View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Research Papers in Economics

The Data-Correcting Algorithm for Supermodular Functions, with Applications to Quadratic Cost Partition and Simple Plant Location Problems

Boris Goldengorin[*], Gerard Sierksma[*], Gert A. Tijssen[*], and Michael Tso[†]

27th February 1998

Abstract

The Data-Correcting(DC) Algorithm is a recursive branch and bound type algorithm, in which the data of a given problem instance are 'corrected' at each branching in such a way that the new instance is polynomially solvable and the result satisfies a prescribed accuracy (the difference between optimal and current solution). In this paper the DC algorithm is used for determining exact or approximate global minima of supermodular functions. The working of the algorithm is illustrated by means of a Simple Plant Location Problem. Computational results, obtained for the Quadratic Cost Partition Problem, show that the DC algorithm outperforms the branch-and-cut algorithm, not only for sparse graphs but also for non-sparse (with density at least 40%) graphs (at least 100 times faster). Moreover, for increasing values of the accuracy parameter ε the computational time decreases exponentially with ε.

Keywords: supermodular function, recursive branch-and-bound algorithm, quadratic cost partition problem, simple plant location problem

[*] Department of Econometrics, University of Groningen, P.O. Box 800, 9700 AV Groningen, The Netherlands; e-mail: B.Goldengorin@eco.rug.nl

[†] Department of Mathematics, University of Manchester, Institute of Science and Technology, UMIST, United Kingdom.

1. Introduction

Many combinatorial optimization problems have as an underlying model the minimization of a supermodular (or, equivalently, maximization of a submodular) function, among them being the simple plant location (SPL) problem , generalized transportation problems, the max-cut problem, set covering and other well known problems involving the minimization of Boolean functions; see Nemhauser et al.[13], Lovasz[11] and Barahona et al.[2].

Although the general problem of the minimization of a supermodular function is known to be NP-hard, there has a sustained research effort aimed at developing practical procedures for solving medium and large-scale problems in this class. Often the approach taken has been problem specific, and supermodularity of the underlying objective function has been only implicit to the analysis. For example, Barahona et al.[2] have addressed the max-cut problem from the point of view of polyhedral combinatorics and developed a branch and cut algorithm, suitable for applications in statistical physics and circuit layout design. Beasley[3] applies Lagrangean heuristics to several classes of location problems including SPL problems and reports results of extensive experiments on a Cray supercomputer. Recently, Lee et al.[10] have made a study of the quadratic cost partition (QCP) problem of which max-cut is a special case, again from the standpoint of polyhedral combinatorics.

There have been fewer published attempts to develop algorithms for minimization of a general supermodular function. We believe that the earliest attempt to exploit supermodularity is the work of Petrov and Cherenin[14], who identified a supermodular structure in their study of railway timetabling. Their procedure was subsequently published by Cherenin[5] as the “method of successive calculations”. Their algorithm however is not widely known in the West (Babayev[1]) where, as far we are aware of, the only general procedure that has been studied in depth is the greedy approximation algorithm from Nemhauser et al.[13]. In another greedy approach, Minoux[12] proposed an efficient implementation known as the “accelerated greedy algorithm” that uses a bound already formulated in Khachaturov[9].

In this paper we propose a branch and bound procedure for minimizing a general supermodular function that is based on a generalization of an exclusion principle first established in Cherenin[5]. The proposed procedure improves on the greedy algorithm in finding either an exact or an approximate solution to within a prescribed accuracy bound. The approach we take is to develop, what we term a *Data-Correcting(DC) Algorithm*, which is a class of algorithms proposed by Goldengorin[7] for the solution of NP-hard problems by “correcting” the data of a given problem instance to obtain a new problem instance belonging to a polynomially solvable class.

Computer experiments on random instances of the QCP problem show an improvement upon published results from Lee et al.[10], particularly when the data correspond to a non-sparse graph.

This paper is organized as follows. In Section 2 we establish two symmetric upper bounds for subproblems of the original problem (Theorem 2.1) which are the base for constructing two preservation rules(Corollary 2.1) similar to the ones in Cherenin[5].

We extend the preservation rules in the case where the conditions of Corollary 2.1 are violated. Corollary 2.2 is an attempt to explain what we can do in the case when the preservation rules are not applicable. Together with Lemmas 4.1 and 4.2, and Theorems 5.1 and Theorem 5.2 we obtain upper bounds for the current accuracy between an $\varepsilon-$optimal and an optimal value.

We present the Preliminary Preservation(PP) algorithm, which originally was constructed by Cherenin[5], and we use it for determining the relevant polynomial solvable case of a supermodular function.

In Section 3 we describe the main idea of the DC algorithm. In Section 4 we introduce a specific correction and show how an upper bound for the difference between ε-optimal and optimal values can be calculated. In Section 5 we describe the DC algorithm for determining either an exact global minimum or an approximation of a global minimum with prescribed accuracy. In Section 6 we illustrate the working of the DC algorithm by means of the SPL problem. In the final Section we compare our computational results to results from Lee et al.[10].

2. Supermodular Functions and the Preliminary Preservation (PP) Algorithm

Let Φ be a real-valued function defined on the power set 2^{I^0} of $I^0 = \{1, 2, ..., m\}$; $m \geq 1$. For each $\omega_1, \omega_2 \in 2^{I^0}$ with $\omega_1 \subseteq \omega_2$, define

$$[\omega_1, \omega_2] = \{\omega \in 2^{I^0} \mid \omega_1 \subseteq \omega \subseteq \omega_2\}.$$

Note that $[\emptyset, I^0] = 2^{I^0}$. The interval $[\varphi, I]$ is called a subinterval of $[\emptyset, I^0]$ if $\emptyset \subseteq \varphi \subseteq I \subseteq I^0$; notation $[\varphi, I] \subseteq [\emptyset, I^0]$. In this paper we mean by an interval always a subinterval of $[\emptyset, I^0]$. Throughout this paper, it is assumed that Φ attains a minimum value on $[\emptyset, I^0]$. The minimum value of the function Φ on the interval $[\varphi, I]$ is denoted by $\Phi^*[\varphi, I]$. For $\varepsilon \geq 0$, the problem of *ε-minimizing* the function Φ on $[\varphi, I]$ is to find an element $\beta \in [\varphi, I]$ such that $\Phi(\beta) \leq \Phi^*[\varphi, I] + \varepsilon$; β is called an *ε-minimum* of Φ on $[\varphi, I]$. The function Φ is called *supermodular* on $[\varphi, I]$ if for each $\beta, \gamma \in [\varphi, I]$ it holds that

$$\Phi(\beta) + \Phi(\gamma) \leq \Phi(\beta \cup \gamma) + \Phi(\beta \cap \gamma).$$

Expressions of the form $I \backslash \{k\}$ and $\varphi \cup \{k\}$ will shortly be written as $I - k$ and $\varphi + k$. Let $k \in I \backslash \varphi$ and $[\varphi, I]$ be an interval. The following theorem establishes a relationship between the optimal values of Φ on the two parts of the partitioning $[\varphi, I - k]$, $[\varphi + k, I]$ of $[\varphi, I]$. Note that $[\varphi, I - k] \cap [\varphi + k, I] = \emptyset$. The theorem can be used to decide in which part of the partition $[\varphi, I - k]$, $[\varphi + k, I]$ of $[\varphi, I]$ a minimum of Φ is located.

Theorem 2.1 *Let Φ be a supermodular function on the interval $[\varphi, I] \subseteq [\emptyset, I^0]$ and let $k \in I \backslash \varphi$. Then the following assertions hold.*

a. If $\Phi(\varphi) - \Phi(\varphi + k) = \theta$, then $\Phi^*[\varphi, I - k] - \Phi^*[\varphi + k, I] \leq \theta$.

b. If $\Phi(I) - \Phi(I - k) = \eta$, then $\Phi^*[\varphi + k, I] - \Phi^*[\varphi, I - k] \leq \eta$.

