**DIMACS Technical Report 92-51**
**March 1993**

# Weighted Multidimensional Search and its Application to Convex Optimization

by

Richa Agarwala and David Fernández-Baca[1,2]
Department of Computer Science
Iowa State University
Ames, Iowa 50011

[1]Visiting Member
[2]Supported in part by the National Science Foundation under grants CCR-8909626 and CCR-9211262.

---

DIMACS is a cooperative project of Rutgers University, Princeton University, AT&T Bell Laboratories and Bellcore.

DIMACS is an NSF Science and Technology Center, funded under contract STC–91–19999; and also receives support from the New Jersey Commission on Science and Technology.

## ABSTRACT

We present a weighted version of Megiddo's multidimensional search technique and use it to obtain faster algorithms for certain convex optimization problems in $\mathbf{R}^d$, for fixed $d$. This leads to speed-ups by a factor of $\log^d n$ for applications such as solving the Lagrangian duals of matroidal knapsack problems and of constrained optimum subgraph problems on graphs of bounded tree-width.

# 1 Introduction

This paper has three main parts. In the first (Section 2) we present a weighted version of the multidimensional search technique of Megiddo [Meg84, Dyer86, Cla86]. Part two (Section 3) discusses the application of our result to a class of convex optimization problems in fixed dimension which were studied earlier by Cohen and Megiddo [Coh91, CoMe91] and, in a different context, by Aneja and Kabadi [AnKa91]. In rough terms, the results in [Coh91, CoMe91, AnKa91] can be summarized as follows. Suppose that $g$ is a concave function whose domain $\mathcal{Q}$ is a convex subset of $\mathbf{R}^d$ and that $g$ is computable in $O(T)$ time by an an algorithm $\mathcal{A}$ that only performs additions, multiplications by constants, copies, and comparisons on intermediate values that depend on the input numbers. Then, $g$ can be maximized in $O(T^{d+1})$ time. Cohen and Megiddo go on to show that substantial speed-ups are possible by exploiting whatever parallelism is inherent to algorithm $A$. Thus, if $\mathcal{A}$ carries out $D$ parallel steps, each of which does at most $M$ comparisons, the running time will be $O((D \log M)^d T)$. By applying weighted multidimensional search and a generalization of Cole's circuit simulation technique [Cole87] we are able to reduce this to $O((D^d + \log^d M)T)$ in some cases.

Lagrangian relaxation is a source of several problems that fall into the framework described above [AnKa91]. This widely-used approach is based on the observation that many hard optimization problems are actually easy problems that are complicated by a relatively small set of side constraints. By "pricing out" the bad side constraints into the objective function, one obtains a simpler convex optimization problem whose optimum solution provides good bounds on the optimum value of the original problem. The third part of this paper (Section 4) explains the application of our results to Lagrangian relaxation problems where the number of bad constraints is fixed. We give two examples of problems where the methods described in Section 3 give faster algorithms than those of [AnKa91, Coh91]: solving the Lagrangian duals of *matroidal knapsack problems* [CMV89] and of certain constrained optimum subgraph problems on graphs of bounded tree-width.

An earlier version of this paper appears in [AgFe92].

# 2 Weighted Multidimensional Search

Let us first introduce some notation. Suppose $\Lambda \subseteq \mathbf{R}^d$ is convex and that $h : \mathbf{R}^d \to \mathbf{R}$ is an affine function. Define $\text{sign}_\Lambda(h)$ as

$$\text{sign}_\Lambda(h) = \begin{cases} 0 & \text{if } h(\lambda) = 0 \text{ for some } \lambda \in \Lambda \\ +1 & \text{if } h(\lambda) > 0 \text{ for all } \lambda \in \Lambda \\ -1 & \text{if } h(\lambda) < 0 \text{ for all } \lambda \in \Lambda \end{cases}$$

We will write sign for $\text{sign}_\Lambda$ when no confusion can arise. A function $h$ is *resolved* if $\text{sign}_\Lambda(h)$ has been computed. Obviously, if $h(\lambda) = a_0$, $\text{sign}(h)$ can be immediately determined from the sign of $a_0$.

Suppose we have a set $\mathcal{H}$ of $d$-dimensional affine functions and an oracle $\mathcal{B}^d$ that can compute $\text{sign}_\Lambda(h)$ for any $h \in \mathcal{H}$. The problem is to resolve every $h \in \mathcal{H}$ using as few oracle calls as possible. The following result is proved in [Meg84, Dyer86, Cla86].

**Theorem 1** *For each fixed $d \geq 0$ there exist positive constants $\beta(d)$ and $\alpha(d)$, $\alpha(d) < 1/2$, and an algorithm* SEARCH *such that, given a set $\mathcal{H}$ of affine functions,* SEARCH *resolves every $h \in \mathcal{H}' \subseteq \mathcal{H}$, where $|\mathcal{H}'| \geq \alpha(d) \cdot |\mathcal{H}|$, by making at most $\beta(d)$ calls to $\mathcal{B}^d$. Furthermore, the work done by* SEARCH *in addition to the oracle calls is $O(|\mathcal{H}|)$.*

(In reality, the above references have proofs of this result for the case where $\Lambda$ is a single point; however, the extension to convex sets is trivial.) By repeatedly applying algorithm SEARCH we can resolve all functions in $\mathcal{H}$ with $O(\log |\mathcal{H}|)$ oracle calls. In this section, we shall prove a weighted version of Theorem 1. Let $S$ be a set on which a weight function $w : S \to \mathbf{R}^+$ has been defined. For $S' \subseteq S$ we write $w(S')$ to denote $\sum_{s \in S'} w(s)$. We have the following result.

**Theorem 2** *For each $d \geq 0$, there exist constants $\beta(d)$ and $\alpha(d)$, $\alpha \leq 1/2$ and an algorithm* WEIGHTED-SEARCH *with the following property. Given a set $\mathcal{H}$ of affine functions and a weight function $w : \mathcal{H} \to \mathbf{R}^+$,* WEIGHTED-SEARCH *resolves every $h \in \mathcal{H}' \subseteq \mathcal{H}$, where $w(\mathcal{H}') \geq \alpha(d) \cdot w(\mathcal{H})$ by making at most $\beta(d)$ calls to $\mathcal{B}^d$. Furthermore, the work done by* WEIGHTED-SEARCH *in addition to the oracle calls is $O(|\mathcal{H}|)$.*

The proof of this theorem will require some preliminary results, which are discussed next.

## 2.1 Preliminaries

Procedure WEIGHTED-SEARCH uses two simple algorithms. The first is MATCH, which given two sets $A$ and $B$, attempts to match disjoint subsets of $B$ with elements of $A$ in a "greedy" manner.

**Algorithm** MATCH

**Input:** Sets $A = \{a_1, \ldots, a_{|A|}\}$ and $B = \{b_1, \ldots, b_{|B|}\}$ and a weight function $w : A \cup B \to \mathbf{R}^+$.

**Output:** Either FAILURE or disjoint sets $S_1, \ldots, S_{|A|}$ such that, for each $i$, $S_i = \{a_i\} \cup D_i$ where $D_i \subseteq B$ and $w(D_i) \geq w(a_i)$.

```
begin
    j ← 1
    for i = 1 to |A| do begin
        D_i ← ∅
        while w(D_i) < w(a_i) do begin
            if j > |B| then return FAILURE
            D_i ← D_i ∪ b_j
```

```
            j ← j + 1
        end
        S_i ← {a_i} ∪ D_i
    end;
    return S_1, ..., S_|A|
end
```

The running time of this algorithm is clearly $O(|B|)$. We shall say that MATCH *succeeds* if it does not return FAILURE. If MATCH succeeds, then the solution returned obviously satisfies its output conditions. The next lemma gives one scenario in which MATCH always succeeds.

**Lemma 1** *If* $\min_{x \in A} w(x) \geq \max_{y \in B} w(y)$ *and* $w(B) \geq 2w(A)$*, then* MATCH *succeeds. Furthermore, each set* $S_i = \{a_i\} \cup D_i$ *returned by* MATCH *satisfies* $w(D_i) < 2w(a_i)$*.*

**Proof:** (By contradiction.) Suppose the conditions of the lemma hold and that MATCH returns FAILURE. Then, there exists a $k$, $1 \leq k \leq |A|$, such that, at the $k$th iteration of the **for** loop, MATCH runs out of elements of $B$ to match up with $a_k$. Let $D_1, \ldots, D_k$ be the subsets of $B$ constructed up to this point, where $D_j = \{d_{j1}, d_{j2}, \ldots, d_{jl_j}\}$. Clearly, $\sum_{j=1}^{k} w(D_j) = w(B)$. By construction, $w(D_k) < w(a_k)$ and, for $1 \leq j \leq k-1$, $w(D_j) - w(d_{jl_j}) < w(a_j)$. Thus, for $1 \leq j \leq k-1$, $w(D_j) < w(a_j) + w(d_{jl_j}) \leq 2w(a_j)$, as $\min_{x \in A} w(x) \geq \max_{y \in B} w(y)$. Also,

$$w(B) = \sum_{j=1}^{k} w(D_j) < \sum_{j=1}^{k} 2w(a_j) \leq 2w(A),$$

a contradiction. Therefore, MATCH succeeds, and for each set $S_i = \{a_i\} \cup D_i$, $w(D_i) < 2w(a_i)$. □

MATCH is invoked by the following algorithm.

