Optimal Expression Evaluation for Data Parallel Architectures

John R. Gilbert* Robert Schreiber†

August 17, 1994

Abstract

A data parallel machine represents an array or other composite data structure by allocating one processor (at least conceptually) per data item. A pointwise operation can be performed between two such arrays in unit time, provided their corresponding elements are allocated in the same processors.

If the arrays are not aligned in this fashion, the cost of moving one or both of them is part of the cost of the operation. The choice of where to perform the operation then affects this cost. If an expression with several operands is to be evaluated, there may be many choices of where to perform the intermediate operations.

We give an efficient algorithm to find the minimum-cost way to evaluate an expression, for several different data parallel architectures. Our algorithm applies to any architecture in which the metric describing the cost of moving an array has a property we call "robustness." This encompasses most of the common data parallel communication architectures, including meshes of arbitrary dimension and hypercubes. We remark on several variations of the problem, some of which we solve and some of which remain open.

Keywords: data parallel architecture, compilers for parallel computers, Steiner trees.

*Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, California 94304.

†Research Institute for Advanced Computer Science, MS 230-5, NASA Ames Research Center, Moffett Field, CA 94035. This author's work was supported by the NAS Systems Division and DARPA via Cooperative Agreement NCC 2-387 between NASA and the University Space Research Association (USRA).

1 Introduction

1.1 The problem

Massively parallel architectures are an emerging fact in scientific computing. They offer very high peak computation rates but in general they must be carefully programmed to avoid performance bottlenecks that can be caused by poor load balancing or by excessive interprocessor or memory traffic. Several such architectures provide especially fast communication paths between neighboring processors on a grid of one, two, or more dimensions. Commercially available examples include such SIMD machines as the MPP, AMT DAP, Connection Machine, and MasPar MP-1, and such MIMD message-passing machines as those from Intel and Ncube.

We shall be concerned in this paper with the following problem: Given n arrays of data $A_1, \ldots, A_n$, all of the same shape, we wish to evaluate an arithmetic expression involving these arrays, arbitrary binary operations (to be applied elementwise), and parentheses. We assume that the arrays are situated at various places in a multicomputer consisting of processors with local memory, connected in some regular fashion. A given metric describes the cost of moving an array from one position to another within the machine. The problem is to determine the positions at which to carry out the individual operations in the expression.

Figure 1 is an example for a two-dimensional grid of processors. Four arrays w, x, y, and z occupy different parts of a two-dimensional grid of processors. Define the "position" of an array to be the position of its upper left, or $(1,1)$, element. We want to evaluate the expression

$$(w \oplus x) \otimes (y \odot z),$$

where $\oplus$, $\otimes$, and $\odot$ are operations that act elementwise on the array. Suppose that moving an entire array one position north, south, east, or west has unit cost. Then the metric is the two-dimensional l_1 "Manhattan metric." We could for example move all the arrays to the position of array w at a cost of 185 (30 for x, 90 for y, and 65 for z); or we could move w to x's position and perform $\oplus$ (cost 30), move y and z to position $(30,60)$ and perform $\odot$ (cost 80), and move that result to x's position to perform $\otimes$ (cost 35), for a total cost of 145.

In this paper we give an efficient algorithm to find the minimum-cost evaluation procedure for an arbitrary expression, provided the metric satisfies a condition that we call *robustness*. Among the cases in which our