Proof. (a) Let $\gamma \in [\varphi, I - k]$, with $\Phi(\gamma + k) = \Phi^*[\varphi + k, I]$. It then follows from the definition of supermodularity that $\Phi(\varphi + k) + \Phi(\gamma) \leq \Phi(\gamma + k) + \Phi(\varphi)$, which implies that $\Phi(\gamma) \leq \Phi(\gamma + k) + \Phi(\varphi) - \Phi(\varphi + k)$. Hence, $\Phi^*[\varphi, I - k] \leq \Phi(\gamma + k) + \Phi(\varphi) - \Phi(\varphi + k)$, implying that $\Phi^*[\varphi, I - k] - \Phi^*[\varphi + k, I] \leq \Phi(\varphi) - \Phi(\varphi + k) = \theta$. The proof of (b) is left to the reader. $\square$

Theorem 2.1 establishes the conditions for constructing two rules for detecting subintervals containing at least one global minimum of Φ on $[\varphi, I]$.

Corollary 2.1 *Let Φ be a supermodular function on the interval $[\varphi, I] \subseteq [\emptyset, I^0]$, and let $k \in I \backslash \varphi$. Then the following assertions hold.*

a. First Preservation (FP) Rule .
If $\Phi(\varphi + k) \geq \Phi(\varphi)$, then $\Phi^*[\varphi, I] = \Phi^*[\varphi, I - k] \leq \Phi^*[\varphi + k, I]$.

b. Second Preservation (SP) Rule.
If $\Phi(I - k) \geq \Phi(I)$, then $\Phi^*[\varphi, I] = \Phi^*[\varphi + k, I] \leq \Phi^*[\varphi, I - k]$.

Proof. 1a. From Theorem 2.1 we have that $\Phi^*[\varphi, I - k] - \Phi^*[\varphi + k, I] \leq \theta = \Phi(\varphi) - \Phi(\varphi + k)$. By assumption $\Phi(\varphi) - \Phi(\varphi + k) = \theta \leq 0$. Hence, $\Phi^*[\varphi, I] = \Phi^*[\varphi, I - k] \leq \Phi^*[\varphi + k, I]$. The proof of 1b is similar. $\square$

In Corollary 2.2 we present an extension of the rules from Corollary 2.1, appropriate to ε-minimization. In other words, the assumptions of the rules from Corollary 2.2 are true when the assumptions of Corollary 2.1 are false.

Corollary 2.2 *Let Φ be a supermodular function on the interval $[\varphi, I] \subseteq [\emptyset, I^0]$, and $k \in I \backslash \varphi$. Then the following assertions hold.*

a. *First θ-Preservation (θ-FP) Rule.*
   If $\Phi(\varphi) - \Phi(\varphi + k) = \theta \geq 0$, then $\Phi^*[\varphi, I - k] - \Phi^*[\varphi, I] \leq \theta$,
   which means that $[\varphi, I - k]$ contains a θ-minimum of $[\varphi, I]$.
b. *Second η-Preservation (η-SP) Rule.*
   If $\Phi(I) - \Phi(I - k) = \eta \geq 0$, then $\Phi^*[\varphi + k, I] - \Phi^*[\varphi, I] \leq \eta$,
   which means that $[\varphi + k, I]$ contains a η-minimum of $[\varphi, I]$.

Proof. The proof of part (a) is as follows. Case 1. If $\Phi^*[\varphi, I] = \Phi^*[\varphi, I - k]$ then $\Phi^*[\varphi, I - k] \leq \Phi^*[\varphi, I - k] + \theta$ or $\Phi^*[\varphi, I - k] \leq \Phi^*[\varphi, I] + \theta$. Case 2. If $\Phi^*[\varphi, I] = \Phi^*[\varphi + k, I]$, then from Theorem 2.1a follows that $\Phi^*[\varphi, I - k] - \Phi^*[\varphi + k, I] \leq \theta$ or $\Phi^*[\varphi, I - k] - \Phi^*[\varphi, I] \leq \theta$. The proof of (b) is similar. □

By means of Corollary 2.1 it is often possible to exclude a large part of $[\emptyset, I^0]$ from consideration when determining a global minimum of Φ on $[\emptyset, I^0]$. The so called *Preliminary Preservation (PP) algorithm* determines a subinterval $[\varphi, I]$ of $[\emptyset, I^0]$ that certainly contains a global minimum Φ, whereas $[\varphi, I]$ cannot be made smaller by using the preservation rules of Corollary 2.1.

Let $[\varphi, I]$ be an interval. For each $i \in I \backslash \varphi$, define $\delta^+(\varphi, I, i) = \Phi(I) - \Phi(I - i)$ and $\delta^-(\varphi, I, i) = \Phi(\varphi) - \Phi(\varphi + i)$; moreover, define $\delta^+_{\min}(\varphi, I) = \min\{\delta^+(\varphi, I, i) \mid i \in I \backslash \varphi\}$, $r^+(I, \varphi) = \min\{r \mid \delta^+(\varphi, I, r) = \delta^+_{\min}(\varphi, I)\}$.

Similarly, for $\delta^-(\varphi, I, i)$ define $\delta^-_{\min}(\varphi, I) = \min\{\delta^-(\varphi, I, i) \mid i \in I \backslash \varphi\}$, $r^-(I, \varphi) = \min\{r \mid \delta^-(\varphi, I, r) = \delta^-_{\min}(\varphi, I)\}$. If no confusion is likely, we shortly write r^-, r^+, δ^-, δ^+ instead of $r^-(I, \varphi)$, $r^+(I, \varphi)$, $\delta^-_{\min}(\varphi, I)$, and $\delta^+_{\min}(\varphi, I)$ respectively.

The Preliminary Preservation Algorithm

```
Procedure PP($\tau, \sigma; \varphi, I$)
==========================================
Input: A supermodular function $\Phi$ on the subinterval $[\tau, \sigma]$ of $[\emptyset, I^0]$.
Output: A subinterval $[\varphi, I]$ of $[\tau, \sigma]$ such that $\Phi^*[\varphi, I] = \Phi^*[\tau, \sigma]$,
        $\Phi(\varphi) > \Phi(\varphi + i)$ and $\Phi(I) > \Phi(I - i)$ for each $i \in I \backslash \varphi$.
==========================================
begin $\varphi := \tau$; $I := \sigma$;
    Step 1: if $\varphi = I$
            then goto Step 4;
    Step 2: Calculate $\delta^+$ and $r^+$;
            if $\delta^+ \leq 0$ {Corollary 2.1b}
```

```
                then  begin call PP($\varphi + r^+, I; \varphi, I$);
                                goto Step 4;
                        end;
        Step 3: Calculate $\delta^-$ and $r^-$;
                if $\delta^- \leq 0$ {Corollary 2.1a}
                then  begin call PP($\varphi, I - r^-; \varphi, I$);
                                goto Step 4;
                        end;
        Step 4:
end;
=========================================
```

Each time φ or I are updated during the execution of the PP algorithm, the conditions of Corollary 2.1 remain satisfied, and therefore the invariant $\Phi^*[\varphi, I] = \Phi^*[\tau, \sigma]$ remains valid at each step. At the end of the algorithm we have that $\min\{\delta^+, \delta^-\} > 0$, which shows that $\Phi(\varphi) > \Phi(\varphi + i)$ and $\Phi(I) > \Phi(I - i)$ for each $i \in I \backslash \varphi$. Hence Corollary 2.1 cannot be applied for further reduction of the interval $[\varphi, I]$ without violation $\Phi^*[\varphi, I] = \Phi^*[\tau, \sigma]$. Note that this remark shows the correctness of the PP algorithm.

