A Multilevel Search Algorithm for the Maximization of Submodular Functions

Boris Goldengorin[1]
Diptesh Ghosh[2]

SOM-theme A Primary Processes within Firms

Abstract

Maximization of submodular functions on a ground set is a $\mathcal{NP}$-hard combinatorial optimization problem. Data correcting algorithms are among the several algorithms suggested for solving this problem exactly and approximately. From the point of view of Hasse diagrams data correcting algorithms use information belonging to only one level in the Hasse diagram adjacent to the level of the solution at hand. In this paper, we propose a data correcting approach that looks at multiple levels of the Hasse diagram and hence makes the data correcting algorithm more efficient. Our computations with quadratic cost partition problems show that this multilevel search effects a eight to ten fold reduction in computation times, so that some of the dense quadratic partition problem instances of size 500, currently considered as some of the most difficult problems and far beyond the capabilities of current exact methods, are solvable on a personal computer working at 300 MHz within ten minutes.
Keywords: Data Correcting, Hasse Diagram, Multilevel Search, Quadratic Cost Partition Problem

[1]Corresponding author. Faculty of Economic Sciences, University of Groningen, The Netherlands. Email: b.goldengorin@eco.rug.nl

[2]P&QM Area, Indian Institute of Management, Ahmedabad, India. Email:diptesh@iimahd.ernet.in

1 Introduction

Let $N = \{1, 2, \ldots, n\}$ and 2^N denote the set of all subsets of N. A function $z : 2^N \rightarrow \Re$ is called *submodular* if for each $I, J \in 2^N$, $z(I) + z(J) \geq z(I \cup J) + z(I \cap J)$. The solution process of many classical combinatorial optimization problems, like the simple plant location problem, the generalized transportation problem, the quadratic cost partition (QCP) problem with nonnegative edge weights, and set covering, involve the maximization of a submodular (or, equivalently, minimization of a supermodular) function, i.e. the problem: $\max\{z(I)|I \subseteq N\}$.

Although the general problem of the maximization of a submodular function is known to be NP-hard, there has been 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 submodularity of the underlying objective function has been only implicit to the analysis. For example, [2] addressed the max-cut problem and developed a branch and cut algorithm, [1] applied Lagrangean heuristics to several classes of location problems including SPL problems, [12] studied the quadratic cost partition problem (QCP) of which max-cut with nonnegative edge weights is a special case, and [6] reported their computational experiments for binary quadratic programs (BQP) with adaptive memory tabu search procedures. [4] and [9] applied a data correcting algorithm to the problem of minimization (maximization) a supermodular (submodular) function by which we can solve to optimality, uncapacitated competitive location problem instances of size 50, and QCP instances on dense graphs up to 100 vertices, respectively.

The purpose of this paper is to improve on the data correcting algorithm proposed in [9]. Data correcting algorithms work by "correcting" the data of a given problem instance to obtain a new problem instance which is polynomially solvable. In case such an instance is not available, the algorithm uses a branching rule to decompose the problem into subproblems and then looks at each of the subproblems individually. The choice of the branching variable in the algorithm in [9] is based on an examination of solutions that are one level deeper in the Hasse diagram from the current solution. The algorithm that we propose here searches more than one level deep in the Hasse diagram. The tradeoff of course is between the effort of searching deeper in the Hasse diagram and the number of nodes pruned by the algorithm as a result of searching deeper in the Hasse diagram.

In the next section in this paper, we describe the data correcting algorithm that we propose in this paper. We first introduce an algorithm called the preliminary preservation algorithm of order r. This is an extension of the PP algorithm presented in [9], which looks at r levels of the Hasse diagram instead of only one level. This algorithm is used to define the class of polynomially solvable instances, i.e. instances for which our algorithm returns either an optimal solution or its approximation bounded by the prescribed value of an accuracy parameter. Section 3 reports our computational experience with our data correcting algorithm for Quadratic Cost Partition Problems. Section 4 concludes the paper.

2 The data correcting algorithm based on multilevel search

In this section we first describe a class of algorithmically defined polynomially solvable instances for submodular function maximization problems. We then describe a data correcting algorithm that uses the class of polynomially solvable instances to solve a general submodular function maximization problem.

The class of polynomially solvable instances that we describe here is an algorithmic class, i.e. they are defined using a polynomial time algorithm called the Preliminary Preservation Algorithm of order r (PPAr). Normally this algorithm terminates with a subgraph of the Hasse diagram of the original instance which is guaranteed to contain the maximum. However, for instances where PPAr returns a subgraph with a single node, that node *is* the maximum, and the instance is solved in polynomial time. Instances such as these make up the class of polynomially solvable instances that we consider here.

