta^*(A) = 0$.
If A has the blocks $B_1, B_2, \dots, B_r$ as immediate successors, we define

$$\Delta^*(A) = \max_{1 \le i \le r} (0, \Delta(A, B_i) + \Delta^*(B_i)) \tag{13}$$

In order to find the globally optimal vertex we store

$$\alpha(A) := \begin{cases} \beta_A, & \text{if } \Delta^*(A) = 0, \\ \alpha(B_s), & \text{if } \Delta^*(A) = \Delta(A, B_s) + \Delta^*(B_s). \end{cases}$$

When A is the root of the skeleton, $\alpha(A)$ shows a globally optimal vertex of the cactus. In order to compute the globally optimum objective function value we define a function Φ on the blocks of the skeleton as follows:

$$\Phi(B) := \begin{cases} f_B(h_B), & \text{if } B \text{ is a leaf of the skeleton}, \\ f_B(h_B) + \Sigma\Phi(B_i), & B_i \text{ are immediate successors of block } B,\ B \neq R. \end{cases}$$

The optimum value f^* is attained by

$$f^* = \Phi(R) - \Delta^*(R),$$

where R is the root vertex of the skeleton.

Summarizing, we perform the following steps: First we compute W, the total weight of all vertices of the cactus. Then we process the blocks of the skeleton according to decreasing level, starting with the (rightmost) leaves of the highest level.

If block A is processed, where h_A is its companion hinge, we define

$$w^*(h_A) := W - (W_A - w(h_A)).$$

Then we call GRAFT(A, h_A) or CYCLE(A, h_A) and compute $f^*(A)$, a locally optimum value which is attained at vertex β_A, and $f_A(h_{B_i})$ for every hinge vertex $h_{B_i} \neq h_A$. We compute for all direct successor blocks B_i, $i = 1, 2, \dots, r$:

$$\Delta(A, B_i) := f_A^* - f_{B_i}^* + f_{B_i}(h_{B_i}) - f_A(h_{B_i})$$

$$\Delta^*(A) := \max_{1 \le i \le r} (0, \Delta(A, B_i) + \Delta^*(B_i))$$

$$\alpha(A) := \begin{cases} \beta_A, & \text{if } \Delta^*(A) = 0, \\ \alpha(B_s), & \text{if } \Delta^*(A) = \Delta(A, B_s) + \Delta^*(B_s). \end{cases}$$

$$\Phi(A) := \begin{cases} f_A^* + \Sigma_{i=1}^r \Phi(B_i), & \text{if } A \text{ is the root of the skeleton}, \\ f_A(h_A) + \Sigma_{i=1}^r \Phi(B_i), & \text{otherwise}. \end{cases}$$

If A is not the root, we define $w(h_A) := W_A$.

Finally, if R is the root of the skeleton: $\alpha(R)$ returns an optimal vertex of the cactus. The optimum objective function value f^* is given by

$$f^* := \Phi(R) - \Delta^*(R).$$

A last small example shall illustrate this approach. Let the cactus be given as in Fig. 8.

The total weight of all vertices of the cactus is 8. We start with the graft G_5 and process C_3, C_4, G_6, C_2 and G_1 in this order. The actual data are shown in Fig. 9 and Table 1.

When processing C_3 we compute

$$\Delta(C_3, G_5) = -1 - 6 + 6 - 12 = -13.$$

When processing C_2 we compute

$$\Delta(C_2, C_3) = -2 + 1 - 1 - 77 = -79,$$

$$\Delta(C_2, C_4) = -2 + 34 - 34 - 77 = -79,$$

$$\Delta(C_2, G_6) = -2 - 20 + 20 + 2 = 0.$$

After obtaining the root G_1 of the skeleton, we see that vertex a is an optimal solution with value