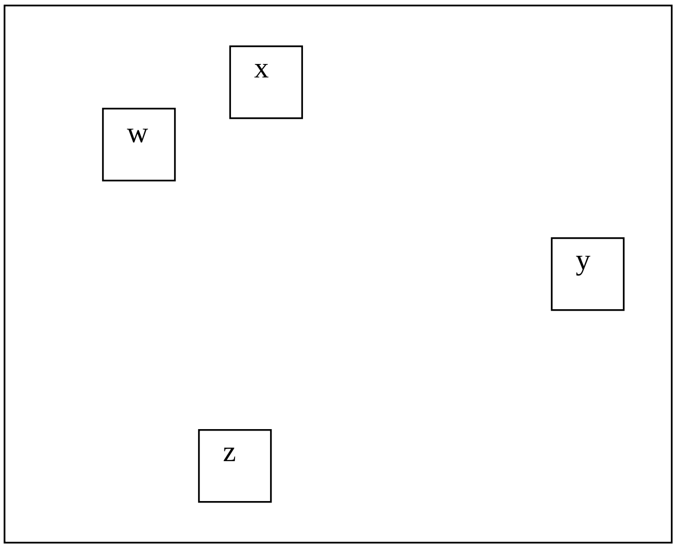

Figure 1: Four arrays are to be combined in the expression $(w \oplus x) \otimes (y \odot z)$. The upper left-hand element of w is at $(10, 80)$, of x at $(30, 90)$, of y at $(80, 60)$, and of z at $(25, 30)$.

results apply are the l_1 metric in any number of dimensions, the l_∞ metric in two dimensions, the discrete metric, and the Hamming distance metric describing shortest distance in a hypercube. We expect that the chief use of these results will be in compiler and run time optimization for massively parallel machines.

This tree embedding problem one of a large class of so-called Steiner problems, which have applications in many areas [8]. The work most closely related to ours is in the context of describing an evolutionary tree in biology. Several specific metrics have been studied, including l_1 [1], the discrete metric [2, 4], and a string mutation distance metric [7]. Our results on robust metrics include those of [1, 2, 4] as special cases.

Several variations of the problem are of interest, including the following.

- A position may be specified in which the final result is to be placed. As described in Section 5, this variant reduces easily to the case where the final result position is free.

- Various metrics are of interest for different architectures. The most realistic metrics include the one-dimensional Euclidean metric, higher-dimensional l_1 and l_∞ metrics, the hypercube metric, and some combinations of these metrics with discrete metrics.

- We could distinguish between arrays stored by rows and arrays stored by columns, and include the cost of any necessary transpositions. More generally we could include the possibility of translating among several alternative representations of a data structure. This does not change the problem; it just makes the metric a bit more complicated.

- We could allow associative and/or commutative rearrangement of the expression tree. As described below, this makes the problem easier in one dimension and harder in higher dimensions.

- We could take common subexpressions into account, possibly even allowing copies of the arrays to be left in strategic positions during a move. This is probably very hard.

1.2 Definitions

We are given a universe P of possible *positions* (corresponding to processors), and a function d that describes the cost of moving data from one position to another. The function d is a metric:

- $d(p,q) = d(q,p) \geq 0$.
- $d(p,q) = 0$ if and only if $p = q$.
- $d(p,q) + d(q,r) \geq d(p,r)$.

If X and Y are sets of positions then $d(X,Y) = \inf\{d(x,y) : x \in X, y \in Y\}$. (If P is finite we can replace inf by min.) The triangle inequality implies that $d(X,p) + d(p,Y) \geq d(X,Y)$ if p is a position and X and Y are sets of positions.

Now we make several definitions that allow us to identify a class of metrics for which we can efficiently find minimum-cost tree embeddings.

The *generalized intersection* of two sets X and Y of positions is defined as

$$X \overset{*}{\cap} Y = \{p \in P \mid d(p,X) + d(p,Y) = d(X,Y)\}.$$

If $X \cap Y$ is nonempty and closed then $X \overset{*}{\cap} Y = X \cap Y$. If X and Y are disjoint intervals on the real line and d is Euclidean distance then $X \overset{*}{\cap} Y$ is the closure of the set of points between X and Y. Figure 2 shows three examples of generalized intersections in the l_1 metric in two dimensions, using as data the positions of the upper left corners of the arrays in Figure 1.

For an arbitrary metric d, we define sets of positions called *generalized intervals* in terms of $\overset{*}{\cap}$ as follows: A generalized interval is either a set containing a single position, or the generalized intersection of two generalized intervals. Thus, for example, the generalized intervals on the Euclidean real line are the nonempty closed bounded intervals. We will need generalized intervals to be nonempty and compact. This is true whenever P is finite (which is the only realistic case), or indeed whenever P is a finite-dimensional complete normed vector space.