Let z be a real-valued function defined on the power set 2^N of $N = \{1, 2, \ldots, n\}; n \geq 1$. For each $S, T \in 2^N$ with $S \subseteq T$, we define

$$[S, T] = \{I \in 2^N \mid S \subseteq I \subseteq T\}.$$

Note that $[\emptyset, N] = 2^N$. Any *interval* $[S, T]$ is a subinterval of $[\emptyset, N]$ if $\emptyset \subseteq S \subseteq T \subseteq N$. We denote this using the notation $[S, T] \subseteq [\emptyset, N]$. In this paper an interval is always a subinterval of $[\emptyset, N]$. Throughout this paper, it is assumed that z attains a finite maximum value on $[\emptyset, N]$ which is denoted by $z^*[\emptyset, N]$, and $z^*[S, T] = \max\{z(I) : I \in [S, T]\}$ for any $[S, T] \subseteq [\emptyset, N]$. We also define

$$\begin{aligned} d_k^+(I) &= z(I + k) - z(I), \text{ and} \\ d_k^-(I) &= z(I - k) - z(I). \end{aligned}$$

The following theorem and corollaries from [9] act as a basis for the Preliminary Preservation (PP) algorithm described therein.

Theorem 1 *Let z be a submodular function on $[S, T] \subseteq [\emptyset, N]$ and let $k \in T \backslash S$. Then the following assertions hold.*

(a) $z^*[S + k, T] - z^*[S, T - k] \leq z(S + k) - z(S)) = d_k^+(S) = d_k^+$,

(b) $z^*[S, T - k] - z^*[S + k, T] \leq z(T - k) - z(T) = d_k^-(T) = d_k^-$.

Corollary 2 *(Preservation rules of order zero). Let z be a submodular function on $[S, T] \subseteq [\emptyset, N]$ and let $k \in T \backslash S$. Then the following assertions hold.*

(a) First Preservation Rule: If $d_k^+(S) \leq 0$, then $z^*[S, T] = z^*[S, T - k] \geq z^*[S + k, T]$.

(b) Second Preservation Rule: If $d_k^-(T) \leq 0$*, then* $z^*[S,T] = z^*[S+k,T] \geq z^*[S,T-k]$.

The PP algorithm accepts an interval $[S,T]$, $S \subseteq T$ and tries to apply Corollary 2 repeatedly. It returns an interval $[X,Y]$, $S \subseteq X \subseteq Y \subseteq T$, such that $z^*[S,T] = z^*[X,Y]$. In this paper we describe a call to the PP algorithm by $PP([S,T])$.

The preservation rules mentioned in Corollary 2 look at a level which is exactly one level deeper in the Hasse diagram than the levels of S and T. However, instead of looking one level deep we may look r levels deep in order to determine whether we can include or exclude an element. Let

$$\begin{aligned} M_r^+[S,T] &= \{I \in [S,T] : |I \backslash S| \leq r\}, \\ M_r^-[S,T] &= \{I \in [S,T] : |T \backslash I| \leq r\}. \end{aligned}$$

The set $M_r^+[S,T]$ is a collection of all sets representing solutions containing more elements than S, and which are no more than r levels deeper than S in the Hasse diagram. Similarly, the set $M_r^-[S,T]$ is a collection of all sets representing solutions containing less elements than T, and which are no more than r levels deeper than T in the Hasse diagram. Let us further define the collections of sets

$$\begin{aligned} N_r^+[S,T] &= M_r^+[S,T] \backslash M_{r-1}^+[S,T], \\ N_r^-[S,T] &= M_r^-[S,T] \backslash M_{r-1}^-[S,T]. \end{aligned}$$

The sets $N_r^+[S,T]$ and $N_r^-[S,T]$ are the collection of sets which are located exactly r levels above S and below T in the Hasse diagram, respectively.

Further, let $v_r^+[S,T] = \max\{z(I) : I \in M_r^+[S,T]\}$, $v_r^-[S,T] = \max\{z(I) : I \in M_r^-[S,T]\}$, $w_{rk}^+[S,T] = \max\{d_t^+(I) : I \in N_r^+[S+k,T]\}$ and $w_{rk}^-[S,T] = \max\{d_t^-(I) : I \in N_r^-[S,T-k]\}$.