In Section 4 we will use Corollary 2.1 to make the interval $[\varphi, I] \subseteq [\tau, \sigma]$ even smaller by violating $\Phi^*[\varphi, I] = \Phi^*[\tau, \sigma]$ not too much, namely, for a prescribed accuracy ε it will hold that $\Phi^*[\varphi, I] - \Phi^*[\tau, \sigma] \leq \varepsilon$.

The following theorem can also be found in Goldengorin[6]. It provides an upper bound for the worst case complexity of the PP algorithm; the complexity function is taken only dependent of the number of comparisons of values for $\Phi(\omega)$.

Theorem 2.2 *The time complexity of the PP algorithm is at most $O(m^2)$.*

Proof. In Steps 2 and 3 at most 2m comparisons are made. If the comparisons do not result in an update of either φ or I, then the algorithm stops. Each time the algorithm is executed, the number of elements in $I \backslash \varphi$ is decreased by at least one. The PP algorithm starts with $I^0 = \{1, 2, .., m\}$, so that the number of comparisons is bounded from above by $(2)[m + (m - 1) + ... + 1] = (m)(m + 1)$. Hence the time complexity of the algorithm is at most $O(m^2)$. □

Note that if the PP algorithm terminates with $\varphi = I$, then φ is a global minimum of Φ on $[\tau, \sigma]$. Any supermodular function Φ on $[\tau, \sigma]$ for which the PP algorithm returns a global minimum for Φ is called a *PP-function.*

In the following example Φ is a PP-function; we use it for illustrating the working of the PP algorithm. Let $I^0 = \{1, 2, 3\}$; the values of Φ are given in Table 2.1.

ω	{Ø}	{1}	{2}	{3}	{1,2}	{1,3}	{2,3}	{1,2,3}
$\Phi(\omega)$	10	10	8	7	8	12	8	20

Table 2.1: An example of a PP-function

After the first execution of Step 3, we have that $[\varphi, I] = [\{\emptyset\}, \{2, 3\}]$, because $\delta^- = \Phi(\emptyset) - \Phi(\{1\}) = 0$, and $r^- = 1$. After the second execution of Step 2 we have that $[\varphi, I] = [\{3\}, \{2, 3\}]$, because $\delta^+ = \Phi(\{2, 3\}) - \Phi(\{2\}) = 0$ and $r^+ = 3$. Finally, after the third execution we have that $[\varphi, I] = \{3\}$, because $\delta^- = \Phi(\{3\}) - \Phi(\{2, 3\}) = -1$, and $r^- = 2$. So, $\varphi = I$, and hence Φ is a $PP-$function.

3. The main idea of the DC algorithm

Recall that if a supermodular function Φ is not a PP-function, then the PP algorithm terminates with a subinterval $[\varphi, I]$ of $[\emptyset, I^0]$ with $\varphi \neq I$ that contains a minimum of Φ without knowing its exact location in $[\varphi, I]$. In this case, the conditions

$$\begin{aligned} \delta^+(\varphi, I, i) > 0 \quad & \text{for } i \in I\backslash\varphi, \quad (1) \\ \text{and } \delta^-(\varphi, I, i) > 0 \quad & \text{for } i \in I\backslash\varphi. \quad (2) \end{aligned}$$

are satisfied at termination of the PP algorithm. The basic idea of the DC algorithm is that if a situation occurs on which both (1) and (2) hold, then the data of the current problem will be ‘corrected’ in such a way that a *corrected* function Ψ satisfies at most one of the conditions (1) or (2). After the correction the following situations may occur:

- The corrected function is not supermodular, and additional conditions which guarantee its supermodularity are not known;

- The corrected function is not supermodular, and additional conditions which guarantee its supermodularity are known, but the complexity of the verification of the supermodularity is equivalent to the complexity of solving the original problem;

- The corrected function is not supermodular, and additional conditions which guarantee the supermodularity are known; moreover, the complexity of the verification of the supermodularity is polynomial (efficient);

- The corrected function is supermodular (and it is possible to prove this fact).

In this paper we will restrict ourselves to the latter situation for which the supermodularity of the corrected function easy to prove. The other situations are also of interest but the investigation of them is subject of further research. Hence a situation is studied for which there is an $i \in I\backslash\varphi$, such that either $\Psi(I-i) \geq \Psi(I)$ or $\Psi(\varphi+i) \geq \Psi(\varphi)$ holds. Now Corollary 2.1 can be applied again, and we are in the situation that the PP algorithm can be applied. For all possible elements i we try to choose one for which the correction procedure satisfies the prescribed ε_0. If such an element i does not exist, we choose an arbitrary $i \in I\backslash\varphi$ and branch the current problem into two subproblems, one on $[\varphi+i, I]$ and one on $[\varphi, I-i]$.

Anyway, we should find answers for the following questions:

- How should the difference between the value of a global minima of the corrected and the uncorrected functions be estimated, and, how does this difference depend on the specific corrections?

- How should the above mentioned difference be decreased in case it does not satisfy the prescribed accuracy ε_0?

The answers to these questions can be found in the Section 4.

4. The Correcting Rules; an extension of the PP algorithm.

In order to preserve the supermodularity we will use the following correcting rules.

Let $\emptyset \subseteq \varphi \subseteq I \subseteq I^0$, and $r^+, r^- \in I\backslash\varphi$. Moreover, let Φ be a supermodular function on $[\emptyset, I^0]$. Then for each $\omega \in [\varphi, I]$ define:
Correcting Rule 1 (CR1):

$$\Psi(\omega) = \begin{cases} \Phi(\omega) + \delta^+(\varphi, I, r^+), & \text{if } r^+ \notin \omega; \\ \Phi(\omega), & \text{otherwise} \end{cases}$$

The Correcting Rule 2 (CR2).

$$\Psi(\omega) = \begin{cases} \Phi(\omega) + \delta^-(\varphi, I, r^-), & \text{if } r^- \in \omega; \\ \Phi(\omega), & \text{otherwise} \end{cases}$$

It can be easily seen that if Φ is supermodular on a certain interval, then so is Ψ.

An extension of the PP algorithm is based on the following lemma's.

Lemma 4.1 *Let Φ be a supermodular function on the interval $[\varphi, I] \subseteq [\emptyset, I^0]$ and let $i \in I\backslash\varphi$. Then the following assertions hold.*

a. If $\delta = \Phi(\varphi) - \Phi(\varphi + i) \leq 0$ *and* $\Phi(\lambda) - \Phi^*[\varphi, I - i] \leq \gamma \leq \varepsilon$*,*
then $\Phi(\lambda) - \Phi^*[\varphi, I] \leq \gamma \leq \varepsilon$*.*
b. If $\delta = \Phi(I) - \Phi(I - i) \leq 0$ *and* $\Phi(\lambda) - \Phi^*[\varphi + i, I] \leq \gamma \leq \varepsilon$*,*
then $\Phi(\lambda) - \Phi^*[\varphi, I] \leq \gamma \leq \varepsilon$*.*
c. If $0 \leq \delta = \Phi(\varphi) - \Phi(\varphi + i) \leq \varepsilon$*, and* $\Phi(\lambda) - \Phi^*[\varphi, I - i] \leq \gamma \leq \varepsilon - \delta$*,*
then $\Phi(\lambda) - \Phi^*[\varphi, I] \leq \gamma + \delta \leq \varepsilon$*.*
d. If $0 \leq \delta = \Phi(I) - \Phi(I - i) \leq \varepsilon$*, and* $\Phi(\lambda) - \Phi^*[\varphi + i, I] \leq \gamma \leq \varepsilon - \delta$*,*
then $\Phi(\lambda) - \Phi^*[\varphi, I] \leq \gamma + \delta \leq \varepsilon$*.*