A metric d is called *robust* if all generalized intervals are nonempty and compact and

$$d(p,I) + d(p,J) \geq d(p, I \overset{*}{\cap} J) + d(I,J)$$

holds for all positions p and all generalized intervals I and J. For example, the Euclidean metric on the real line is robust. We will see that the Euclidean metric l_2 in two dimensions is not robust, although the l_1 and l_∞ metrics in two dimensions are robust.

The following lemma is easy to verify.

Lemma 1 *The sum of two robust metrics is a robust metric.* ∎

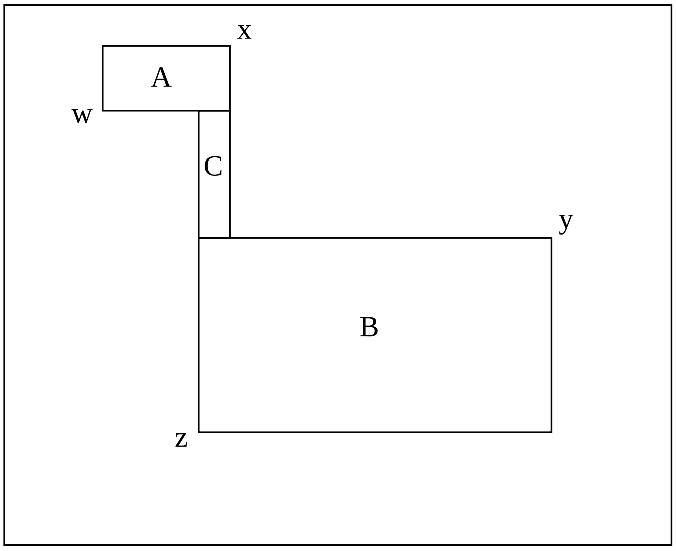

Figure 2: Generalized intersections in the l_1 metric on a two-dimensional grid. Region A is $\{w\} \overset{*}{\cap} \{x\}$, region B is $\{y\} \overset{*}{\cap} \{z\}$, and region C is $A \overset{*}{\cap} B$.

We conclude with some definitions concerning embeddings of a tree into a metric space.

Let T be a rooted binary tree in which each node is either a *leaf* with no children or an *internal node* with two children. We will use lower-case italic $x, y, \ldots$ for nodes.

If x is a node in a tree T, then $T(x)$ is the subtree of T rooted at x. An *embedding* of T is a choice of a position $\pi(x)$ for every node x of T. The cost of an embedding π of T is $\mathrm{cost}(\pi, T)$, which we will write as $\mathrm{cost}(T)$ when the embedding is implicit. It is defined as

$$\mathrm{cost}(\pi, T) = \sum_{\text{edges } \{x,y\} \text{ of } T} d(\pi(x), \pi(y)).$$

In our problems the positions of the leaves of T are fixed. The cost of a minimum-cost embedding of T subject to those fixed leaf positions is $\mathrm{mincost}(T)$. The leaf positions are always fixed in the sequel, and we will not mention the fact explicitly again.

If x is a node of T, we define $\mathrm{Opt}(x)$ as the set of possible values of $\pi(x)$ in minimum-cost embeddings of $T(x)$. That is,

$$\mathrm{Opt}(x) = \{p \in P \mid \exists \text{ a min-cost embedding } \pi \text{ of } T(x) \text{ with } \pi(x) = p\}.$$

Notice that $\mathrm{Opt}(x)$ minimizes only the cost of the subtree rooted at x, not all of T.

2 Robust metrics

In this section we present an efficient algorithm for constructing minimum-cost embeddings for robust metrics. Throughout the section, we consider the variant in which the final result position is free, and commutative and associative rearrangement of the expression are not allowed.

The main theorem about robust metrics says that every subtree has a locally optimal embedding that can be extended to an embedding of the entire tree.