Theorem 3 *Let z be a submodular function on $[S,T] \subseteq [\emptyset, N]$ with $k \in T \backslash S$ and let r be a positive integer. Then the following assertions hold.*

(a) If $|N_r^+[S+k,T]| > 0$*, then* $z^*[S+k,T] - \max\{z^*[S,T-k], v_r^+[S,T]\} \leq \max\{w_{rk}^+[S,T], 0\}$.

(b) If $|N_r^-[S,T-k]| > 0$*, then* $z^*[S,T-k] - \max\{z^*[S+k,T], v_r^-[S,T]\} \leq \max\{w_{rk}^-[S,T], 0\}$.

Proof: We prove only part (a) since the proof of the part (b) is similar. We may represent the partition of interval $[S,T]$ as follows:

$$[S,T] = M_r^+[S,T] \cup \bigcup_{I \in N_r^+[S,T]} [I,T].$$

Using this representation on the interval $[S+k, T]$, we have $z^*[S+k, T] = \max\{v_r^+[S+k, T], \max\{z^*[I+k, T] : I \in N_r^+[S, T]\}\}$. Let $I(k) \in \arg\max\{z^*[I+k, T] : I \in N_r^+[S, T]\}$.

There are two cases to consider: $z^*[I(k)+k, T] \geq v_r^+[S+k, T]$, and $z^*[I(k)+k, T] < v_r^+[S+k, T]$.

In the first case $z^*[S+k, T] = z^*[I(k)+k, T]$. For $I(k) \in N_r^+[S, T]$ we can apply Theorem 1(a) on the interval $[I(k), T]$ to obtain $z^*[I(k)+k, T] - z^*[I(k), T-k] \leq d_k^+(I(k))$, so that in this case $z^*[S+k, T] - z^*[I(k), T-k] \leq d_k^+(I(k))$. Note that for $[I(k), T-k] \subseteq [S, T-k]$ we have $z^*[S, T-k] \geq z^*[I(k), T-k]$, which implies that $z^*[S+k, T] - z^*[S, T-k] \leq d_k^+(I(k))$. Adding two maximum operations we get

$$z^*[S+k, T] - \max\{z^*[S, T-k], v_r^+[S+k, T]\} \leq \max\{d_k^+(I(k)), 0\}.$$

Since $w_{rk}^+[S, T]$ is the maximum of $d_k^+(I)$ for $I \in N_r^+[S+k, T]$, we have the required result.

In the second case $z^*[S+k, T] = v_r^+[S+k, T]$ which implies that $z^*[S+k, T] - v_r^+[S+k, T]\} = 0$ or $z^*[S+k, T] - \max\{z^*[S, T-k], v_r^+[S+k, T]\} \leq 0$. Adding a maximum operation with $w_{rk}^+[S, T]$ completes the proof of case (a). □

Corollary 4 *(Preservation rules of order r). Let z be a submodular function on* $[S, T] \subseteq [\emptyset, N]$ *and let* $k \in T \backslash S$. *Then the following assertions hold.*

(a) First Preservation Rule of Order r: If $w_{rk}^+[S, T] \leq 0$, *then* $z^*[S, T] = \max\{z^*[S, T-k], v_r^+[S+k, T]\} \geq z^*[S+k, T]$.

(b) Second Preservation Rule of Order r: If $w_{rk}^-[S, T] \leq 0$, *then* $z^*[S, T] = \max\{z^*[S+k, T], v_r^-[S, T-k]\} \geq z^*[S, T-k]$

Notice that when we apply Corollary 2 to an interval, we get a reduced interval, however, when we apply Corollary 4, we get a value v_r in addition to a reduced interval.

It can be proved by induction that the portion of the Hasse diagram eliminated by preservation rules of order $r-1$ while searching for a maximum of the submodular function will certainly be eliminated by preservation rules of order r. In this sense, preservation rules of order r are not weaker than preservation rules of order $r-1$. (A detailed proof for the result that preservation rules of order 1 are not weaker than preservation rules of order 0, refer to [10]).

In order to apply Corollary 4, we need functions that compute the values of $w_{rk}^+[S, T]$, $w_{rk}^-[S, T]$ $v_r^+[S+k, T]$, and $v_r^-[S, T-k]$. To that end, we define two recursive functions, *PPArplus* to compute $w_{rk}^+[S, T]$ and $v_r^+[S+k, T]$, and *PPArminus* to compute $w_{rk}^-[S, T]$ and $v_r^-[S, T-k]$. The pseudocode for *PPArplus* is shown below. Its output is a 3-tuple, containing, in order, $w_{rk}^+[S, T]$ and $v_r^+[S+k, T]$, and a solution in $M_r^+[S+k, T]$ whose

objective function value is $\nu_r^+[S+k,T]$. The pseudocode for *PPArminus* can be constructed in a similar manner.