$$f^* = \Phi(G_1) - \Delta^*(G_1) = 0.$$

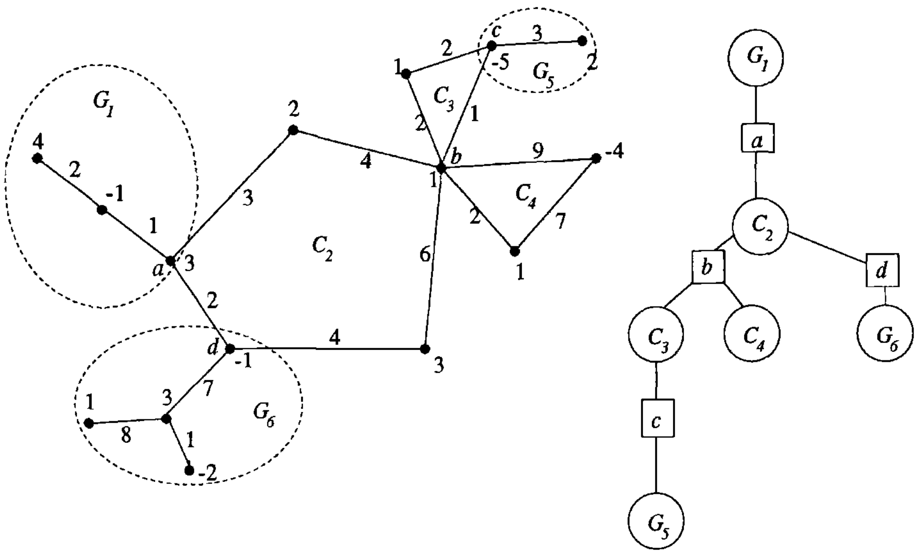

Figure 8. A cactus with 3 grafts G_1, G_5, G_6 and 3 cycles C_2, C_3, C_4 together with its rooted skeleton

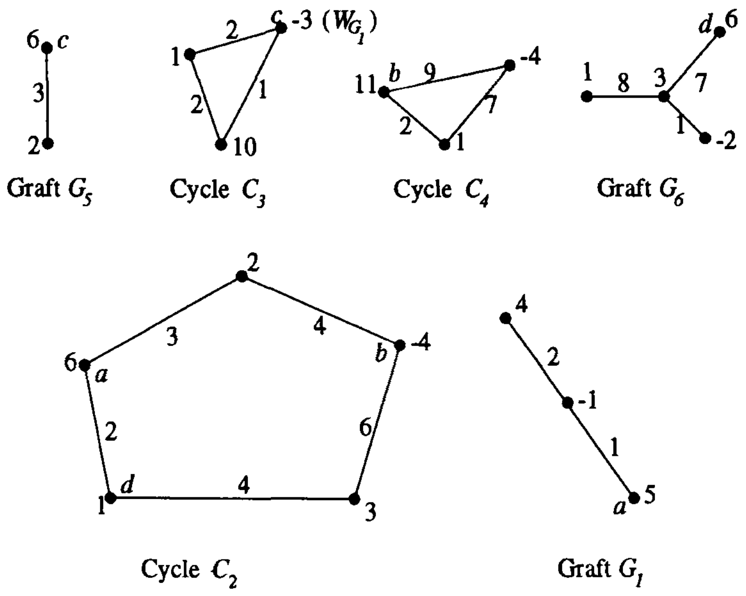

Figure 9. The local optimization problems for the cactus in Fig. 8

Table 1. The secondary steps for the cactus shown in Fig. 8

B	f_B^*	β_B	$\Delta^*(B)$	$\alpha(B)$	$\Phi(B)$	w_B
G_5	6	c	0	c	6	$-3[w(c)]$
C_3	-1	b	0	b	5	$-1[w(b)]$
C_4	-34	b	0	b	-34	$-4[w(b)]$
G_6	20	d	0	d	20	$1[w(d)]$
C_2	-2	a	0	a	-11	$5[w(a)]$
G_1	11	a	0	a	0	

Since every block is processed only once and the processing time of a block with k vertices is $O(k)$, we get an overall time complexity of $O(n)$ for a cactus with n vertices.

References

[1] Auletta, V., Parente, D., Persiano, G.: Dynamic and static algorithms for optimal placement of resources in a tree. Theor. Comput. Sci. *165*, 441–461 (1996).

[2] Bern, M. W., Lawler, E. L., Wong, A.: Linear-time computation of optimal subgraphs of decomposable graphs. J. Algorithms *8*, 216–235 (1987).

[3] Chen, M.-L., Francis, R. L., Lawrence, J. F., Lowe, T. J., Tufekci, S.: Block-vertex duality and the one-median problem. Networks *15*, 395–412 (1985).

[4] Courant, R., Robbins, H.: What is Mathematics? Oxford: Oxford University Press 1941.

[5] Goldman, A. J.: Optimal center location in simple networks. Transport. Sci. *5*, 212–221 (1971).

[6] Hakimi, S. L.: Optimum locations of switching centers and the absolute centers and medians of a graph. Oper. Res. *12*, 450–459 (1964).

[7] Hua, L. K., et al.: Applications of mathematical models to wheat harvesting. Chin. Math. *2*, 77–91 (1962).
[8] Kariv, O., Hakimi, S. L.: An algorithmic approach to network location problems, Part 2: The p-median. SIAM J. Appl. Math. *37*, 539–560 (1979).
[9] Krarup, J.: On 'A Complementary Problem; of Courant and Robbins, Report 96/39, DIKU (Dept. of Computer Science, University of Copenhagen). To appear in Location Theory.

R. E. Burkard
Technische Universität Graz
Institut für Mathematik
Steyrergasse 30
A-8010 Graz, Austria
e-mail: burkard@opt.math.tu-graz.ac.at

J. Krarup
University of Copenhagen
Department of Computer Science
Universitetsparken 1
DK-2100 Copenhagen, Denmark
e-mail: krarup@diku.dk