**Algorithm** PAIRING

**Input:** Sets $A$, $B$, a weight function $w : A \cup B \rightarrow \mathbf{R}^+$, and a number $m$ such that $W/2 \geq w(B) \geq w(A) \geq (W/2 - m)$, where $W = w(A \cup B) + m$

**Output:** $k \geq 0$ disjoint sets $S_1, \ldots, S_k$, and an element $e$ satisfying the following conditions:

(P1) Each $S_i$ has the form $S_i = \{c_i\} \cup D_i$, where $e \neq c_i$, and either
(1) for all $i$, $e, c_i \in A$ and $D_i \subseteq B$, or
(2) for all $i$, $e, c_i \in B$ and $D_i \subseteq A$.

(P2) for all $i$, $2w(c_i) > w(D_i) \geq w(c_i)$.

(P3) $\sum_{i=1}^{k} w(c_i) + w(e) + m \geq W/6$.

**Step 1.** Find $a \in A$ and $b \in B$ such that there exists a set $A' \subseteq \{x \in A : x \neq a, w(x) = w(a)\}$, and a set $B' \subseteq \{x \in B : x \neq b, w(x) = w(b)\}$, such that $w(A_1), w(B_1) \leq w(A)/3$ and $w(A_1 + a), w(B_1 + b) \geq w(A)/3$, where $A_1 = \{x \in A : w(x) > w(a)\} \cup A'$ and $B_1 = \{x \in B : w(x) > w(b)\} \cup B'$. Let $A_2 = A - A_1$ and $B_2 = B - B_1$.

**Step 2.** If $w(a) \geq w(b)$, do the following steps.

**Step 2(a).** If $w(a) + m \geq W/6$, then return $k = 0$ and $e = a$.

**Step 2(b).** Call MATCH with inputs $A_1$ and $B_2$. Let $S_1, \ldots, S_{|A_1|}$ be the sets returned by this call. Return $S_1, \ldots, S_{|A_1|}$, and $e = a$.

**Step 3.** If $w(a) < w(b)$, do the following steps.

**Step 3(a).** If $w(b) + m \geq W/6$, then return $k = 0$ and $e = b$.

**Step 3(b).** Call MATCH with inputs $B_1$ and $A_2$. Let $S_1, \ldots, S_{|B_1|}$ be the sets returned by this call. Return $S_1, \ldots, S_{|B_1|}$ and $e = b$.

**Lemma 2** PAIRING *correctly computes output satisfying conditions (P1)–(P3).*

**Proof:** If the output is returned in Step 2(a) or Step 3(a), the conditions are trivially satisfied. We now consider Step 2(b); the analysis for Step 3(b) is similar. By construction, $\min_{x \in A_1} w(x) \geq \max_{y \in B_2} w(y)$ and $w(B_2) \geq 2w(A)/3 \geq 2w(A_1)$. Since the conditions of Lemma 1 are satisfied, MATCH succeeds and conditions (P1) and (P2) are satisfied. Since, MATCH works correctly, we have $\sum_{i=1}^{|A_1|} w(c_i) = w(A_1)$. Therefore, $\sum_{i=1}^{k} w(c_i) + w(e) + m = w(A_1) + w(a) + m$. As $w(A_1) + w(a) \geq w(A)/3$,

$$\sum_{i=1}^{k} w(c_i) + w(e) + m \geq w(A)/3 + m \geq (w(A) + m)/3$$

Since $w(A) + m \geq W/2$, we obtain

$$\sum_{i=1}^{k} w(c_i) + w(e) + m \geq W/6$$

as desired. □

PAIRING can be implemented to run in $O(n)$ time, where $n = |A| + |B|$. Step 1 takes $O(n)$ time as elements $a$ and $b$ can be found by repeated median finding [CLR90]. Steps 2 and 3 also take linear time, as MATCH takes linear time.

## 2.2 The Search Algorithm

We shall now prove Theorem 2. The implementation of WEIGHTED-SEARCH that we propose is an extension of Megiddo [Meg84] and Dyer's [Dyer86] algorithms for unweighted multi-dimensional search (see Theorem 1). Suppose $\mathcal{H} = \{h_1, \ldots, h_n\}$, where $h_i(x) = a_i^T \lambda + d_i$. If $a_i = 0$, $\text{sign}(h_i)$ is simply the sign of $d_i$, and no oracle calls are needed. Thus, the presence of $h_i$'s with $a_i = 0$ can only help. We shall henceforth assume that $a_i \neq 0$, for $i = 1, \ldots, n$. In this case, each affine function $h_i$ corresponds to a hyperplane $H_i \in \mathbf{R}^d$ where $H_i = \{\lambda : h_i(\lambda) = 0\}$. Computing $\text{sign}(h_i)$ is thus equivalent to determining whether $H_i$ intersects $\Lambda$ and, if not, which side of $H_i$ contains $\Lambda$. We shall find it convenient to deal interchangeably with hyperplanes and affine functions and to extend the weight function $w$ to these hyperplanes by making $w(H_i) = w(h_i)$.

The numbers $\beta(d)$ and $\alpha(d)$ are derived recursively with respect to the dimension. For $d = 1$, the hyperplanes are $n$ real numbers $\lambda_1, \ldots, \lambda_n$. In this case, WEIGHTED-SEARCH finds the weighted median $\lambda_m$, inquires about its position relative to $\lambda^*$, and resolves either $\{\lambda_i : \lambda_i \leq \lambda_m\}$ or $\{\lambda_i : \lambda_i \geq \lambda_m\}$. Thus, $\beta(1) = 1$ and $\alpha(1) = 1/2$. For $d \geq 2$ we proceed as follows.

Form a set $\mathcal{H}_\infty = \{H_i : a_{i2} = 0\}$. Each $H_i \in \mathcal{H} - \mathcal{H}_\infty$ intersects the $\lambda_1$–$\lambda_2$ plane in a straight line $a_{i1}\lambda_1 + a_{i2}\lambda_2 = b_i$. Since $\text{sign}(h(\lambda)) = \text{sign}(l) \cdot \text{sign}(h(\lambda)/l)$, we can rewrite the equations of these hyperplanes so that $a_{i1} \geq 0$. Let the *slope* $\alpha_i$ of $H_i$ be the same as that of $a_{i1}\lambda_1 + a_{i2}\lambda_2 = b_i$ with respect to $\lambda_2 = 0$; i.e., let $\alpha_i = (-a_{i1}/a_{i2})$. Let $\alpha^*$ be the weighted median of the set $\{\alpha_i\}$ where the weight of $\alpha_i$ is $w(H_i)$. Now we make the slopes of roughly weighted half of the hyperplanes nonnegative and weighted half nonpositive by using the change of variables $\lambda_2 = \lambda_2' + \alpha^* \lambda_1$ and $a_{i1} = a_{i1}' - \alpha^* a_{i2}$. This change of variables is only done to simplify the exposition and, indeed, needs to be reversed before making an oracle call. For convenience, we now drop the primes on $\lambda_1'$ and $a_{i2}'$. Recalculate the slopes of the hyperplanes after making this change in variables. All hyperplanes that originally had a slope of $\alpha^*$ will have 0 slope. Let $\mathcal{H}_0 = \{H_i : \alpha_i = 0\}$, $\mathcal{H}_- = \{H_i : \alpha_i < 0\}$, and $\mathcal{H}_+ = \{H_i : \alpha_i > 0\}$.

Let $m = w(\mathcal{H}_\infty) + w(\mathcal{H}_0)$ and $W = w(\mathcal{H})$. Since 0 is our new weighted median slope, $w(\mathcal{H}_-) \leq (W - w(\mathcal{H}_\infty))/2 \leq W/2$ and $w(\mathcal{H}_-) + w(\mathcal{H}_0) \geq (W - w(\mathcal{H}_\infty))/2$. Therefore, $w(\mathcal{H}_-) \geq W/2 - w(\mathcal{H}_\infty)/2 - w(\mathcal{H}_0) \geq W/2 - m$. Similarly, $W/2 \geq w(\mathcal{H}_+) \geq (W/2 - m)$. Thus, sets $\mathcal{H}_-, \mathcal{H}_+$ and the number $m$ satisfy the preconditions of PAIRING — assuming, without loss of generality, that $w(\mathcal{H}_-) \leq w(\mathcal{H}_+)$. WEIGHTED-SEARCH calls PAIRING$(\mathcal{H}_-, \mathcal{H}_+, m)$. Let $S_1, \ldots, S_l$, $e$ be the sets and the element returned, where $S_i = \{c_i\} \cup D_i$. By output condition (P3) of PAIRING,

$$\sum_{i=1}^{l} w(c_i) + w(e) + m \geq W/6 \tag{1}$$

Next, we resolve the hyperplane associated with $e$, denoted by $H_e$, by calling the oracle directly. For the hyperplanes corresponding to elements in $S_1, \ldots, S_l$, we do the following.