---

function PPArplus$([S,T],r,k)$
begin
    $w \leftarrow -\infty$; $\nu \leftarrow -\infty$; $\nu set \leftarrow \emptyset$;
    $(w,\nu,\nu set) \leftarrow$ *IntPPArPlus*$([S+k,T],r,w,\nu,\nu set)$;
    return $(w,\nu,\nu set)$;
end;

function IntPPArplus$([X,Y],r,w,\nu,\nu set)$
begin
    for each $t \in Y \setminus X$ *do begin*
        if $z(X+t) > \nu$ *then begin*
            $\nu \leftarrow z(X+t)$;
            $\nu set \leftarrow (X+t)$;
        end;
        if $d_t^+(X+t) > w$ *then* $w \leftarrow d_t^+(X+t)$;
        if $d_t^+(X+t) > 0$ *and* $r > 1$ *then*
            $(w,\nu,\nu set) \leftarrow$ *IntPPArPlus*$([X+t,Y],r-1,w,\nu,\nu set)$;
    end;
    return $(w,\nu,\nu set)$;
end;

Note that *PPArplus* and *PPArminus* are both $\mathcal{O}(n\binom{n}{r})$, i.e. polynomial for a fixed value of r. However, in general, they are not polynomial in r.

We now use *PPArplus* and *PPArminus* to describe the Preliminary Preservation Algorithm of order r (PPAr). Note that if $r = 0$ we obtain the PP algorithm which is the Preliminary Preservation Algorithm of order 0. Given a submodular function z on $[X,Y] \subseteq [\emptyset,N]$, PPAr outputs a subinterval $[S,T]$ of $[X,Y]$ and a set B such that $z^*[X,Y] = \max\{z^*[S,T], z(B)\}$ and $\min\{w_{rk}^+[S,T],\ w_{rk}^-[S,T]\} > 0$ for all $k \in T \setminus S$. At iteration i, the algorithm, when the search has been restricted to $[S_i,T_i]$, starts by applying the PP algorithm (from [9]) to this interval and reducing it to $[S_i',T_i']$. If $|T_i' \setminus S_i'| > 0$, an element $k \in T_i' \setminus S_i'$ is chosen, and the algorithm tries to apply Corollary 4(a) to decide whether it belongs to the set that maximizes $z(\cdot)$ over $[S_i,T_i]$ or not. If it does, then the search is restricted to the interval $[S_i'+k,T_i']$. Otherwise, the search tries to apply Corollary 4(b) to decide whether the interval can be reduced to $[S_i',T_i'-k]$.

Algorithm PPAr$([S,T],r)$
begin
$X \leftarrow S$, $Y \leftarrow T$; $B \leftarrow \arg\max\{z(S), z(T)\}$;
while $Y \neq X$ *do begin*
$[S_i, T_i] \leftarrow PP([X,Y])$;
$d^+ \leftarrow \min\{d_k^+(S) | k \in T \backslash S\}$;
$d^- \leftarrow \min\{d_k^-(T) | k \in T \backslash S\}$;
if $d^+ > d^-$ *then begin*
$k^+ \leftarrow \arg\min\{d_t^+(S) | t \in T \backslash S\}$;
$(w, v, vset) \leftarrow$ *PPArplus*$([S_i, T_i], r, k^+)$;
if $v > z(B)$ *then* $B \leftarrow vset$;
if $w \leq 0$ *then* $Y \leftarrow T_i - k^+$;
else return $([S_i, T_i], B)$;
else begin
$k^- \leftarrow \arg\min\{d_t^-(S) | t \in T \backslash S\}$;
$(w, v, vset) \leftarrow$ *PPArminus*$([S_i, T_i], r, k^-)$;
if $v > z(B)$ *then* $B \leftarrow vset$;
if $w \leq 0$ *then* $X \leftarrow S_i + k^-$;
else return $([S_i, T_i], \{w_{ri}^+[S_i, T_i]\}, \{w_{ri}^-[S_i, T_i]\}, B)$;
end;
end;
end;

It is clear that if $r = |T \setminus S|$, PPAr will always find an optimal solution to our problem. However, PPAr is not polynomial in r, and so PPAr with large values of r is not practically useful.