Proof. The proof of (a) is as follows. From $\delta \leq 0$ and Corollary 2.1a we obtain that $\Phi^*[\varphi, I] = \Phi^*[\varphi, I - i]$. Hence $\Phi(\lambda) - \Phi^*[\varphi, I] = \Phi(\lambda) - \Phi^*[\varphi, I - i] \leq \gamma \leq \varepsilon$. Since the proof of (b) is similar to that of (a) we conclude with a proof of (c).
From $\delta \geq 0$ and Corollary 2.2a we obtain that $\Phi^*[\varphi, I - i] - \Phi^*[\varphi, I] \leq \delta$ or $-\Phi^*[\varphi, I] \leq -\Phi^*[\varphi, I - i] + \delta$ or $\Phi(\lambda) - \Phi^*[\varphi, I] \leq \Phi(\lambda) - \Phi^*[\varphi, I - i] + \delta \leq \gamma + \delta \leq \varepsilon - \delta + \delta = \varepsilon$. □

The following lemma implies the branching step in the DC algorithm.

Lemma 4.2 *Let* Φ *be an arbitrary function on the interval* $[\varphi, I] \subseteq [\emptyset, I^0]$ *and let* $k \in I \backslash \varphi$*. Then for any* $\varepsilon \geq 0$ *the following assertion holds.*
If $\Phi(\lambda^-) - \Phi^*[\varphi, I - k] \leq \gamma^- \leq \varepsilon$*, and* $\Phi(\lambda^+) - \Phi^*[\varphi + k, I] \leq \gamma^+ \leq \varepsilon$*, for some* $\lambda^-, \lambda^+ \in [\emptyset, I^0]$ *and some* γ^- *and* γ^+*, then* $\min\{\Phi(\lambda^-), \Phi(\lambda^+)\} - \Phi^*[\varphi, I] \leq \max\{\gamma^-, \gamma^+\} \leq \varepsilon$.

Proof. Suppose that $\Phi^*[\varphi, I] = \Phi^*[\varphi, I - k]$. Hence $\min\{\Phi(\lambda^-), \Phi(\lambda^+)\} - \Phi^*[\varphi, I] = \min\{\Phi(\lambda^-), \Phi(\lambda^+)\} - \Phi^*[\varphi, I - k] \leq \Phi(\lambda^-) - \Phi^*[\varphi, I - k] \leq \gamma^- \leq \varepsilon$.
If $\Phi^*[\varphi, I] = \Phi^*[\varphi + k, I]$, then $\min\{\Phi(\lambda^-), \Phi(\lambda^+)\} - \Phi^*[\varphi, I] = \min\{\Phi(\lambda^-), \Phi(\lambda^+)\} - \Phi^*[\varphi + k, I] \leq \Phi(\lambda^+) - \Phi^*[\varphi + k, I] \leq \gamma^+ \leq \varepsilon$. Hence, $\min\{\Phi(\lambda^-), \Phi(\lambda^+)\} - \Phi^*[\varphi, I] \leq \max\{\gamma^-, \gamma^+\} \leq \varepsilon$. □

The main step of the DC algorithm, to be formulated in Section 5, is, what we call, the Procedure DC(). The input parameters of the Procedure DC() are an interval $[\varphi, I]$, and a prescribed value of ε; the output parameters are λ and γ, with $\lambda \in [\emptyset, I^0]$, and $\Phi(\lambda) - \Phi^*[\varphi, I] \leq \gamma \leq \varepsilon$. The value of γ is an upper bound for the accuracy of $\Phi(\lambda) - \Phi^*[\varphi, I]$, and may sometimes be smaller than the prescribed accuracy ε. The procedure starts with trying to make the interval $[\varphi, I]$ as small as possible by using Lemmas 4.1a and 4.1b. If this is not possible, it splits the interval into two subintervals, and adjusts Φ to Ψ. Then, with the help of Lemmas 4.1c and 4.1d it may be possible to fathom one of the two subintervals. If this is not possible, the

Procedure DC() will use the following branching rule.

Branching Rule.
For $k \in \arg\max\{\max[\delta^-(\varphi, I, i), \delta^+(\varphi, I, i)] \mid i \in I \backslash \varphi\}$, split the interval $[\varphi, I]$ into two subintervals $[\varphi + k, I]$, $[\varphi, I - k]$, and use the prescribed accuracy ε of $[\varphi, I]$ for both subintervals.

Our choice for the branching variable $i \in I \backslash \varphi$ is motivated by the observation that $\delta^+(\varphi, I, r^+) \geq \delta^+(\varphi, I - k, r^+)$ and $\delta^+(\varphi, I, r^-) \geq \delta^-(\varphi + k, I, r^-)$, following straightforwardly from the supermodularity of Φ. Hence, the values of δ^+, δ^-, for given r^+, r^-, are seen to decrease monotonically with successive branchings. Our choice is aimed at making the right hand sides δ^+, δ^- as small as possible after branching (and if possible nonpositive), with the purpose of increasing the ‘probability’ of satisfying the preservation rules (see Corollary 2.1). Moreover, this branching rule makes the upper bound for the difference between a γ-minimum and a global minimum as large as possible.

Note that in Procedure DC() λ need not be in the interval $[\varphi, I]$. In most branch and bound algorithms a solution for a subproblem is searched inside the solution space of that subproblem. From the proofs of Lemmas 4.1 and 4.2 it can been seen that this is not necessary here. For any prescribed accuracy ε the Procedure DC() reads now as follows.

```
Procedure DC($\varphi, I, \varepsilon; \lambda, \gamma$)
==========================================
Input: A supermodular function $\Phi$ on the interval $[\varphi, I]$, $\varepsilon \geq 0$.
Output: $\lambda \in [\emptyset, I^0]$ and $\gamma \geq 0$ such that $\Phi(\lambda) - \Phi^*[\varphi, I] \leq \gamma \leq \varepsilon$.
==========================================
begin
      Step 1: if $\varphi = I$
            then begin $\lambda := \varphi$; $\gamma := 0$;
                       goto Step 7;
                 end
      Step 2: Calculate $\delta^+$ and $r^+$;
            if $\delta^+ \leq 0$
            then begin call DC($\varphi + r^+, I, \varepsilon; \lambda, \gamma$);
                       {Lemma 4.1b} goto Step 7;
                 end
      Step 3: Calculate $\delta^-$ and $r^-$;
            if $\delta^- \leq 0$
            then begin call DC($\varphi, I - r^-, \varepsilon; \lambda, \gamma$);
```

```
                {Lemma 4.1a} goto Step 7;
            end
    Step 4: if $\delta^+ \leq \varepsilon$
            then begin call DC($\varphi + r^+, I, \varepsilon - \delta^+; \lambda, \gamma$);
                $\gamma := \gamma + \delta^+$ {Lemma 4.1d};
                goto Step 7;
            end
    Step 5: if $\delta^- \leq \varepsilon$
            then begin call DC($\varphi, I - r^-, \varepsilon - \delta^-; \lambda, \gamma$);
                $\gamma := \gamma + \delta^-$ {Lemma 4.1c};
                goto Step 7;
            end
    Step 6: Select $k \in I \backslash \varphi$ (Branching Rule)
            call DC($\varphi + k, I, \varepsilon; \lambda^+, \gamma^+$)
            call DC($\varphi, I - k, \varepsilon; \lambda^-, \gamma^-$)
            $\lambda :=$ arg min{$\Phi(\lambda^-), \Phi(\lambda^+)$} {Lemma 4.2}
            $\gamma :=$ max{$\gamma^-, \gamma^+$}
    Step 7: {$\Phi(\lambda) - \Phi^*[\varphi, I] \leq \gamma \leq \varepsilon$}
end;
=========================================
```

In Section 6 we will illustrate this algorithm by solving a SPL problem.