Suppose that for each set $S_i = \{c_i\} \cup D_i$, $c_i$ corresponds to a hyperplane $H_i \in \mathcal{H}_-$ and that $D_i$, has a corresponding set of hyperplanes $\{H_{i1}, H_{i2}, \ldots, H_{iq_i}\} \subseteq \mathcal{H}_+$. (The analysis

for the case where $c_i$ is associated with a hyperplane in $\mathcal{H}_+$ completely analogous.) For each $i$, form pairs $(H_i, H_{i1}), (H_i, H_{i2}), \ldots, (H_i, H_{iq_i})$. By Lemma 1, for each $i$ and $j$,

$$w(H_{ij}) \leq w(H_i) \tag{2}$$

Consider a typical pair $(H_i, H_{ij})$. Since $H_i$ and $H_{ij}$ have strictly negative and strictly positive slopes, respectively, their intersection is a $(d-1)$-dimensional hyperplane. Through this intersection, we can draw hyperplanes $H_{ij}^{(1)}$ and $H_{ij}^{(2)}$ whose slopes are $+\infty$ and 0 respectively. Mathematically,

$$\begin{aligned} H_{ij}^{(1)}: \qquad (a_{i2} - a_{ij2})\lambda_1 &= (a_{i2}b_{ij} - a_{ij2}b_i) - \sum_{r=3}^{d}(a_{i2}a_{ijr} - a_{ij2}a_{ir})\lambda_r \\ H_{ij}^{(2)}: \qquad (a_{i2} - a_{ij2})\lambda_2 &= (b_i - b_{ij}) - \sum_{r=3}^{d}(a_{ir} - a_{ijr})\lambda_r \end{aligned}$$

Note that $H_{ij}^{(1)}$ and $H_{ij}^{(2)}$ are $(d-1)$-dimensional hyperplanes. Now, assign a weight of $\min(w(H_i), w(H_{ij}))$ to each of $H_{ij}^{(1)}$ and $H_{ij}^{(2)}$. ¿From equation (2), $\min(w(H_i), w(H_{ij})) = w(H_{ij})$. This along with condition (P3) of PAIRING gives us

$$\sum_{r=1}^{q_i} w(H_{ir}^{(1)}) = \sum_{r=1}^{q_i} w(H_{ir}^{(2)}) = \sum_{r=1}^{q_i} w(H_{ir}) \geq w(c_i) = w(H_i) \tag{3}$$

Recursively apply WEIGHTED-SEARCH to the set of $(d-1)$-dimensional hyperplanes $\{H_{ij}^{(1)}\} \cup \mathcal{H}_\infty$. This requires $\beta(d-1)$ oracle calls. Let $W_\infty$ and $W_1$ denote the weights of the hyperplanes resolved ¿from sets $\mathcal{H}_\infty$ and $\{H_{ij}^{(1)}\}$, respectively, and let $w(\{H_{ij}^{(1)}\}) = \sum_{i=1}^{l}\sum_{r=1}^{q_i} w(H_{ir})$. Then,

$$W_\infty + W_1 \geq \alpha(d-1)\left(w(\mathcal{H}_\infty) + w(\{H_{ij}^{(1)}\})\right). \tag{4}$$

Let $\mathcal{H}^{(2)}$ be the set of hyperplanes in $\{H_{ij}^{(2)}\}$ for which the corresponding $H_{ij}^{(1)}$'s has been resolved in the previous step. Recursively apply WEIGHTED-SEARCH to the set of $(d-1)$-dimensional hyperplanes in $\mathcal{H}^{(2)} \cup \mathcal{H}_0$. This requires at most $\beta(d-1)$ oracle calls. Letting $W_0$ and $W_{12}$ denote the weights of the hyperplanes resolved ¿from sets $\mathcal{H}_0$ and $\mathcal{H}^{(2)}$, respectively, we have

$$W_0 + W_{12} \geq \alpha(d-1)(w(\mathcal{H}_0) + W_1). \tag{5}$$

To summarize the algorithm up to this point, observe that, from the original set $\mathcal{H}$, we have resolved an element $e$ of weight $w(e)$, a subset of weight $W_\infty$ of the planes in $\mathcal{H}_\infty$, a subset of weight $W_0$ of the planes in $\mathcal{H}_0$. In addition to this, we have resolved a subset of weight $W_1$ of the hyperplanes in set $\{H_{ij}^{(1)}\}$, and a subset of weight $W_{12}$ of the hyperplanes in set $\mathcal{H}^{(2)}$. For each hyperplane contributing to $W_{12}$, we have also resolved its pair in the set $\{H_{ij}^{(1)}\}$. However, $W_1$ and $W_{12}$ represent the total weights of sets of auxiliary hyperplanes, rather than elements of $\mathcal{H}$. We shall now show that by resolving such auxiliary hyperplanes, we are guaranteeing the resolution of sufficiently many hyperplanes from $(\mathcal{H}_- \cup \mathcal{H}_+) - \{H_e\}$.

**Lemma 3** *Let $W_a$ be the total weight of the hyperplanes resolved in $(\mathcal{H}_- \cup \mathcal{H}_+) - \{H_e\}$. Then, $W_a \geq W_{12}/2$.*

**Proof:** Consider a particular set $S_i = \{H_i\} \cup \{H_{i1}, H_{i2}, \ldots, H_{iq_i}\}$. The auxiliary hyperplanes formed by the intersection of hyperplanes in $S_i$ are $(H_{i1}^{(1)}, H_{i1}^{(2)}), \cdots, (H_{iq_i}^{(1)}, H_{iq_i}^{(2)})$. Suppose $H_{i1}^{(2)}, H_{i2}^{(2)}, \cdots H_{ip}^{(2)}$ were resolved in the second recursive call. Then these hyperplanes contributed to $W_{12}$. Hence, the contribution, $C_i$, of the auxiliary hyperplanes resulting from $S_i$ to $W_{12}$ is $C_i = \sum_{j=1}^{p} w(H_{ij}^{(2)})$ and $W_{12}$ can be written as

$$W_{12} = \sum_{i=1}^{l} C_i. \tag{6}$$

For each $H_{ij}^{(2)}$ that gets resolved, its corresponding $H_{ij}^{(1)}$ has already been resolved in the first recursive call. We now rely on an observation of Megiddo [Meg84], who noted that if we know the position of $\Lambda$ relative to both $H_{ij}^{(1)}$ and $H_{ij}^{(2)}$, we can determine the position of $\Lambda$ relative to at least one of $H_i$ and $H_{ij}$. Let $R_i$ be the sum of the weights of the hyperplanes resolved from $S_i$. Since $\cup_{i=1}^{l} S_i \subseteq (\mathcal{H}_- \cup \mathcal{H}_+) - \{H_e\}$,

$$W_a \geq \sum_{i=1}^{l} R_i \tag{7}$$

We have two cases to consider:

**Case I:** In each pair $H_{ij}$ is resolved. Then, due to equation (3) we have

$$R_i = \sum_{j=1}^{p} w(H_{ij}) = \sum_{j=1}^{p} w(H_{ij}^{(2)}) = C_i$$

**Case II:** $H_i$ is resolved in at least one pair. Then, equation (3) along with condition (P3) of PAIRING implies that

$$R_i \geq w(H_i) = w(c_i) > \sum_{j=1}^{l} w(H_{ij})/2 \geq \sum_{j=1}^{p} w(H_{ij}^{(2)})/2 = C_i/2$$

Therefore, in either case $R_i \geq C_i/2$. This along with equations (6) and (7) gives us the following:

$$W_a \geq \sum_{i=1}^{l} R_i \geq \sum_{i=1}^{l} C_i/2 = W_{12}/2$$

Therefore, $W_a \geq W_{12}/2$. □

Let $W_T$ be the total weight of the hyperplanes ¿from $\mathcal{H}$ that are resolved by our algorithm; i.e., $W_T = w(e) + W_\infty + W_0 + W_a$. Using Lemma 3 and equations (3)–(5), we have:

$$\begin{aligned} W_T &\geq w(e) + W_\infty + W_0 + W_{12}/2 \\ &\geq w(e) + W_\infty + \alpha(d-1)\cdot(w(\mathcal{H}_0) + W_1)/2 \\ &\geq w(e) + \alpha(d-1)\cdot\left(w(\mathcal{H}_0) + \alpha(d-1)\cdot(w(\mathcal{H}_\infty) + w(\{H_{ij}^{(1)}\})\right)/2 \\ &\geq \alpha(d-1)^2\cdot\left(w(e) + w(\mathcal{H}_0) + w(\mathcal{H}_\infty) + \sum_{i=1}^{l} w(c_i)\right)/2 \\ &\geq \alpha(d-1)^2\cdot W/12 \end{aligned}$$

¿From the preceding discussion, we conclude that the number of oracle calls satisfies $\beta(d) = 2\beta(d-1)+1$, with $\beta(1) = 1$, and that the fraction of the total weight satisfies $\alpha(d) \geq \alpha(d-1)^2/12$, with $\alpha(1) = 1/2$. Hence, $\beta(d) = 2^d - 1$ and $\alpha(d) = 12/24^{2^{d-1}}$.

The same arguments as in [Dyer86] can be used to show that the total work done by WEIGHTED-SEARCH is $O(n)$. We omit the details.

## 2.3 Improving the Efficiency of the Search

Following Dyer [Dyer86], the *efficiency* of a search scheme is the ratio $e = \alpha(d)/\beta(d)$. As for unweighted search, the efficiency of a weighted search scheme will affect the running time of the algorithms that use the scheme as a subroutine. The search scheme we have just presented has $e$ that is doubly exponentially small in $d$. Borrowing ideas from [Dyer86], we shall sketch how to make the efficiency singly exponentially small.