We can embed PPAr in a branch and bound framework to describe DCAr, a data correcting algorithm based on PPAr. It is similar to the DCA proposed in [9]. For DCAr we are given a submodular function z to be maximized over an interval $[S, T]$, and an accuracy parameter ε_o, and we need to find a solution such that the difference between the objective function values of the solution output by DCAr and the optimal solution will not exceed ε_o.

Notice that for a submodular function z, PPAr with a fixed r may terminate with $T \neq S$ and $\min\{w_{ri}^+[S,T], w_{ri}^-[S,T] \mid i \in T \backslash S\} = \omega > 0$. The basic idea behind DCAr is that if this situation occurs, then the data of the current problem is corrected in such a way that ω is non-positive for the corrected function and PPAr can continue. Moreover, each correction of z needs to be carried out in such a way that the corrected function remains submodular. The attempted correction is carried out implicitly, in a manner similar to the one in [9] but

using Corollary 4 instead of Corollary 2. Thus, for example, if $w^+_{rj}[S,T] = \omega \leq \varepsilon_o$, then PPAr is allowed to continue, but the accuracy parameter reduced to $\varepsilon_o - \max\{0, \omega\}$.

If such a correction is not possible, i.e. if ω exceeds the accuracy parameter, then we branch on a variable $k \in \arg\max\{d^+_i(S), d^-_i(T) | i \in T \setminus S\}$ to partition the interval $[S,T]$ into two intervals $[S+k,T]$ and $[S,T-k]$. This branching rule was proposed in [7]. An upper bound for the value of z for each of the two intervals is then computed to see if either of the two can be pruned. We use an upper bound from [11], described as follows. Let $d^+(S,T) = \{d^+_i(S) : d^+_i(S) > 0, i \in T \setminus S\}$ and $d^-(S,T) = \{d^-_i(T) : d^-_i(T) > 0, i \in T \setminus S\}$. Further let $d^+[i]$ (respectively $d^-[i]$) denote the ith largest element of $d^+(S,T)$ (respectively $d^-(S,T)$). Then ub described below is an upper bound to $z^*[S,T]$.

$$ub[S,T] = \max\{\min_{i=1,\ldots,|T\setminus S|}\{z(S) + \sum_{j=1}^{i} d^+[j], z(T) + \sum_{j=1}^{i} d^-[j]\}\}.$$

The following pseudocode describes DCAr formally.

Algorithm DCAr$([S,T], \varepsilon, r)$
begin
    $best\_set \leftarrow \arg\max\{z(S), z(T)\}$;
    $best \leftarrow z(best\_set)$;
    $(best\_set, best) \leftarrow$ *IntDCAr*$([S,T], \varepsilon, r, best\_set, best)$;
    return $best\_set$;
end.

function IntDCAr$([S,T], \varepsilon, r, best\_set, best)$
begin
    $([S,T], \{w^+_{rk}\}, \{w^-_{rk}\}, B] \leftarrow$*PPAr*$([S,T], r)$;
    if $z(B) > best$ *then begin*
        $best\_set \leftarrow B$;
        $best \leftarrow z(B)$;
    end;
    if $S = T$ *return* $(best\_set, best)$;
    $\omega^+ \leftarrow \max\{w^+_{rk}[S,T] | k \in T \setminus S\}$;
    choose k^+ *from* $\min\{k | w^+_{rk}[S,T]| = \omega^+, k \in T \setminus S\}$;
    $\omega^- \leftarrow \max\{w^-_{rk}[S,T] | k \in T \setminus S\}$;
    choose k^- *from* $\min\{k | w^-_{rk}[S,T]| = \omega^-, k \in T \setminus S\}$;
    if $\omega^+ \leq \varepsilon$ *then* (* Correction *)
        IntDCAr$([S + k^+, T], \varepsilon - \max\{0, \omega^+\}, r, best_set, best)$;
    else if $\omega^- \leq \varepsilon$ *then* (* Correction *)
        IntDCAr$([S, T - k^-], \varepsilon - \max\{0, \omega^-\}, r, best_set, best)$;

```
    else begin (* Branching [S, T] → [S + k, T], [S, T − k] *)
        choose k from arg max{d_i^+(S), d_i^−(T) | i ∈ T \ S};
        if ub[S + k, T] > best then begin (* Bounding *)
            (bs_1, b_1) ←IntDCAr([S + k, T], ε, r, best_set, best);
            if b_1 > best then begin
                best_set ← bs_1;
                best ← b_1;
            end;
        end;
        if ub[S, T − k] > best then begin (* Bounding *)
            (bs_2, b_2) ←IntDCAr([S, T − k], ε, r, best_set, best);
            if b_2 > best then begin
                best_set ← bs_2;
                best ← b_2;
            end;
        end;
    end;
end;
```