Theorem 2 *Let d be a robust metric on a universe P of positions. Let T be a binary expression tree with positions specified for its leaves. The following are true for any internal node x of T with children y and z.*

1. $\mathrm{Opt}(x) = \mathrm{Opt}(y) \overset{*}{\cap} \mathrm{Opt}(z)$.
2. *Every embedding π of $T(x)$ satisfies*

$$\mathrm{cost}(\pi, T(x)) \geq \mathrm{mincost}(T(x)) + d(\pi(x), \mathrm{Opt}(x)).$$

3. *For all $p \in \mathrm{Opt}(x)$ there is a minimum-cost embedding π of $T(x)$ with $\pi(x) = p$, $\pi(y) \in \mathrm{Opt}(y)$, and $\pi(z) \in \mathrm{Opt}(z)$.*

Proof. We induct on the size of $T(x)$. If the root is a leaf there is nothing to prove. For the inductive step, let x be an internal node with children y and z. Let $G = \mathrm{Opt}(y) \overset{*}{\cap} \mathrm{Opt}(z)$, let $c_y = \mathrm{mincost}(T(y))$, and let $c_z = \mathrm{mincost}(T(z))$. Note that G is nonempty because d is robust and, by the inductive hypothesis, $\mathrm{Opt}(y)$ and $\mathrm{Opt}(z)$ are generalized intervals.

For any embedding π of $T(x)$, we have

$$\mathrm{cost}(T(x)) = d(\pi(x), \pi(y)) + \mathrm{cost}(T(y)) + d(\pi(x), \pi(z)) + \mathrm{cost}(T(z)).$$

Applying part (2) of the inductive hypothesis to $T(y)$ and $T(z)$, and then using the triangle inequality, we get

$$\mathrm{cost}(T(x)) \geq d(\pi(x), \mathrm{Opt}(y)) + c_y + d(\pi(x), \mathrm{Opt}(z)) + c_z.$$

Because d is robust we have $d(\pi(x), \mathrm{Opt}(y)) + d(\pi(x), \mathrm{Opt}(z)) \geq d(\pi(x), G) + d(\mathrm{Opt}(y), \mathrm{Opt}(z))$. Therefore

$$\mathrm{cost}(T(x)) \geq c_y + c_z + d(\mathrm{Opt}(y), \mathrm{Opt}(z)) + d(\pi(x), G). \tag{1}$$

Now consider any $p_x \in G$. Since $\mathrm{Opt}(y)$ and $\mathrm{Opt}(z)$ are compact, there are positions $p_y \in \mathrm{Opt}(y)$ and $p_z \in \mathrm{Opt}(z)$ with $d(p_x, p_y) = d(p_x, \mathrm{Opt}(y))$ and $d(p_x, p_z) = d(p_x, \mathrm{Opt}(z))$. By the induction hypothesis there are embeddings of $T(y)$ and $T(z)$ of cost c_y and c_z with $\pi(y) = p_y$ and $\pi(z) = p_z$. Because $G = \mathrm{Opt}(y) \overset{*}{\cap} \mathrm{Opt}(z)$, we have $d(p_x, \mathrm{Opt}(y)) + d(p_x, \mathrm{Opt}(z)) = d(\mathrm{Opt}(y), \mathrm{Opt}(z))$. If $\pi(x) = p_x$, then, we have

$$\mathrm{cost}(T(x)) = c_y + c_z + d(\mathrm{Opt}(y), \mathrm{Opt}(z)).$$

Inequality (1) says that we can do no better, and that if $\pi(x) \notin G$ then $\mathrm{cost}(T(x)) > c_y + c_z + d(\mathrm{Opt}(y), \mathrm{Opt}(z))$. Therefore $\mathrm{mincost}(T(x)) = c_y + c_z + d(\mathrm{Opt}(y), \mathrm{Opt}(z))$ and $G = \mathrm{Opt}(x)$, proving conclusion (1). Then inequality (1) is exactly conclusion (2). Conclusion (3) follows from the construction, finishing the proof. ∎

We can find a minimum-cost embedding in linear time by using conclusions (1) and (3) of this theorem, provided we can efficiently compute the generalized intersection of two generalized intervals and find the point of a generalized interval closest to a given position.