Let us write $\mathcal{S}(d, \beta, \alpha)$ to denote a weighted search scheme that, given a set of weighted affine functions in $\mathbf{R}^d$ of total weight $W$, resolves a fraction of total weight $\alpha\cdot W$ using $\beta$ oracle calls. Thus, the algorithm that we have developed can be denoted by $\mathcal{S}(d, 2^d - 1, 12/24^{2^{d-1}})$. Suppose that we have a $\mathcal{S}(d-1, \beta(d-1), \alpha(d-1))$ scheme $S_{d-1}$. To obtain a search procedure for $\mathbf{R}^d$, proceed as follows. First, construct a $\mathcal{S}(d-1, \beta', \alpha')$ procedure with

$$\alpha' = 1 - (1-\alpha(d))^r \qquad \text{and} \qquad \beta' = r\cdot\beta(d).$$

by carrying out $r$ iterations, each of which consists of applying $S_{d-1}$ and removing the hyperplanes that are resolved. Next, use $\mathcal{S}(d-1, \beta', \alpha')$ and the pairing scheme described earlier to obtain a a procedure $\mathcal{S}(d, \beta'', \alpha''))$ where

$$\alpha'' = [1 - (1-\alpha(d-1))^r]^2/12, \qquad \text{and} \qquad \beta'' = 2\cdot r\cdot\beta(d-1) + 1.$$

Applying this procedure $l$ times gives us a procedure $\mathcal{S}(d, \beta(d), \alpha(d))$ that solves $d$-dimensional hyperplanes with

$$\beta(d) = l\cdot\beta'' = l(2r\cdot\beta(d-1)+1), \qquad \text{and}$$

$$\alpha(d) = 1 - (1-\alpha'')^l = 1 - \left\{1 - \frac{1}{12}[1-(1-\alpha(d-1))^r]^2\right\}^l.$$

We can use this framework to obtain a scheme $\mathcal{S}(d, \beta(d), \alpha(d)$ where $\alpha(d) \geq 1/12$ for all $d$. For $d = 1$, we can easily obtain a scheme $\mathcal{S}(1, 1, 1/2)$. Suppose $\alpha(d-1) \geq 1/12$. If we choose $l = 2$ and $r = 15$, we get

$$\alpha(d) = 1 - \left\{1 - \frac{1}{12}[1 - (1 - 1/12)^{15}]^2\right\}^2 \geq 1/12,$$

as desired. Now,

$$\beta(d) = 2 \cdot (2 \cdot 15 \cdot \beta(d-1) + 1) \leq 2(60^{d-1}),$$

and we have a procedure $\mathcal{S}(d, 2(60^{d-1}), 1/12)$ whose efficiency is singly exponentially small.

# 3 Convex Optimization in Fixed Dimension

An algorithm is *piecewise affine* if the only operations it performs on intermediate values that depend on the input numbers are additions, multiplications by constants, copies, and comparisons [Coh91, CoMe91]. Several well-known algorithms fall into this category, including many network optimization algorithms [Chv83, Tarj83]. Suppose $\mathcal{Q} \subseteq \mathbf{R}^d$ is a (possibly empty) convex set defined by a set of at most $l$ linear inequalities, where $l$ is some fixed integer. Let $g : \mathcal{Q} \to \mathbf{R}$ be a concave function. Our goal is to compute

$$g^* = \max\{g(\lambda) : \lambda \in \mathcal{Q}\}. \tag{8}$$

or, if $\mathcal{Q} = \emptyset$, to return a message that this problem is infeasible. Cohen and Megiddo [Coh91, CoMe91] showed that, if $g$ is computable by a piecewise affine algorithm that runs in time $T$ and makes $D$ sets of at most $M$ comparisons, then problem (8) can be solved in $O((D \log M)^d T)$ time. Closely related results were obtained by Norton, Plotkin, and Tardos [NPT92] and Aneja and Kabadi [AnKa91]. The main result of this section is to show that weighted multidimensional search in conjunction with Cole's circuit simulation technique [Cole87] can sometimes be used to solve (8) in $O((D^d + \log^d M)T)$ time.

To streamline the presentation, for the most part we shall omit any mention of constants that depend on $d$. The magnitude of these values is discussed in Section 3.3.

## 3.1 The Basic Scheme

We now review the solution scheme of Megiddo and Cohen and Aneja and Kabadi [Coh91, AnKa91] as it forms the basis for our algorithm. Our presentation is somewhat simpler, among other reasons because it avoids the notion of "minimal weak approximation" used in [Coh91]. We shall assume that problem (8) is bounded. This is done without loss of generality, since unbounded problems can be handled by Seidel's technique of adding "constraints at infinity" [Sei91]. Note also that if $g$ is computable by a piecewise affine algorithm, it is the lower envelope of a finite set of linear functions [Coh91]. We say that a linear function $f : \mathbf{R}^d \to \mathbf{R}$ is *active* at $\lambda^{(0)} \in \mathcal{Q}$ if $g(\lambda^{(0)}) = f(\lambda^{(0)})$ and $g(\lambda) \leq f(\lambda)$ for all $\lambda \in \mathcal{Q}$ and we shall write $\Lambda$ to denote the set of maximizers of $g$.

Let us refer to the algorithm that solves a $d$-dimensional problem of the form (8) as algorithm $\mathcal{C}^d$. Let $H$ be a hyperplane in $\mathbf{R}^d$, and let $g^*_H$ denote the maximum of $g$ on $H$; i.e.,

$$g^*_H = \max\{g(\lambda) : \lambda \in H \cap \mathcal{Q}\}$$

Suppose we have an oracle $\mathcal{B}^d$ that, as in Section 2, returns $\text{sign}_\Lambda(h)$ for any given affine function $h$. Moreover, if $h$ defines a hyperplane $H$, $\mathcal{B}^d$ returns $g^*_H$, assuming $H \cap \mathcal{Q} \neq \emptyset$.

Obviously, $\mathcal{A}$ can play the role of $\mathcal{C}^0$. For $d \geq 1$, $\mathcal{C}^d$ proceeds as follows. First, it determines whether $\mathcal{Q}$ is empty and, if so, returns a message saying that (8) is infeasible. Since $\mathcal{Q}$ is defined by a fixed number of linear inequalities, this takes $O(1)$ time. If $\mathcal{Q} \neq \emptyset$, $\mathcal{C}^d$ uses Megiddo's algorithm simulation technique [Meg79, Meg83] to do one of two things. The first option is to find a hyperplane $H$ defined by $h(\lambda) = 0$, such that $\text{sign}_\Lambda(h) = 0$. Then, clearly $g^* = g^*_H$. The second option is to find a linear function $f$ and a set of linear inequalities $\mathcal{L}$ defining a non-empty convex set $\mathcal{Q}^* \subseteq \mathcal{Q}$ such that

(C1) $\mathcal{Q}^* \subseteq \Lambda$ and

(C2) $f$ is active at every $\lambda \in \mathcal{Q}^*$.

In this case, solving (8) reduces to solving the linear programming problem

$$\max\{f(\lambda) : \lambda \in \mathcal{Q}^*\},$$

which can be done in time linear in $\mathcal{L}$, since $d$ is fixed [Meg84]. $\mathcal{C}^d$ relies on the observation that, because $\mathcal{A}$ is piecewise affine and its inputs are linear functions of $\lambda$, all the intermediate values of its real variables can be represented implicitly as linear forms in $\lambda$. Using this representation, a single computation path of $\mathcal{A}$, may correspond to the evaluation of $g(\lambda)$ for a *set* of distinct $\lambda$-values.

Suppose that for $s \leq r$, we know how to find a set $\mathcal{Q}' \subset \mathbf{R}^d$ defined by a set of linear inequalities $\mathcal{L}$ such that $\mathcal{Q}' \cap \Lambda \neq \emptyset$ and such that the outcomes of the first $r$ steps of any computation path of $\mathcal{A}$ for every $\lambda^* \in \mathcal{Q}^*$ are exactly the same (when values are represented implicitly). We wish to find such a set for $s = r + 1$. Before proceeding, note that finding $\mathcal{Q}^*$ when $s = 0$ is trivial, since we can choose $\mathcal{Q}^* = \mathcal{Q}$. For $s = r + 1$, observe that knowing the outcomes of the first $r$ steps tells us what the $(r+1)^{st}$ step of $\mathcal{A}$ will be; we now need to determine the outcome of this step. If the $(r+1)^{st}$ step is an addition of two or more numbers, a multiplication by a constant, or an assignment, $\mathcal{C}^d$ does the corresponding operations with linear forms and proceeds to the next step of $\mathcal{A}$.