---

3 Computational experience

In this section we report our computational experience with DCAr. We choose the quadratic cost partition problem as a test bed, since this problem has been earlier used to test the performance of the DCA algorithm in [9]. The quadratic cost partition problem (QCP) can be described as follows (see e.g., [12]). Given nonnegative real numbers q_{ij} and real numbers p_i with $i, j \in N = \{1, 2, \ldots, n\}$, the QCP is the problem of finding a subset $S \subseteq N$ such that the function $z(S) = \sum_{j \in S} p_i - \frac{1}{2} \sum_{i,j \in S} q_{ij}$ will be maximized. The density d of a QCP instance is the ratio of the number of finite q_{ij} values to $n(n-1)/2$, and is expressed as a percentage. It is proved in Theorem 2.2 of [12] that $z(\cdot)$ is submodular.

In [9] computational experiments with QCP have been restricted to instances of size not more than 80, because of comparison purposes with results from [12]. For these instances, it was shown that the average calculation times grow exponentially when the number of vertices increases and reduce exponentially with increasing density.

In this paper we report the performance of DCAr on QCP instances of varying size and densities. The maximum time that we allow for an instance is 10 CPU minutes on an personal computer running on a 300MHz Pentium processor with 64 MB memory. The algorithms have been implemented in Delphi 3.

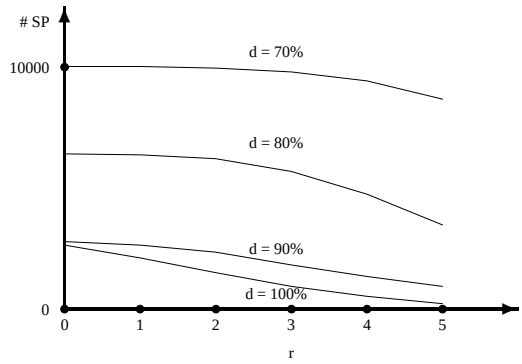

Figure 1: Average number of subproblems generated against r for QCP instances with $n = 100$ and varying d values

The instances we test our algorithms on are statistically similar to the instances in [12]. Instances of size n and density d% are generated as follows. A graph with n nodes and $\frac{d}{100} \times \frac{n(n-1)}{2}$ random edges is generated. The edges are assigned costs from a $\mathcal{U}[1, 100]$ distribution. n edges connect each node to itself, and these edges are assigned costs from a $\mathcal{U}[0, 100]$ distribution. The distance matrix of this graph forms a QCP instance.

We first report the effect of varying the value of r on the performance of DCAr(r). It is intuitive that DCAr(r) will require more execution times when the value of r increases. Our computation experience with 10 QCP instances of size 100 and different densities is shown in Figures 1-3. Figure 1 shows the number of subproblems generated when r is increased from a value of 0 (the DCA in [9]) to 5. As is intuitive, the number of subproblems reduce with increasing r for all density values. Figure 2 shows the execution times of DCAr(r) with varying d and r values. Recall that when the value of r increases, the time required at each subproblem increases, since PPAr requires more computations for larger r values. The decrease in the number of subproblems approximately balance the increase in the time at each subproblem for r values in the range 0 through 4. When $r = 5$, the computation times for DCAr(r) increase significantly for all densities. From Figure 2 it seems that for dense graphs, r values of 3 or 4 are most favorable. This effect also holds for larger instances — Figure 3 shows the execution times for instances of size $n = 200$ and density $d = 100$%.

We next report the results of our experiments to solve large sized QCP instances with DCAr(r). Using results obtained from the previous part of our study, we choose to use DCAr(3) as our algorithm of choice. We consider instances of the QCP with size n ranging from 100 to 500 and densities varying between 10% and 100%. We try to solve these instances exactly ($\varepsilon_o = 0$%), and with a prescribed accuracy $\varepsilon_o = 5$% within 10 minutes.