5. The Data-Correcting(DC) Algorithm

The DC algorithm is a branch-and-bound type algorithm, and is presented as a recursive procedure.

The Data-Correcting Algorithm

```
=========================================
Input: A supermodular function $\Phi$ on $[\emptyset, I^0]$ and a prescribed
       accuracy $\varepsilon_0 \geq 0$.
Output: $\lambda \in [\emptyset, I^0]$ and $\gamma \geq 0$ such that $\Phi(\lambda) - \Phi^*[\emptyset, I^0] \leq \gamma \leq \varepsilon_0$.
=========================================
 begin
       call DC($\emptyset, I^0, \varepsilon_0; \lambda, \gamma$)
 end;
=========================================
```

Theorem 5.1 *(Correctness of the DC algorithm). For any supermodular function Φ defined on the interval $[\emptyset, I^0]$ and for any accuracy $\varepsilon_0 \geq 0$, the DC algorithm constructs a $\lambda \in [\emptyset, I^0]$ and a $\gamma \geq 0$ such that $\Phi(\lambda) - \Phi^*[\emptyset, I^0] \leq \gamma \leq \varepsilon_0$.*

Proof. We only need to show that each step of the DC algorithm is correct. The correctness of Step 1 follows from the fact that if $\varphi = I$ then the interval $[\varphi, I]$ contains a unique solution and λ satisfies the prescribed accuracy ε_0 (i.e., $\Phi(\lambda) - \Phi^*[\varphi, I] = \Phi(\lambda) - \Phi(\lambda) = 0 \leq \gamma \leq \varepsilon_0$).

The correctness of Steps 2 and 3 follows from Lemma 4.1b and Lemma 4.1a, respectively. The correctness of Steps 4 and 5 follows from Lemma 4.1d and Lemma 4.1c, respectively; and the correctness of Step 6 follows from Lemma 4.2. So, if the Procedure DC() is called with the arguments $\emptyset$, I^0 and ε_0, then, when it is finished, $\Phi(\lambda) - \Phi^*[\emptyset, I^0] \leq \gamma \leq \varepsilon_0$ holds. $\square$

In the following theorem we show how a current value of γ (in the DC algorithm) can be decreased.

Theorem 5.2 *Let Φ be an arbitrary function on the interval $[\varphi, I] \subseteq [\emptyset, I^0]$ and let $k \in I\backslash\varphi$. Then for any $\varepsilon \geq 0$ the following assertion holds.*
If $\Phi(\lambda^-) - \Phi^*[\varphi, I - k] \leq \gamma^- \leq \varepsilon$ and $\Phi(\lambda^+) - \Phi^*[\varphi + k, I] \leq \gamma^+ \leq \varepsilon$ for some $\lambda^-, \lambda^+ \in [\emptyset, I^0]$ and some γ^- and γ^+, then $\min\{\Phi(\lambda^-), \Phi(\lambda^+)\} - \Phi^*[\varphi, I] \leq \min\{\Phi(\lambda^-), \Phi(\lambda^+)\} - \min\{\Phi(\lambda^+) - \gamma^+, \Phi(\lambda^-) - \gamma^-\} = \gamma \leq \max\{\gamma^-, \gamma^+\} \leq \varepsilon$.

Proof. $\min\{\Phi(\lambda^-), \Phi(\lambda^+)\} - \Phi^*[\varphi, I] =$
$\min\{\Phi(\lambda^-), \Phi(\lambda^+)\} - \min\{\Phi^*[\varphi + k, I], \Phi^*[\varphi, I - k] =$
$\min\{\Phi(\lambda^-), \Phi(\lambda^+)\} + \max\{-\Phi^*[\varphi + k, I], -\Phi^*[\varphi, I - k] \leq$
$\min\{\Phi(\lambda^-), \Phi(\lambda^+)\} + \max\{\gamma^+ - \Phi(\lambda^+), \gamma^- - \Phi(\lambda^-)\} =$
$\min\{\Phi(\lambda^-), \Phi(\lambda^+)\} - \min\{\Phi(\lambda^-) - \gamma^-, \Phi(\lambda^+) - \gamma^+\} \leq \max\{\gamma^-, \gamma^+\} \leq \varepsilon$. $\square$

Note that in Lemma 4.2 and Theorem 5.2 Φ need not be a supermodular function. In order to use this theorem in the Procedure DC(), it is enough to replace $\gamma := \max\{\gamma^-, \gamma^+\}$ in step 6 by $\gamma := \min\{\Phi(\lambda^-), \Phi(\lambda^+)\} - \min\{\Phi(\lambda^+) - \gamma^+, \Phi(\lambda^-) - \gamma^-\}$. We can always construct examples for which $\Phi^*[\varphi, I] = \min\{\Phi(\lambda^+) - \gamma^+, \Phi(\lambda^-) - \gamma^-\}$, and therefore we can assert that the bound in Theorem 5.2 is the best possible. Namely, suppose that $\varepsilon = 12$; $\Phi(\lambda^-) = 16$, $\gamma^- = 9$, $\Phi^*[\varphi, I - k] = 7$, $\Phi(\lambda^+) = 15$, $\gamma^+ = 8$, and $\Phi^*[\varphi + k, I] = 10$. Then, $\Phi(\lambda^-) - \Phi^*[\varphi, I - k] = 16 - 7 = 9 \leq \gamma^- = 9 \leq \varepsilon$, and $\Phi(\lambda^+) - \Phi^*[\varphi + k, I] = 15 - 10 = 5 \leq \gamma^+ \leq \varepsilon$. Moreover, $\Phi^*[\varphi, I] = 7 = \min\{16 - 9, 15 - 8\}$ and $\gamma = \min\{16, 15\} - \min\{16 - 9, 15 - 8\} = 8 < \max\{\gamma^-, \gamma^+\}$.

It is possible to make the DC algorithm more efficient if we fathom subproblems by using lower bounds. For subproblems of the form $\min\{\Phi(\omega) \mid \omega \in [\varphi, I]\} = \Phi^*[\varphi, I]$, the following lemma includes two lower bounds. The lemma is due to Khachaturov[9].

Lemma 5.1 *If $\Phi(\varphi) - \Phi(\varphi + i) \geq 0$ and $\Phi(I) - \Phi(I - i) \geq 0$ for all $i \in I\backslash\varphi$, then $lb_1 = \Phi(\varphi) - \sum_{i \in I\backslash\varphi}[\Phi(\varphi) - \Phi(\varphi + i)] \leq \Phi^*[\varphi, I]$, and $lb_2 = \Phi(I) - \sum_{i \in I\backslash\varphi}[\Phi(I) - \Phi(I - i)] \leq \Phi^*[\varphi, I]$.*