If the $(r+1)^{st}$ step is a comparison between two variables, $\mathcal{C}^d$ compares the corresponding linear forms $f_1(\lambda)$ and $f_2(\lambda)$ using $\mathcal{B}^d$ to resolve the function $h(\lambda) = f_1(\lambda) - f_2(\lambda)$. Suppose $h(\lambda) = 0$ defines a hyperplane $H$. If $\text{sign}_\Lambda(h) = 0$, then $g^*$ is the value of $g^*_H$ returned by the oracle, and $\mathcal{C}^d$ halts. Otherwise, $\mathcal{C}^d$ updates $\mathcal{L}$ by adding the inequality $h(\lambda) > 0$ if $\text{sign}(h) = +1$, or the inequality $h(\lambda) < 0$ if $\text{sign}(h) = -1$. The next step to be simulated from $\mathcal{A}$ will be the action corresponding to $f_1(\lambda) > f_2(\lambda)$ or $f_1(\lambda) < f_2(\lambda)$ depending on whether $\text{sign}(h)$ is $+1$ or $-1$. If $h$ is a constant function, the oracle's job is trivial, since the outcome of the comparison is independent of $\lambda$ and the simulation proceeds accordingly.

If $\mathcal{C}^d$ simulates $\mathcal{A}$ to completion, $\mathcal{Q}^*$ will satisfy condition (C1). Furthermore, the output of $\mathcal{A}$ will be a linear function $f$ satisfying condition (C2). Since $\mathcal{A}$ does $O(T)$ comparisons, $|\mathcal{L}| = O(T)$ and the resulting linear program in $d$ variables can be solved in $O(T)$ time [Meg84]. The total time for algorithm $\mathcal{C}^d$ is therefore $O(T \cdot b(d))$, where $b(d)$ is the running time of $\mathcal{B}^d$. We now turn our attention to the implementation of $\mathcal{B}^d$.

**Implementing the oracle.** Let $h(\lambda) = \sum_{i=1}^{d} a_i\lambda_i + b$ be the function to be resolved. If $a_i = 0$ for $i = 1, \ldots, n$, sign$(h)$ depends simply on the sign of $b$. Otherwise, $H = \{\lambda : h(\lambda) = 0\}$ is a hyperplane in $\mathbf{R}^d$. To resolve $h$, $\mathcal{B}^d$ first determines if $H \cap \mathcal{Q} = \emptyset$. If this is so, then, since $\Lambda \subseteq \mathcal{Q}$, we simply need to find a point $\lambda^{(0)} \in \mathcal{Q}$ and evaluate $\text{sign}_{\{\lambda^{(0)}\}}(h)$. Determining if the intersection of $H$ and $\mathcal{Q}$ is non-empty, and finding a point inside $\mathcal{Q}$ take $O(1)$ time, since $\mathcal{Q}$ is defined by a set of $O(l)$ linear inequalities and $l$ is fixed.

¿From now on, assume $H \cap \mathcal{Q} \neq \emptyset$. Now, if $\Lambda \cap H = \emptyset$, due to concavity of $g$, the set of all points $\lambda'$ such that $g(\lambda') > g^*_H$ is contained in only one side of $H$. This observation leads to the following result, which is the basis for the implementation of $\mathcal{B}^d$.

**Lemma 4** *Let $H = \{\lambda : h(\lambda) = 0\}$ be a hyperplane in $\mathbf{R}^d$ and for every real number $a$, let $H(a)$ denote the hyperplane given by $H(a) = \{\lambda : h(\lambda) = a\}$. Then,*

$$sign_{\Lambda}(h) = \begin{cases} +1 & \text{if } (\exists \epsilon > 0)[g^*_{H(\epsilon)} > g^*_H] \\ -1 & \text{if } (\exists \epsilon > 0)[g^*_{H(-\epsilon)} > g^*_H] \\ 0 & \text{otherwise} \end{cases}$$

*Furthermore, if $sign_{\Lambda}(h) = +1$ [$sign_{\Lambda}(h) = -1$], then $g^*_{H(\gamma)} > g^*_H$ [$g^*_{H(-\gamma)} > g^*_H$] for all $\gamma \in (0, \epsilon]$, where $\epsilon > 0$ is sufficiently small.*

The Lemma tacitly assumes that $H(\epsilon) \cap \mathcal{Q} \neq \emptyset$ [$H(-\epsilon) \cap \mathcal{Q} \neq \emptyset$] for some $\epsilon > 0$. If $H(\epsilon) \cap \mathcal{Q} = \emptyset$ [$H(-\epsilon) \cap \mathcal{Q} = \emptyset$] for every $\epsilon > 0$, we can immediately conclude that sign$(h) \neq +1$ [sign$(h) \neq -1$]. Thus, we shall continue assuming that $H(\epsilon) \cap \mathcal{Q} \neq \emptyset$ and $H(-\epsilon) \cap \mathcal{Q} \neq \emptyset$ for some $\epsilon > 0$.

Lemma 4 implies that we can implement $\mathcal{B}^d$ by computing $g^*_H$, $g^*_{H(\epsilon)}$ and $g^*_{H(-\epsilon)}$ for sufficiently small $\epsilon > 0$. Computing $g^*_H$ is a $(d-1)$-dimensional problem of the same form as that of computing $g^*$; hence, $g^*_H$ can be calculated by recursively calling $\mathcal{C}^{d-1}$. We can also compute $g^*_{H(\epsilon)}$ using $\mathcal{C}^{d-1}$, provided $\mathcal{C}^{d-1}$ treats $\epsilon$ as a symbolic constant whose only known attribute is being arbitrarily small and positive. (The details of computing $g^*_{H(-\epsilon)}$ are analogous and therefore omitted.) The output $g^*_{H(\epsilon)}$ of this execution of $\mathcal{C}^{d-1}$ will depend linearly on $\epsilon$; i.e., $g^*_{H(\epsilon)} = g_0 + g_1\epsilon$. The values of $g^*_{H(\epsilon)}$ and $g^*_H$ are compared by computing $y = \text{sign}(g^*_H - g^*_{H(\epsilon)}) = \text{sign}((g^*_H - g_0) - g_1\epsilon)$. If $|g^*_H - g_0| > 0$, $d = \text{sign}(g^*_H - g_0)$, since $\epsilon$ is arbitrarily small. Otherwise, $d = \text{sign}(-g_1)$, since $\epsilon$ is positive. Of course, $\mathcal{C}^{d-1}$ will itself call $\mathcal{B}^{d-1}$, which will introduce a perturbation of its own. In order to deal effectively with the various symbolic perturbations, we shall establish a certain ordering among them.

The state of the execution of $\mathcal{C}^d$ is partially described by sequence of currently active procedure calls (i.e., calls that have not yet been completed). Let us follow one sequence of

procedure calls $\mathcal{C}^d \rightarrow \mathcal{B}^d \rightarrow \mathcal{C}^{d-1} \rightarrow \mathcal{B}^{d-1} \rightarrow \ldots \rightarrow \mathcal{B}^{d-r+1} \rightarrow \mathcal{C}^{d-r}$. Within this sequence, for $0 \leq j \leq r-1$, $\mathcal{B}^{d-j} \rightarrow \mathcal{C}^{d-j-1}$ corresponds to one of the three calls to $\mathcal{C}^{d-j-1}$ done by $\mathcal{B}^{d-j}$; we refer to this part of the sequence as *level* $j$. Each level reduces the dimension of the problem by one. Also, depending on which of the three calls the level corresponds to, the call may or may not introduce a perturbation. If it does, we shall refer to the perturbation as $\epsilon_j$. Let $\mathcal{J} = \{i_1, \ldots, i_s\} \subseteq \mathcal{I} = \{0, \ldots, r\}$ consist of all $j$ such that a perturbation is introduced up to level $j$. We assume that $0 \leq i_1 \leq r$ and, for $0 \leq j \leq s-1$, $0 \leq i_j < i_{j+1} \leq r$. The set $\mathcal{J}$ indicates which perturbations are "active" at the current stage of the execution of $\mathcal{C}^d$. The problem to be solved at level $r$ can thus be expressed as:

$$g_r^* = \max\{g(\lambda) : \lambda \in \mathcal{Q}'(\epsilon_{i_1}, \ldots, \epsilon_{i_s})\}$$

where $\mathcal{Q}'(\epsilon_{i_1}, \ldots, \epsilon_{i_s})$ is a $(d-r)$-dimensional subset of $\mathbf{R}^d$, defined by the intersection of $O(d)$ linear inequalities in $\{\lambda_i\}$ and $\epsilon_{i_1}, \ldots, \epsilon_{i_s}$.