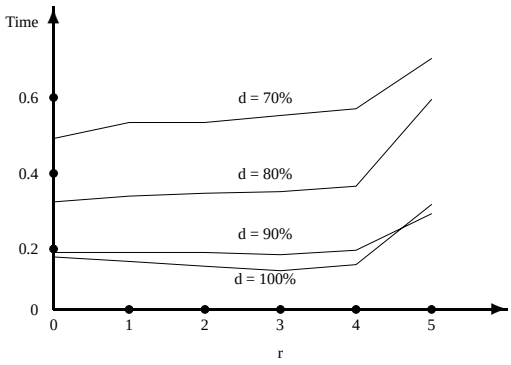

Figure 2: Average execution time (in seconds) against r for QCP instances with $n = 100$ and varying d values

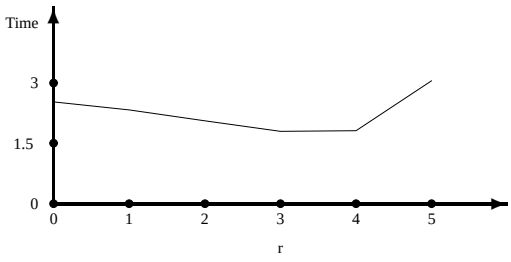

Figure 3: Average execution time (in seconds) against r for QCP instances with $n = 200$ and $d = 100\%$

We report in Table 1 the average execution times in seconds for exact and approximate solutions with DCA(PPA3). The figures in parenthesis report the average execution times for these instances with the DCA from [9]. The entries marked '*' could not be solved within 10 minutes. From the table, we note that the execution times increase exponentially with increasing problem size and decreasing problem densities. Therefore QCP instances with 500 vertices and densities between 90% and 100% are the largest instances which can be solved by the DCAr(3) within 10 minutes on a standard personal computer. We also see that on an average DCAr(3) takes roughly 11% of the time taken by DCA for the exact solutions, and roughly 13% of the time taken by DCA for the approximate solutions. The reduction in time is more pronounced for problems with higher size and higher densities.

Table 1 here

4 Concluding remarks

In this paper we propose a data correcting algorithm DCAr for the class of submodular functions, which extends the DCA algorithm proposed in [9]. It does this by using a preliminary preservation algorithm that looks at multiple levels of the Hasse diagram. Theorem 3, being a generalization of Theorem 1, forms the basis of the DCAr(r) algorithm. Theorem 3a states that if an interval $[S, T]$ is split into $[S, T-k]$ and $[S+k, T]$, then the maximum value of all differences between the submodular function values z at levels $r+1$ and r is an *upper bound* to the difference between the unknown optimal value on the discarded subinterval and the maximum of the unknown value of the preserved subinterval and the maximum value of $z(I)$ on r levels of the Hasse diagram. Theorem 3b can be explained in a similar manner. These upper bounds are used to implicitly "correct" the value of z by correcting the value of the current accuracy.

We have tested the DCAr(r) on the QCP instances which are statistically similar to the instances in [12]. In all the instances tested, the average calculation time increases exponentially with decreasing density values for all prescribed accuracy values. This behavior differs from the results of the branch and cut algorithm in [12], in which calculation times increase when densities increase. This effect is also demonstrated for all algorithms based on linear programming (see, e.g., [3], [14], and [15]). This behavior makes the DCA an algorithm of choice for QCP instances with high densities. Our experiments with different values of r in the PPAr show that for the QCP instances from [12], the best r values are 3 and 4. This effect becomes more pronounced when the density of the corresponding instances approach 100%.

We have used the DCAr(3) to solve QCP instances with up to 500 vertices on dense graphs within 10 minutes using a personal computer with 64 MB RAM operating at 300

MHz. These show an eight to ten-fold improvement on the performance of the DCA in [9]. Since the data-correcting approach is efficient for solving large QCP instances defined on the dense graphs, while branch-and-cut algorithms are efficient for solving large instances on sparse graphs, it will be interesting to investigate hybrids of the two for solving large instances of the QCP for all densities. We plan to implement these in a follow-up to this work. We think that in addition to good hybrid algorithms, each of these algorithms will benefit by using ideas from one into other one.

Acknowledgements The authors would like to thank Gregory Gutin and Marius de Vink for their help in the preparation of this paper.

References

[1] J.E. Beasley. (1993). Lagrangean heuristics for location problems. *European J. Opernl. Res.* vol 65, pp 383–399.

[2] F. Barahona, M. Grötschel, M. Junger, and G. Reinelt. (1988). An application of combinatorial optimization to statistical physics and circuit layout design. *Oper. Res.* vol 36(3), pp 493–512.

[3] F. Barahona, M. Junger, and G. Reinelt. (1989). Experiments in quadratic 0-1 programming. *Math. Prog.* vol 44, pp 127–137.

[4] S. Benati (2003). An improved branch and bound method for the uncapacitated competitive location problem. *Annals of Ops. Res.* vol 122, pp 43–58.