Proof. We give the proof for lb_1. The proof for lb_2 is left to the reader. Let $\alpha \in [\varphi, I]$ be such that $\Phi^*[\varphi, I] = \Phi(\alpha)$. Let $\alpha\backslash\varphi = \{i_1, ..., i_r\}$. From the supermodularity of Φ, we have that $\Phi(\alpha) \geq \Phi(\alpha - i_1) - [\Phi(\varphi) - \Phi(\varphi + i_1)]$ and also that $\Phi(\alpha - i_1) \geq \Phi[(\alpha - i_1) - i_2] - [\Phi(\varphi) - \Phi(\varphi + i_2)]$. By substituting the last inequality into the previous inequality, we obtain $\Phi(\alpha) \geq \Phi((\alpha - i_1) - i_2) - [\Phi(\varphi) - \Phi(\varphi + i_1)] - [\Phi(\varphi) - \Phi(\varphi + i_2)]$. Applying this procedure $r - 1$ times, we obtain

$$\Phi(\alpha) \geq \Phi(\varphi) - \sum_{i \in \alpha\backslash\varphi}[\Phi(\varphi) - \Phi(\varphi + i)].$$

Since $\Phi(\varphi) - \Phi(\varphi + i) \geq 0$ for all $i \in I\backslash\varphi$ we have that

$$lb_1 = \Phi(\varphi) - \sum_{i \in I\backslash\varphi}[\Phi(\varphi) - \Phi(\varphi + i)] \leq \Phi(\varphi) - \sum_{i \in \alpha\backslash\varphi}[\Phi(\varphi) - \Phi(\varphi + i)] \leq \Phi(\alpha).$$

□

How can we incorporate such a lower bound into the DC algorithm? During the running of the DC program we keep in a global variable β the subset of I^0 that has the lowest function value found sofar. Then we can include a Step 3a after Step 3 in the Procedure DC().

```
Step 3a: Calculate $lb := \max\{lb_1, lb_2\}$;
         **if** $\Phi(\beta) - lb \leq \varepsilon$
         **then begin** $\lambda := \beta$; $\gamma := \Phi(\beta) - lb$;
                   **goto** Step 7;
              **end**
```

It is obvious that λ and γ satisfy $\Phi(\lambda) - \Phi^*[\varphi, I] \leq \Phi(\beta) - lb = \gamma \leq \varepsilon$. Note that in this case, β is, in general, not an element of the interval $[\varphi, I]$.

The DC algorithm can also be used as a fast greedy heuristic. If the prescribed accuracy ε_0 is taken very large, it will never happen that a branch is made in Step

6; the interval $[\varphi, I]$ is halved in every recursive call of the algorithm until $\varphi = I$, and a 'greedy' solution is found. Moreover, the calculated accuracy γ gives insight into the quality of the found solution; it is an upper bound for the difference of the found solution and an optimal solution, which is in general better than the one given in Minoux[12].

6. The Simple Plant Location Problem; an illustration

The DC algorithm is used for determining a global minimum (0-minimum) and a 2-minimum for the SPL problem of which the data are presented by Table 6.1. This example is borrowed from Boffey[4].

Location		Delivery cost to cite				
i	r_i	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
1	7	7	15	10	7	10
2	3	10	17	4	11	22
3	3	16	7	6	18	14
4	6	11	7	6	12	8

Table 6.1: The data of the SPL problem

For solving the SPL problem it suffices to solve the problem $\min\{P(\omega) \mid \omega \in [\emptyset, I^0]\} = P^*[\emptyset, I^0] = P(\alpha)$ with

$$P(\omega) = \sum_{i\in\omega} r_i + \sum_{j=1}^{n} \min_{i\in\omega} c_{ij}, \;\; I^0 = \{1, 2, 3, 4\}, \;\; n = 5.$$

As usual, in the SPL problem r_i is the fixed cost of opening a plant on location i, and c_{ij} is the cost of satisfying the demand of customer j by plant i. The recursive trees of the solutions for the cases $\varepsilon_0 = 0$ and $\varepsilon_0 = 2$ are depicted in Figure 6.1 and Figure 6.2, respectively. Each subproblem is represented by a box in which the input parameters and the output values are shown. At each arc of the trees the corresponding steps of the Procedure DC() are indicated. The prescribed accuracy $\varepsilon_0 = 0$ cannot be satisfied at the second level of the tree. Hence, the tree is branched. In the case of $\varepsilon_0 = 2$ the DC algorithm applies the branching rule at the third level because after the second one the value of the current accuracy is equal to 1 ($\varepsilon = 1$).

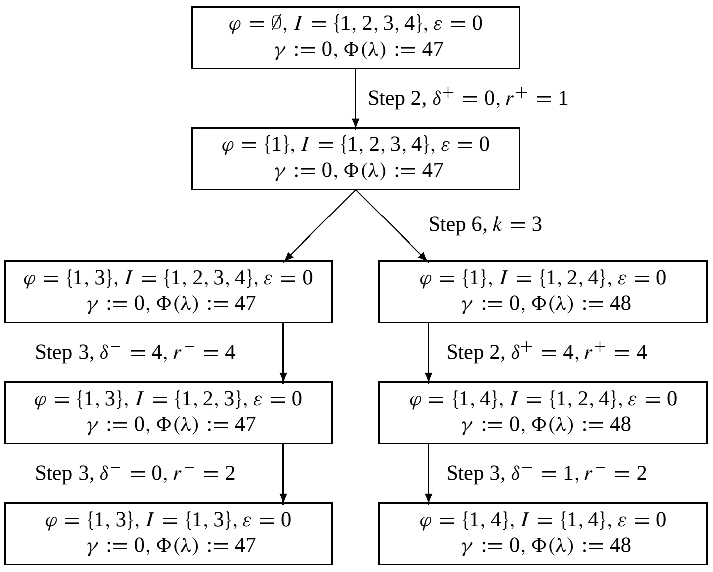
$\varphi = \emptyset, I = \{1,2,3,4\}, \varepsilon = 0$
$\gamma := 0, \Phi(\lambda) := 47$
Step 2, $\delta^+ = 0, r^+ = 1$
$\varphi = \{1\}, I = \{1,2,3,4\}, \varepsilon = 0$
$\gamma := 0, \Phi(\lambda) := 47$
Step 6, $k = 3$
$\varphi = \{1,3\}, I = \{1,2,3,4\}, \varepsilon = 0$
$\gamma := 0, \Phi(\lambda) := 47$
$\varphi = \{1\}, I = \{1,2,4\}, \varepsilon = 0$
$\gamma := 0, \Phi(\lambda) := 48$
Step 3, $\delta^- = 4, r^- = 4$
Step 2, $\delta^+ = 4, r^+ = 4$
$\varphi = \{1,3\}, I = \{1,2,3\}, \varepsilon = 0$
$\gamma := 0, \Phi(\lambda) := 47$
$\varphi = \{1,4\}, I = \{1,2,4\}, \varepsilon = 0$
$\gamma := 0, \Phi(\lambda) := 48$
Step 3, $\delta^- = 0, r^- = 2$
Step 3, $\delta^- = 1, r^- = 2$
$\varphi = \{1,3\}, I = \{1,3\}, \varepsilon = 0$
$\gamma := 0, \Phi(\lambda) := 47$
$\varphi = \{1,4\}, I = \{1,4\}, \varepsilon = 0$
$\gamma := 0, \Phi(\lambda) := 48$