Now, suppose $\mathcal{C}^{d-r}$ invokes $\mathcal{B}^{d-r}$ to resolve a hyperplane

$$H(\epsilon_{i_1}, \ldots, \epsilon_{i_s}) = \{\lambda : h(\lambda, \epsilon_{i_1}, \ldots, \epsilon_{i_s}) = 0\}.$$

For every real number $a$, let $H'(\epsilon_{i_1}, \ldots, \epsilon_{i_s}, a) = \{\lambda : h(\lambda, \epsilon_{i_1}, \ldots, \epsilon_{i_s}) = a\}$. Then, applying Lemma 4, $\mathcal{B}^d$ solves three problems:

$$\begin{aligned}
g_{r+1}^*(0) &= \max\{g(\lambda) : \lambda \in \mathcal{Q}'(\epsilon_{i_1}, \ldots, \epsilon_{i_s}) \cap H'(\epsilon_{i_1}, \ldots, \epsilon_{i_s}, 0)\} \\
g_{r+1}^*(\epsilon_{r+1}) &= \max\{g(\lambda) : \lambda \in \mathcal{Q}'(\epsilon_{i_1}, \ldots, \epsilon_{i_s}) \cap H'(\epsilon_{i_1}, \ldots, \epsilon_{i_s}, \epsilon_{r+1})\} \\
g_{r+1}^*(-\epsilon_{r+1}) &= \max\{g(\lambda) : \lambda \in \mathcal{Q}'(\epsilon_{i_1}, \ldots, \epsilon_{i_s}) \cap H'(\epsilon_{i_1}, \ldots, \epsilon_{i_s}, -\epsilon_{r+1})\},
\end{aligned}$$

where $\epsilon_{r+1} > 0$. By Lemma 4, if there exists an $\epsilon_{r+1} > 0$ such that $g_{r+1}^*(\epsilon_{r+1}) > g_{r+1}^*(0)$, then $g_{r+1}^*(\gamma) > g_{r+1}^*$ for *any* $\gamma \in (0, \epsilon_{r+1}]$. Thus, when dealing symbolically with $\epsilon_{r+1}$, we can assume that it is arbitrarily smaller than any one of $\epsilon_{i_1}, \ldots, \epsilon_{i_s}$. By the same reasoning, when dealing with perturbations $\epsilon_{i_1}, \ldots, \epsilon_{i_s}$, we can always act under the assumption that

$$0 < \epsilon_{i_s} \ll \epsilon_{i_{s-1}} \ll \cdots \ll \epsilon_{i_2} \ll \epsilon_{i_1} \ll 1. \tag{9}$$

Since $\mathcal{A}$ is piecewise affine, all numbers manipulated at any level of the execution of $\mathcal{C}^d$ are linear forms in the $\lambda_i$'s and the $\epsilon_i$'s. Suppose the execution of $\mathcal{C}^d$ produces a sequence of procedure calls that eventually triggers a comparison between two values. If the values involve $\lambda$ (and, possibly, some $\epsilon_i$'s), the comparison will be handled by an oracle call. Otherwise, we will be comparing linear forms in the $\epsilon_i's$. For a correct implementation of $\mathcal{C}^d$, it suffices to deal properly with the second kind of comparison. Suppose the two numbers have the form $u = u_0 + \sum_{j=1}^s u_j \epsilon_{i_j}$ and $v = v_0 + \sum_{j=1}^s v_j \epsilon_{i_j}$. We must compute sign$(t)$, where $t = t_0 + \sum_{j=1}^s t_j \epsilon_{i_j}$ and $t_j = u_j - v_j$, $j = 1, \ldots, s$. Obviously, if $t_j = 0$ for $j = 0, 1, \ldots, s$, sign$(t) = 0$. Otherwise, there is a smallest subscript $d$, $0 \leq d \leq s$ such that $|t_d| > 0$. By (9), $\epsilon_{i_{d+1}}, \ldots, \epsilon_{i_s}$, can be assumed to be arbitrarily smaller than $\epsilon_{i_d}$. Thus sign$(t) =$ sign$(t_d)$. We should note that the use of perturbation techniques is common in mathematical programming [Chv83, Sch86], one example being the *lexicographic rule* applied in the simplex

algorithm. Perturbation methods have also found applications in computational geometry [Ede87].

Let $c(d)$ be the running time of $\mathcal{C}^d$. Since at any level, the number of perturbations that $\mathcal{C}^j$ will have to deal with is $d$, and $d$ is fixed, the running time of $\mathcal{C}^j$ will be the same, asymptotically, whether it deals with a perturbed or an unperturbed problem. As we have seen, $c(d)$ is $O(T \cdot b(d))$, where $b(d)$ is the running time of $\mathcal{B}^d$, and $b(d)$ is $O(c(d-1))$ because $\mathcal{B}^d$ is implemented via three recursive calls to $\mathcal{C}^{d-1}$. Since $b(1) = O(t)$, we conclude that $c(d)$ is $O(T^{d+1})$.

## 3.2 Speeding up the Search

The main bottleneck in algorithm $\mathcal{C}^d$ is the need to apply oracle $\mathcal{B}^d$ to each affine function generated during the simulation of algorithm $\mathcal{A}$. One way to reduce this problem is to arrange things so that by using a small number of oracle calls we are able to resolve a large number of functions. Megiddo [Meg83] proposed a way to do this in the context of one-dimensional problems, an idea that has subsequently been used in multi-dimensional optimization [Coh91, NPT92]. Megiddo's approach is to simulate the execution of a *parallel* algorithm $\mathcal{A}$ for computing $g(\lambda)$ rather than a sequential one. Suppose $\mathcal{A}$ uses $M$ processors and carries out at most $D$ parallel steps. In each step of the simulation, a *batch* of at most $M$ comparisons is carried out. In $\mathcal{C}^d$'s simulation of $\mathcal{A}$, each such comparison has an associated affine function $h$ which can be resolved using $\mathcal{B}^d$. Every parallel step produces a set of $O(M)$ hyperplanes, which can be resolved with $O(\log M)$ oracle calls using Theorem 1. The running time of $\mathcal{C}^d$ is therefore $O(b(d) \cdot D \cdot \log M + T)$, where $b(d)$ is the running time of $\mathcal{B}^d$. Since $\mathcal{B}^d$ is implemented by making at most three recursive calls to $\mathcal{C}^{d-1}$, and $b(1) = O(T)$, the running time of $\mathcal{C}^d$ will be $O((D \log^d M)^d T)$.

Cole [Cole87] showed that one can improve on Megiddo's results for certain important special cases. Like Megiddo's method, Cole's technique applies to one-dimensional parametric search problems, but we shall show that it can be extended to higher dimensions. What follows shall require some elementary knowledge of *combinational circuits* as described, say, in [CLR90]. A *combinational circuit* $\mathcal{G}$ is a directed acyclic graph whose nodes are *combinational elements* (e.g., adders, min gates, etc.), and where an edge from element $e_1$ to element $e_2$ implies that the output of $e_1$ is an input to $e_2$. Elements of zero fan-in are *inputs*; elements of zero fan-out are *outputs*. An element is said to be *active* if all its inputs are known, but the associated operation has not been carried out yet. An element is said to have been *resolved* when the associated operation has been carried out.

Now, suppose that the algorithm $\mathcal{A}$ simulated by $\mathcal{C}^d$ is implemented as a combinational circuit $\mathcal{G}$ of width $M$ and depth $D$, whose elements are min gates, adders, subtractors, and multiplier gates where one of the two inputs is a constant. Megiddo's approach would simulate $\mathcal{G}$ level by level, in $D$ steps, where each step would carry out the operations of the gates at a given level. The operations within a level would be carried out using Theorem 1, with $O(\log M)$) calls to the oracle $\mathcal{B}^d$. In Cole's approach, each step only resolves a fixed fraction of of the active nodes, using only a constant number of oracle calls. The choice

of which nodes to resolve is guided by a weight function $w : V(\mathcal{G}) \rightarrow \mathbf{R}$. To describe the strategy precisely, we will need some notation. The *active weight*, $W$, of the circuit is the sum of the weights of its active elements. Let $\alpha \leq 1/2$ be a positive number. An *$\alpha$-oracle with respect to $w$* — or simply an $\alpha$-oracle — is a procedure that is guaranteed to resolve a set of active elements whose total weight is at least $\alpha W/2$. The following is a restatement and an extension of a result in [Cole87].

**Lemma 5** *Let $\mathcal{G}$ be a combinatorial circuit of width $M$ and depth $D$. Let $d_{min} = \min\{d_I, d_O\}$, where $d_I$ ($d_O$) denotes the maximum fan-in (fan-out) of an element of $\mathcal{G}$. Then, there exists a weight function $w$ such that $\mathcal{G}$ can be evaluated with $O(D \log d_{min} + \log M)$ calls to an $\alpha$-oracle with respect to $w$.*

**Proof:** Let the weight function $w$ be defined as follows. The weight of each output element is 1, and the weight of each internal element is twice the sum of weights of its immediate descendants. Then scale the weights to make the total weight of input elements equal to $M$.

**Lemma 6** *At the start of the $(k+1)^{st}$ iteration, $k \geq 0$, the active weight is at most $(1 - \alpha/2)^k \cdot M$.*

**Proof:** By induction on $k$. The result holds for $k = 0$ since, at the start of the first iteration, only the input elements are active and their total weight is $M$. To prove the inductive step, it suffices to show that at each iteration the active weight is reduced by a factor of at least $\alpha/2$.