[5] V.P. Cherenin. (1962). 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 (in Russian).

[6] F. Glover, G. A. Kochenberger, and B. Alidaee. (1998). Adaptive memory tabu search for binary quadratic programs. *Man. Sc.* vol 44(3), pp 336–345.

[7] B. Goldengorin. (1983). A correcting algorithm for solving some discrete optimization problems. *Soviet Math. Dokl.* vol 27, pp 620–623.

[8] B. Goldengorin and G. Gutin. (1998). Polynomially solvable cases of the supermodular set function minimization problem. *Research Report TR/6/98*. Department of Mathematics and Statistics, Brunel University, Uxbridge, Middlesex, United Kingdom.

[9] B. Goldengorin, G. Sierksma, G.A. Tijssen, and M. Tso. (1999). The data-correcting algorithm for the minimization of supermodular functions. *Man. Sc.* vol 45(11), pp 1539–1551.

[10] B. Goldengorin. (2002). Data correcting algorithms in combinatorial optimization. *Ph.D. Thesis, SOM Research Institute*. University of Groningen, Groningen, The Netherlands.

[11] V.R. Khachaturov. (1989). *Mathematical methods of regional programming*. Moscow, Nauka, (in Russian).

[12] H. Lee, G.L. Nemhauser, and Y. Wang. (1996). Maximizing a submodular function by integer programming: Polyhedral results for the quadratic case. *European J. Opernl. Res.* vol 94, pp 154–166.

[13] M. Minoux. (1977). Accelerated greedy algorithms for maximizing submodular set functions. *In: J. Stoer (Ed.). Actes Congres IFIP*. Berlin: Springer Verlag, pp 234–243.

[14] P.M. Pardalos and G.P. Rodgers. (1990). Computational aspects of a branch and bound algorithm for quadratic zero-one programming. *Computing* vol 40, pp 131–144.

[15] S. Poljak and F. Rendl. (1995). Solving the max-cut problem using eigenvalues. *Discr. Appl. Math.* vol 62, pp 249–278.

Table 1: Average execution times in seconds for DCAr(3) for QCP instances

Size $\rightarrow$	$n = 100$		$n = 200$		$n = 300$		$n = 400$		$n = 500$	
Density (%) $\downarrow$	$\varepsilon_o = 0\%$	$\varepsilon_o = 5\%$	$\varepsilon_o = 0\%$	$\varepsilon_o = 5\%$	$\varepsilon_o = 0\%$	$\varepsilon_o = 5\%$	$\varepsilon_o = 0\%$	$\varepsilon_o = 5\%$	$\varepsilon_o = 0\%$	$\varepsilon_o = 5\%$
100	0.098	0.094	2.63	2.444	18.316	17.179	85.827	85.096	229.408	222.883
	(0.831)	(0.752)	(64.372)	(38.351)	(340.681)	(229.396)	(1894.811)	(1162.396)	*	*
90	0.138	0.118	3.824	3.607	37.931	34.972	173.063	166.996	624.925	608.755
	(1.027)	(0.916)	(78.614)	(49.926)	(794.037)	(583.754)	(3505.892)	(1996.544)	*	*
80	0.28	0.228	9.506	8.186	98.69	89.685	679.914	580.789	*	*
	(1.784)	(1.108)	(217.898)	(162.455)	(2681.973)	(1875.603)	*	(3604.715)	*	*
70	0.393	0.304	17.643	15.693	413.585	364.48	*	*	*	*
	(2.498)	(1.593)	(631.492)	(376.629)	*	(3165.384)	*	*	*	*
60	0.731	0.517	86.33	72.931	*	*	*	*	*	*
	(3.509)	(2.874)	(1414.103)	(895.426)	*	*	*	*	*	*
50	1.752	1.298	345.723	267.445	*	*	*	*	*	*
	(9.382)	(5.931)	*	(1937.673)	*	*	*	*	*	*
40	3.457	2.179	*	*	*	*	*	*	*	*
	(17.245)	(10.327)	*	*	*	*	*	*	*	*
30	11.032	5.88	*	*	*	*	*	*	*	*
	(48.013)	(22.209)	*	*	*	*	*	*	*	*
20	47.162	17.477	*	*	*	*	*	*	*	*
	(195.82)	(74.841)	*	*	*	*	*	*	*	*
10	70.081	12.196	*	*	*	*	*	*	*	*
	(446.293)	(95.122)	*	*	*	*	*	*	*	*

Note: Figures in parenthesis quote average execution times in seconds for DCA from [9].