Figure 6.1: The recursive tree of the SPL problem with $\varepsilon_0 = 0$

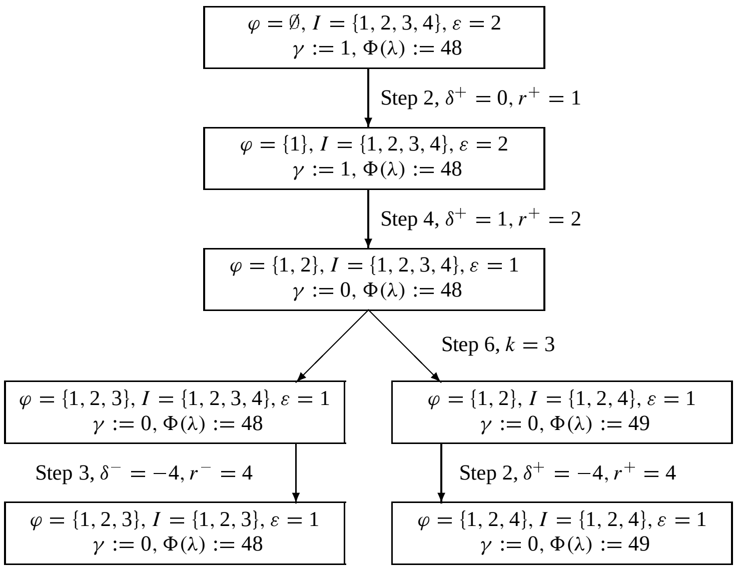
$\varphi = \emptyset, I = \{1,2,3,4\}, \varepsilon = 2$
$\gamma := 1, \Phi(\lambda) := 48$
Step 2, $\delta^+ = 0, r^+ = 1$
$\varphi = \{1\}, I = \{1,2,3,4\}, \varepsilon = 2$
$\gamma := 1, \Phi(\lambda) := 48$
Step 4, $\delta^+ = 1, r^+ = 2$
$\varphi = \{1,2\}, I = \{1,2,3,4\}, \varepsilon = 1$
$\gamma := 0, \Phi(\lambda) := 48$
Step 6, $k = 3$
$\varphi = \{1,2,3\}, I = \{1,2,3,4\}, \varepsilon = 1$
$\gamma := 0, \Phi(\lambda) := 48$
$\varphi = \{1,2\}, I = \{1,2,4\}, \varepsilon = 1$
$\gamma := 0, \Phi(\lambda) := 49$
Step 3, $\delta^- = -4, r^- = 4$
Step 2, $\delta^+ = -4, r^+ = 4$
$\varphi = \{1,2,3\}, I = \{1,2,3\}, \varepsilon = 1$
$\gamma := 0, \Phi(\lambda) := 48$
$\varphi = \{1,2,4\}, I = \{1,2,4\}, \varepsilon = 1$
$\gamma := 0, \Phi(\lambda) := 49$

Figure 6.2: The recursive tree of the SPL problem with $\varepsilon_0 = 2$

7. The Quadratic Cost Partition Problem; computational experiments

For given real numbers $p_i, i \in I^0$ and nonnegative real numbers q_{ij} with $i, j \in I^0$, the Quadratic Cost Partition(QCP) problem is the problem of finding a subset $T \subseteq I^0$ such that the weight

$$\sum_{i\in T} p_i - \frac{1}{2}\sum_{i,\, j\in T} q_{ij}$$

is as large as possible. If I^0 is the vertex set, E is the edge set of an edge-weighted graph $G = (I^0, E)$, $E \subseteq I^0 \times I^0$ and $w_{ij} \geq 0$ are the edge weights, then define a cut $\delta(T)$ as the edge set of which for each edge one end is in T and the other in $I^0 \backslash T$. The Max-Cut problem is a QCP problem with

$$p_i = \sum_{j\in I^0} w_{ij} \text{ and } q_{ij} = 2w_{ij}.$$

We have used randomly generated connected graphs; the number of vertices varies from 40 to 80 with the density d varying from 10% to 100% (i.e. $d \in [0.1 - 1.0]$). The values of p and q are uniformly distributed in the intervals [0,100] and [1,100], respectively. The computational results are summarized in Table 7.1. We have tested the DC algorithm on the QCP test problems from Lee et al.[10], and have made a comparison between our results and those from Lee et al.

Each problem set is labeled by the number of vertices of the graph together with their densities d. For example, problem 50/7 refers to graphs with 50 vertices and density $d = 0.7$(or 70%), problem 40 refers to complete graphs with 40 vertices. For each combination of density and number of vertices, five random problems were solved. The column 'Lee et al.' of Table 7.1 contains the average computational times for the problems on a RISC 6000 workstation as given in Lee et al.,[10]. The DC algorithm was coded by means of Turbo Pascal 6.0 and was executed on a PC with a processor with 133 Mhz. Cells with "min", "avg", and "max" in Table 7.1 shows minimum, maximum and average performances of two statistics for the DC algorith: 'the number of subproblems' solved and 'the number of discarded subproblems' indicating the number of subproblems discarded by means of the lower bounds lb_1 and lb_2 from Lemma 5.1.

When the graph has at most 40 vertices, the problem is very easy, and the calculation times are smaller than 0.05 sec. For problems with at least 40 vertices the average calculation times grow exponentially with decreasing values of the density d (see

Prob.	Time average, sec		# of generated subpr.			# of fathomed subpr.		
	Lee et al.	DC	min	avg	max	min	avg	max
40/2	0.97	0.10	618	797	972	306	396	481
40/3	2.09	0.08	470	640	793	235	313	385
40/4	6.79	0.05	430	539	735	204	258	354
40/5	6.63	0.028	428	497	584	201	231	278
40/6	8.62	0.038	340	387	434	153	173	192
40/7	11.40	0.030	204	216	267	85	100	116
40/8	14.57	0.028	217	261	292	80	95	103
40/9	8.46	0.012	107	154	223	34	42	56
40	13.89	0.004	119	160	213	33	38	48
50/1	0.56	0.29	1354	1885	2525	686	945	1258
50/2	5.36	0.45	2100	2778	3919	1042	1393	1971
50/3	16.19	0.27	1671	2074	2565	814	1019	1268
50/4	95.32	0.18	1183	1576	1976	576	755	950
50/5	38.65	0.08	870	943	1051	414	447	502
50/6	43.01	0.07	646	725	798	291	321	345
50/7	48.07	0.05	610	648	714	245	270	294
60/2	12.11	1.56	5470	8635	11527	2718	4303	5744
60/3	183.02	0.71	3481	5069	7005	1736	2519	3478
60/4	150.50	0.39	2450	3037	3895	1221	1503	1917
60/5	137.22	0.22	1701	2080	2532	825	1012	1236
70/2	437.74	4.89	15823	23953	34998	7909	11971	17486
70/3	367.50	1.91	9559	11105	13968	4769	5540	6967
80/1	20.87	28.12	55517	92836	132447	27771	46418	66228
80/2	864.27	17.10	64261	66460	68372	32102	33202	34160

Table 7.1: The comparison of computational results

Figure 7.1) for all values of ε_0. This behavior differs from the results of the algorithm of Lee et al; their calculation times grow with increasing densities. This behavior is common for all $m > 40$. For problems with density more than 10% our algorithm is faster than the algorithm of Lee et al. For one problem(80.1) with density equal to 10% our algorithm uses more time.

Some typical properties of the behavior of the DC algorithm are shown in Figures 7.1, 7.2 and 7.3.

In Figure 7.2 can be seen that the calculation time of the DC algorithm grows exponentially when the number of vertices increases. This is to be expected since the QCP problem is NP-hard. In Figure 7.3 it is shown how the calculation times of the DC algorithm depend on the value of ε_0. We have used different prescribed accuracies varying from 0% to 5% of the value of the global minimum.