Suppose element $e$ is resolved. Then $e$ ceases to be active, but all its descendants may become active. Hence, the resolution of $e$ reduces the active weight by at least $w(e)/2$. Let the active weight of network be $W$. In one step, the $\alpha$-oracle resolves a set of elements whose total weight is at least $\alpha \cdot W$. Thus, in one step, the active weight is reduced from $W$ to $(1 - \alpha/2)W$. □

**Lemma 7** *The weight of any circuit element is at least $(2d_{min})^{-D}$.*

**Proof:** After the initial weight assignment, but prior to scaling, the total weight of the elements at depth $j$ is at most $M(2d_{min})^{D-j}$. Thus, the total weight of the input nodes is at most $M(2d_{min})^D$. Hence the scaling factor is at most $(2d_{min})^D$. Since, prior to scaling, every element has a weight of at least one, after scaling the weight of any circuit element will be at least $(2d_{min})^{-D}$. □

**Lemma 8** *Let $\gamma = c(D \log 2d_{min} + \log M)$, where $c = \lfloor 1 - 1/\log_2(1 - \alpha/2) \rfloor$. Then, there will be no active elements after $k \geq \gamma$ iterations.*

**Proof:** First, observe that

$$\begin{aligned}\gamma &> -\left(D\log 2d_{min} + \log M\right)/\log_2(1-\alpha/2) \\ &= -\left(\log M \cdot (2d_{min})^D\right)/\log_2(1-\alpha/2) \\ &= \log_{(1-\alpha/2)}(M\cdot(2d_{min})^D)^{-1} \qquad (10)\end{aligned}$$

By Lemma 6, the active weight at the start of the $(k+1)^{st}$ iteration is at most $(1-\alpha/2)^k \cdot M$. Using (10) and the fact that $(1-\alpha/2)$ is less that one, we have

$$\begin{aligned}(1-\alpha/2)^k \cdot M &\leq (1-\alpha/2)^\gamma \cdot M \\ &< (1-\alpha/2)^{\log_{(1-\alpha/2)}(M\cdot(2d_{min})^D)^{-1}} \cdot M \\ &= (2d_{min})^{-D}.\end{aligned}$$

As the weight of any element in the circuit is at least $(2d_{min})^{-D}$ and the active weight is strictly less than this weight, there are no active elements after the $k^{th}$ iteration. □

By Lemma 8, there will be no active elements after $c(D\log 2d_{min} + \log M)$ steps, where $c$ depends only on $\alpha$. Thus, $\mathcal{G}$ can be evaluated with $O(D\log d_{min} + \log M)$ calls to an $\alpha$-oracle. □

In order to use Lemma 5, we need to give an efficient implementation of the $\alpha$-oracle. Let $A$ be the set of active elements of $\mathcal{G}$ and let $A_1 \subseteq A$ be the set of adders, subtractors, and constant multipliers. Each $e \in T_1$ can be resolved immediately by simply doing the corresponding operation on the input linear forms. The remaining active elements are comparators, every one of which has an associated affine function. Let the set of all such functions be $\mathcal{H} = \{h_1, \cdots, h_n\}$, where $h_i$ is the function associated with $e_i \in A - A_1$, and assign a weight of $w(e_i)$ to $h_i$. We can resolve a fixed fraction of the functions with $O(1)$ oracle calls using algorithm WEIGHTED-SEARCH of section 2 with one slight modification: If at any point during its execution WEIGHTED-SEARCH encounters a hyperplane $H$ (even an auxiliary one) such that $H \cap \Lambda \neq \emptyset$, it stops and returns $g^*_H$. The running time of the $\alpha$-oracle is therefore $O(b(d))$, where $b(d)$ is the running time of $\mathcal{B}^d$. Thus, Lemma 5 leads to an implementation of $\mathcal{C}^d$ whose running time is $O(b(d)(D+\log M)+D\cdot M)$. As before $\mathcal{B}^d$ is implemented by making at most three recursive calls to $\mathcal{C}^{d-1}$. ¿From this, and the fact that $b(1) = O(T)$, we can deduce that the running time of $\mathcal{C}^d$ is $O((D^d + \log^d M)T)$.

## 3.3 Some Remarks on Constant Factors

The use of multidimensional search schemes seems to lead invariably to large constants that depend on $d$ [Dyer86]. Using standard techniques [Cla86, Dyer86], it can be shown that the algorithms described in this section have hidden constants of the form $2^{O(d^2)}$, provided the search algorithm with singly exponentially small efficiency is used. Some improvements are possible. For the case where all the weights are powers of 1/4, as would occur if the circuit to

be simulated is a comparator-based sorter, we can obtain a search scheme with $\alpha(d) = 1/3$ and $\beta(d) = 2(20^{d-1})$; the details are technical and, hence, omitted. Using this improved scheme, the running time of the optimization algorithm will still have a constant of the form $2^{O(d^2)}$, but the constant inside the $O$ will be smaller.

# 4 Solving the Lagrangian Dual when the Number of Constraints is Fixed

The method of Lagrangian relaxation, originally developed by Held and Karp [HeKa70, HeKa71], is motivated by the observation that many combinatorial problems that are known to be NP-hard can be viewed as easy problems complicated by a relatively small set of side constraints. More formally, we consider optimization problems of the following sort:

$$Z_P = \min\{c^T x : Ax \leq 0, x \in X\}, \tag{11}$$

where $c$ is a $n \times 1$ vector, $A$ is a $d \times n$ matrix, $x$ is a $n \times 1$ vector and $X$ is a polyhedral subset of $\mathbf{R}^d$. The set of inequalities $Ax \leq 0$ constitutes the complicating set of constraints, in the sense that, in its absence, the problem is polynomially solvable.

The *Lagrangian relaxation* of (11) is obtained by pricing out the constraints $Ax \leq 0$ into the objective function by introducing a vector $\lambda = (\lambda_1, \cdots, \lambda_d)$ of Lagrange multipliers as follows:

$$Z_D(\lambda) = \min\{c^T x + \lambda^T Ax : x \in X\}. \tag{12}$$

It is well known that $Z_D(\lambda) \leq Z_P$ for all $\lambda \geq 0$ [Fis81]. Thus, if there is an polynomial-time algorithm to compute $Z_D(\lambda)$ for any fixed $\lambda \geq 0$, problem (12) will provide an efficient way to obtain a lower bound on the solution to (11). Such a bound can be of great utility in branch-and-bound methods. The best lower bound on $Z_P$ attainable via (12) is given by:

$$Z_D^* = \max\{Z_D(\lambda) : \lambda \geq 0\}. \tag{13}$$

Problem (13) is the *Lagrangian dual* of (11) with respect to the set of constraints $Ax \leq 0$ and $Z_D^*$ is the *value* of the Lagrangian dual.

Computational experiments have repeatedly shown that $Z_D^*$ provides excellent lower bounds on the optimum solution of $Z_P$ [Fis81], thus motivating the search for efficient algorithms to solve the Lagrangian dual. One widely-used method is *subgradient optimization*, first proposed in [HeKa71]. Despite its success in practice, this technique is not known to be a polynomial-time algorithm, even if (12) can be solved in polynomial time.

It is well known that if $Z_D(\lambda)$ can be computed in polynomial time for each fixed $\lambda \geq 0$, then the Lagrangian dual can be solved in polynomial time [Sch86]. Recently, Bertsimas and Orlin [BeOr91] have presented faster polynomial time algorithms for certain special cases. An issue that has received some attention [AnKa91] is whether there exist *strongly polynomial* algorithms to solve the Lagrangian dual. (An algorithm is said to be strongly polynomial if the number of arithmetic operations it carries out is polynomially bounded,

independently of the magnitudes of the input numbers.) The algorithms discussed above are not strongly polynomial, even if $Z_D(\lambda)$ can be computed in strongly polynomial time.

We shall be interested here only in the case where the number $d$ of complicating constraints is fixed. Since $Z_D$ is a concave function [Sch86], if $Z_D(\lambda)$ is computable in strongly polynomial time by a piecewise affine algorithm, the results of Megiddo and Cohen described in Section 3 imply the existence of strongly polynomial time algorithms to solve the Lagrangian dual. We focus our attention on two broad families of problems where weighted multidimensional search allows us to obtain faster algorithms that the Megiddo-Cohen approach: matroidal knapsack problems and of a class of constrained optimum subgraph problems on graphs of bounded tree-width.

## 4.1 Matroidal Knapsack Problems

What follows presupposes some familiarity with matroid theory (see, e.g., [Law76]). Consider a matroid $\mathcal{M} = (E, \mathcal{G})$ where $E$, the *ground set*, is a finite set and $\mathcal{G}$ is a collection of certain subsets of $E$ called *independent sets*. We assume that $\mathcal{G}$ is given in a *concise* form; i.e., there is an algorithm with running time $c(n)$, polynomial in $n = |E|$, for finding whether a given subset of $E$ is independent. Suppose each element $e \in E$ has a *value* $v(e)$. In ordinary matroid optimization problems, one must find an optimum *base* (maximal independent set) of maximum total value. The standard algorithm for doing so is the *greedy method*, which first sorts the elements according to value and then considers the elements in nonincreasing order. An element $e$ is added to the current set $A$ if $A \cup \{e\}$ is independent. The greedy algorithm takes time $O(n \log n + nc(n))$.

In *multi-constrained matroidal knapsack (MMK)* problems, in addition to a value, each $e \in E$ has a $d$-dimensional *size* vector $s(e)$ and there is a $d$-dimensional *capacity* vector $C$. The problem is to find a base $G^*$ such that

$$Z^* = \sum_{e \in G^*} v(e) = \max_{G \in \mathcal{G}} \left\{ \sum_{e \in G} v(e) : \sum_{e \in G} s(e) \leq C \right\}$$

We refer the reader to Camerini et. al. [CMV89] for a discussion of the various applications of these problems, as well as for references. MMK problems are in general NP-hard. We can bound $Z^*$ by solving its Lagrangian dual. Let

$$Z_D(\lambda) = \max_{G \in \mathcal{G}} \left\{ \sum_{e \in G} v(e) - \lambda^T (\sum_{e \in G} s(e) - C) \right\}$$

The Lagrangian dual is:

$$Z_D^* = \min\{Z_D(\lambda) : \lambda \geq 0\}. \tag{14}$$

In [CMV89], Camerini et al. outline an algorithm for (14) whose running time is not guaranteed to be polynomial. Noting that the crucial first stage of the greedy method (where all comparisons are done) can be carried out in parallel using a $O(\log n)$-depth, $O(n)$-width sorting circuit [AKS83], we can use the Cohen-Megiddo technique to obtain a

$O((n \log n + n \cdot c(n)) \cdot \log^{2d} n)$ algorithm using the approach outlined in Section 3, with the greedy algorithm playing the role of algorithm $\mathcal{A}$. Using Lemma 5, and the weighted multidimensional search algorithm, we obtain a $O((n \log n + n \cdot c(n)) \cdot \log^{d} n)$ algorithm. We note that if the underlying matroidal problem has a more specialized structure (e.g., if it is the spanning tree problem), even faster algorithms are possible.

## 4.2 Constrained Optimum Subgraph Problems

Optimum subgraph problems have the following form. Given a graph $G$ with real-valued vertex and edge weight functions $w_V : V(G) \rightarrow \mathbf{R}$ and $w_E : E(G) \rightarrow \mathbf{R}$, respectively, find an optimum (i.e., minimum- or maximum-weight) subgraph $H$ satisfying a property $P$. Well-known examples of such problems are minimum-weight dominating set, minimum-weight vertex cover, and the traveling salesman problem. Let us write $\mathsf{val}_G(H)$ to denote $\sum_{v \in V(H)} w_V(v) + \sum_{e \in E(H)} w_E(e)$, where $H$ is a subgraph of $G$. We can express all optimum subgraph problems as

$$z_G^P = \text{opt}\{\mathsf{val}_G(H) : H \text{ a subgraph of } G \text{ satisfying } P\}, \tag{15}$$

where "opt" is either "min" or "max", depending on the problem.

Even though many optimum subgraph problems are known to be NP-complete, several researchers have developed methodologies for devising linear-time algorithms for graphs of *bounded tree-width* [ALS91, ArPr89, Bod87, BPT88, BLW87, Wim87] (for a definition of tree-width, see [RoSe86]). While their approaches differ from each other in several respects, in essence they all deal with subgraph problems that have certain "regularity" properties that make them amenable to dynamic programming solutions. The class of regular problems is broad, and includes the subgraph problems mentioned above, as well as many others, such as the maximum cut problem and the Steiner tree problem (see, e.g., [ALS91, BPT88, BLW87]).

Suppose that, in addition to a weight function, every $v \in V(G)$ ($e \in E(G)$) has a $d$-dimensional size vector $s_V(v)$ ($s_E(e)$). The problem is to solve (15) subject to the knapsack-like constraint

$$\sum_{v \in V(H)} s_V(v) + \sum_{e \in E(H)} s_E(e) \leq t$$

where $t$ is a $d$-dimensional capacity vector. Even if the unconstrained problem is polynomially-solvable, the constrained one may be NP-hard. Such is the case, for example, for the dominating set problem on trees (which are graphs of tree-width 1), even if $d = 1$ [McPe90].

For every $v \in V(G)$, let $W_V(v, \lambda) = w_V(v) + \lambda^T s_V(v)$ and for every $e \in E(G)$, let $W_E(e, \lambda) = w_E(e) + \lambda^T s_E(e)$. Let us write $\mathsf{Val}_G(H, \lambda)$ to denote $\sum_{v \in V(H)} W_V(v, \lambda) + \sum_{e \in E(H)} W_E(e, \lambda)$, where $H$ is a subgraph of $G$. The Lagrangian relaxation of problem (15), subject to the knapsack constraints, is

$$Z_G^P(\lambda) = \text{opt}\{\mathsf{Val}_G(H, \lambda) : H \text{ a subgraph of } G \text{ satisfying } P\}. \tag{16}$$

If property $P$ is regular, there exists a $O(n)$-time algorithm to compute $Z_G^P(\lambda)$ for any fixed $\lambda$. Also, as proved in [FeSl92], there exists a $O(n)$-size, $O(\log n)$-depth combinational circuit

that computes $Z_G^P(\lambda)$ for any fixed $\lambda$. Thus, the results of Cohen and Megiddo summarized in Section 3 imply that the Lagrangian dual can be solved in $O(n \log^{2d} n)$ time. Using weighted multidimensional search and Lemma 5, we can improve this to $O(n \log^d n)$.

# References


[AgFe92] R. Agarwala and D. Fernández-Baca. Solving the Lagrangian dual when the number of constraints is fixed. To appear in *Proceedings of 13th Conference on Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science, 1992.

[AKS83] M. Ajtai, J. Komlós, and E. Szemerédi. A $O(n \log n)$ sorting network. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, pp 1–9, 1983.

[ALS91] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12:308–340.

[AnKa91] Y. P. Aneja and S. N. Kabadi. Polynomial algorithms for lagrangean relaxations in combinatorial problems. Manuscript.

[ArPr89] S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems restricted to partial $k$-trees. *Discr. Appl. Math.*, 23:11–24 (1989).

[BeOr91] D. Bertsimas and J.B. Orlin. A technique for speeding up the solution of the Lagrangean dual. Working Paper WP# 3278-91-MSA, Sloan School of Management, MIT, April 1991. In *Proceedings of IPCO 92.*

[BLW87] M.W. Bern, E.L. Lawler, and A.L. Wong. Linear time computation of optimal subgraphs of decomposable graphs. *J. Algorithms*, 8:216–235, 1987.

[Bod87] H.L. Bodlaender. Dynamic programming on graphs with bounded tree-width. Technical Report RUU-CS-88-4, University of Utrecht, 1988. Extended Abstract in *Proceedings of ICALP88.*

[BPT88] R.B. Borie, R.G. Parker, and C.A. Tovey. Automatic generation of linear-time algorithms ¿from predicate-calculus descriptions of problems on recursively-constructed graph families. Manuscript, 1988. To appear in *Algorithmica.*

[Cla86] K.L. Clarkson. Linear programming in $O(n \times 3^{d^2})$ time. *Information Processing Letters* 22:21–24 (1986).

[CMV89] P.M. Camerini, F. Maffioli, and C. Vercellis. Multi-constrained matroidal knapsack problems. *Mathematical Programming* 45:211–231 (1989).

[CLR90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, MA 1990.

[Chv83] V. Chvátal. *Linear Programming.* W.H. Freeman, San Francisco 1983.

[Coh91] E. Cohen. Combinatorial algorithms for optimization problems. Report No. STAN-CS-91-1366, Department of Computer Science, Stanford University, Stanford, CA 94305.

[CoMe91] E. Cohen and N. Megiddo. Complexity analysis and algorithms for some flow problems. In *Proc. 2nd ACM Symposium on Discrete Algorithms*, pp. 120–130, 1991.

[Cole87] R. Cole. Slowing down sorting networks to obtain faster sorting algorithms. *J. Assoc. Comput. Mach.* 34(1):200–208, 1987.

[Dyer86] M. E. Dyer. On a multidimensional search technique and its application to the Euclidean one–centre problem. *SIAM J. Comput.* 15(3):725–738 (1986).

[Ede87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag, Heidelberg 1987

[FeSl92] D. Fernández-Baca and G. Slutzki. Parametric problems on graphs of bounded tree-width. In *Proceedings of the 3rd Scandinavian Workshop on Algorithm Theory*, Springer-Verlag LNCS, 1992.

[Fis81] M. L. Fisher. The Lagrangian relaxation method for solving integer programming problems. *Management Science* 27(1):1–18, (1981).

[HeKa70] M. Held and R.M. Karp. The traveling salesman problem and minimum spanning trees. *Operations Research* 18:1138–1162 (1970).

[HeKa71] M. Held and R.M. Karp. The traveling salesman problem and minimum spanning trees: part II. *Mathematical Programming* 6:6–25 (1971).

[Law76] E. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart, and Winston, 1976.

[McPe90] J. McHugh and Y. Perl. Best location of service centers in a treelike network under budget constraints. *Discrete Mathematics*, 86:199–214 (1990).

[Meg79] N. Megiddo. Combinatorial optimization with rational objective functions. *Math. Oper. Res.* 4:414–424.

[Meg83] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. Assoc. Comput. Mach.* 30(4):852–865, (1983).

[Meg84] N. Megiddo. Linear programming in linear time when the dimension is fixed. *J. Assoc. Comput. Mach.*, 31:114–127, 1984.

[NPT92] C.H. Norton, S.A. Plotkin, and É. Tardos. Using separation algorithms in fixed dimension. *J. Algorithms* 13:79–98 (1992).

[RoSe86] N. Robertson and P.D. Seymour. Graph minors II: Algorithmic aspects of tree-width. *J. Algorithms* 7:309–322 (1986).

[Sch86] A. Schrijver. *Theory of Linear and Integer Programming.* Wiley, Chichester 1986.

[Sei91] R. Seidel. Small-dimensional linear programming and convex hulls made easy. *Discrete Comput. Geom.* 6:423–434 (1991).

[Tarj83] R.E. Tarjan. *Data Structures and Network Algorithms.* SIAM Press, Philadelphia 1983.

[Wim87] T.V. Wimer. Linear algorithms on $k$-terminal graphs. Ph.D. Thesis, Report No. URI-030, Clemson University (1987).