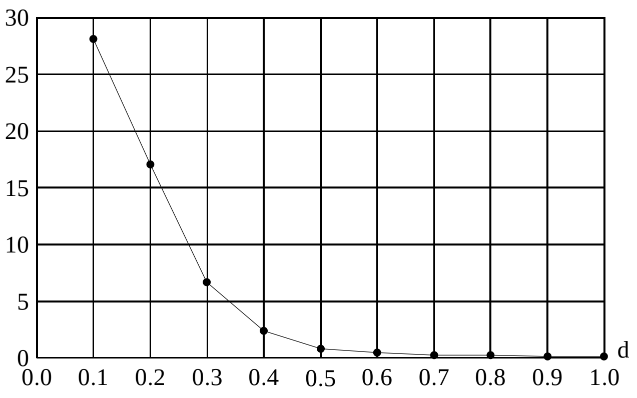

Figure 7.1: Average calculation time against the density d (case: $m = 80$, $\varepsilon_0 = 0$)

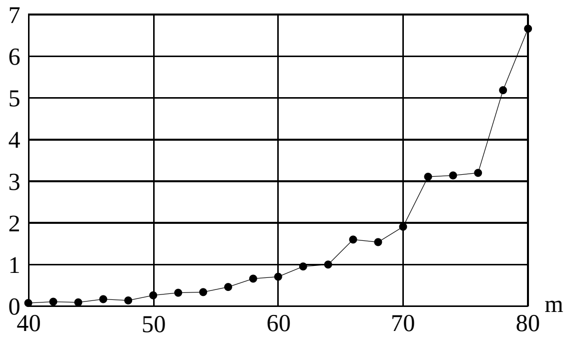

Figure 7.2: Average calculation time against the number of vertices m (case: $d = 0.3$, $\varepsilon_0 = 0$)

In all experiments with $\varepsilon_0 > 0$ the maximum of the value of the calculated γ (denoted by γ_{max}) is at most $0.01949 \times \Phi^*[\emptyset, I^0]$. Moreover, for all tested problems with density at least 30%, γ_{max} is zero. Hence, for all tested problems with $d \geq 0.3$ we found an exact global minimum with a calculation time of at most 5 sec. In Figure 7.4, γ_{max} is

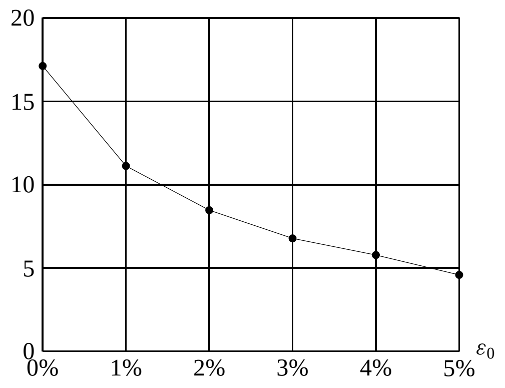

Figure 7.3: Average time against the prescribed accuracy ε_0 (case: $m = 80$, $d = 0.2$)

depicted for the various values of ε_0.

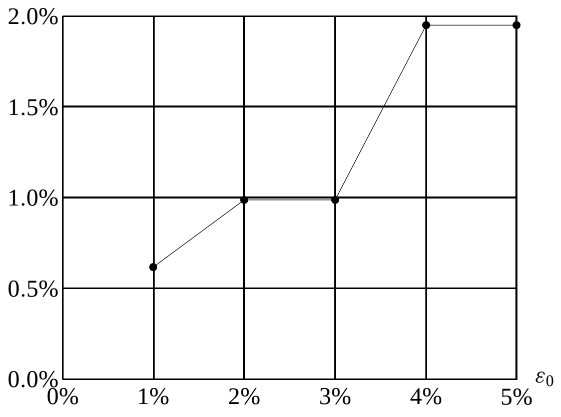

Figure 7.4: γ_{max} as percentage of the value of a global minimum

8. Conclusions

Theorem 2.1 can be considered as the base of our Data Correcting algorithm. It states that if an interval $[\varphi, I]$ is split into $[\varphi, I - k]$ and $[\varphi + k, I]$, then if we know the difference between the supermodular function values $\Phi(\varphi)$ and $\Phi(\varphi + k)$, or between $\Phi(I)$ and $\Phi(I-k)$, then this difference is an upper bound for the difference of the (unknown!) optimal values on the two subintervals. This difference is used for 'correcting' the current data (values of Φ) in the algorithm.

Another keystone in the paper is Theorem 5.2. For any two subsets S^+ and S^- that cover the feasible region, say, $S = S^+ \cup S^-$, it enables us to derive a new (and sharper than the usual one), upper bound between an upper bound of the optimal value of the objective function on either S^+ or S^- and the optimal value of the function on S. This new and sharper upper bound, when implemented in the DC algorithm, yields a decrease of the calculated accuracy. Moreover, this bound can also be build into other branch and bound type algorithms for reducing the calculated accuracy.

The DC algorithm presented in this paper is a recursive branch and bound type algorithm. This recursion makes it possible to present a short proof of its correctness; see Theorem 5.1.

We have tested the DC algorithm for the case of QCP problems, and used the data from Lee et al[10]. The main striking computational result is the ability of the DC algorithm to find exact solutions for QCP problems with densities larger than 30% and with prescribed accuracies of five percent within fractions of second. For example, an exact global optimum of the QCP problem with 80 vertices and 100% density, was found within 0.22 sec on a PC with a 133 Mhz processor. We point out that when ε_0 is taken very large, the DC algorithm behaves as a greedy algorithm.

Finally, we would like to remark that the DC algorithm can be used for general classes of combinatorial optimization problems that are reducible to the supermodular minimization problem.

References

[1] Dj.A. Babayev, "Comments on the note of Frieze," *Mathematical Programming* 7 (1974) 249–252.

[2] F. Barahona, M. Grötschel, M. Jūnger, and G. Reinelt, "An application of combinatorial optimization to statistical physics and circuit layout design," *Operations Research* 36 (1988) 493–512.

[3] J.E. Beasley, "Lagrangean heuristics for location problems," *European Journal of Operational Research* 65 (1993) 383–399.

[4] T.B. Boffey, *Graph Theory in Operations Research*, Academic Press, London (1982).

[5] V.P. Cherenin, "Solving some combinatorial problems of optimal planning by the method of successive calculations," in: *Proceedings of the Conference on*

Experiences and Perspectives of the Application of Mathematical Methods and Electronic Computers in Planning (Mimeograph, Novosibirsk (1962), (in Russian).

[6] B. Goldengorin, “On the stability of solutions in problems with a single connected component of local minima,” in: *Models and Methods of Solving Problems of Economic Systems Interaction*, Nauka, Novosibirsk (1982) pp. 149–160, (in Russian).

[7] B. Goldengorin, “A correcting algorithm for solving some discrete optimization problems,” *Soviet Mathematical Doklady* 27 (1983) 620–623.

[8] B. Goldengorin, *Requirements of Standards: Optimization Models and Algorithms*, Russian Operations Research, Hoogezand, The Netherlands (1995).

[9] V.R. Khachaturov, “Some problems of the consecutive calculation method and its applications to location problems,” Ph.D thesis, Central Economics & Mathematics Institute, Russian Academy of Sciences, Moscow (1968) (in Russian).

[10] H. Lee, G.L. Nemhauser, and Y. Wang, “Maximizing a submodular function by integer programming: Polyhedral results for the quadratic case,” *European Journal of Operational Research* 94 (1996) 154–166.

[11] L. Lovasz, “Submodular functions and convexity,” in: A. Bachem et al. (Eds.), *Mathematical Programming: The State of the Art* (1983) 235–257.

[12] M. Minoux, “Accelerated greedy algorithms for maximizing submodular set functions,” in: Actes Congres IFIP, J. Stoer (Ed.), Springer Verlag, Berlin(1977) 234–243.

[13] G.L. Nemhauser, L.A. Wolsey, and M.L. Fisher, “An analysis of approximations for maximizing submodular set functions-I,” *Mathematical Programming* 14 (1978) 265–294.

[14] A. Petrov and V. Cherenin, “An improvement of Train Gathering Plans Design Methods,” *Zheleznodorozhnyi Transport* 3 (1948) 60–71 (in Russian).

Groningen, February 1998.