Here is the algorithm for the general case (assuming a robust metric). The input is a tree T with root r, and specified values $\pi(x)$ for every leaf x. The output is an optimal choice of $\pi(x)$ for every internal node of T. Section 4 describes the algorithm in more detail for some specific metrics.

Step 1. Set $\mathrm{Opt}(x) = \{\pi(x)\}$ for each leaf x.

Step 2. Traverse the tree in postorder, computing $\mathrm{Opt}(x) = \mathrm{Opt}(y) \overset{*}{\cap} \mathrm{Opt}(z)$ for each internal node with children y and z.

Step 3. Set $\pi(r)$ to be any element of $\mathrm{Opt}(r)$.

Step 4. Traverse the tree in preorder, computing π as follows: Place the child y of x at the point of $\mathrm{Opt}(y)$ closest to $\pi(x)$.

3 Specific metrics

Here we consider several possible metrics. Throughout the section, we consider the variant in which the final result position is free, and commutative and associative rearrangement of the expression are not allowed.

If the set of positions is k-dimensional space, the l_p metric (for integer $p > 0$) has

$$d((x_1, \ldots, x_k), (y_1, \ldots, y_k)) = \left(\sum_{1 \leq i \leq k} |x_i - y_i|^p \right)^{1/p}.$$

In the l_∞ metric, the distance between x and y is $\max_i |x_i - y_i|$. In the *discrete metric*, the distance between x and y is 0 if $x = y$, or 1 if $x \neq y$.

One dimension. In one dimension on either the real line or (more realistically) a finite interval of the integers, all the l_p metrics are the same as normal Euclidean distance. This is a realistic metric for one-dimensional processor arrays. As mentioned in Section 2, this metric is robust and the algorithm gives a minimum-cost embedding in time linear in the size of the tree.

The l_1 metric in more than one dimension. This is a realistic metric for a grid of processors with connections to their nearest neighbors. The metric is the sum of the one-dimensional l_1 distances along each coordinate axis, so Lemma 1 implies that it is robust. In fact, the problem separates into an independent one-dimensional Euclidean problem for each coordinate. For fixed dimension, therefore, the optimal layout can be found in linear time. Section 4 gives a detailed algorithm.

The l_∞ metric in more than one dimension. This is a realistic metric for a grid of processors with connections to their nearest neighbors and also to their diagonal neighbors. In two dimensions, for example, this is realistic for a nine-point mesh of processors. The l_∞ metric is robust in two dimensions—in fact it is just a rotation and scaling of the l_1 metric, so minimum-cost embeddings can still be found in linear time. Section 4 gives a detailed algorithm. We do not know whether l_∞ is robust in more than two dimensions or not.

The l_2 metric in more than one dimension. This is normal Euclidean distance, which is not realistic for any existing processor architecture. The metric is not robust in two or more dimensions. For a two-dimensional example, consider $I = \{(0,1)\}$, $J = \{(0,-1)\}$, and $p = (1,0)$. Then $I \overset{*}{\cap} J$ is the closed interval on the y axis from $(0,1)$ to $(0,-1)$. We have $d(p,I) + d(p,J) = 2\sqrt{2} < 3 = d(p, I \overset{*}{\cap} J) + d(I,J)$. This example also shows that Theorem 2 fails to hold for l_2: If y is the parent of leaves fixed at I and J and x is the parent of y and a leaf fixed at p, then $\mathrm{Opt}(y) = I \overset{*}{\cap} J$ but there is no minimum-cost embedding of $T(x)$ with $\pi(y) \in \mathrm{Opt}(y)$. Using a different approach based on work of Melzak [6], Hwang [5] gives a linear-time algorithm to find a minimum-cost embedding in the l_2 metric in two dimensions.

The hypercube metric. Here the positions are vertices of a hypercube and the metric is Hamming distance, or shortest path length in the hypercube. This metric is robust, although representing a generalized interval requires one datum for each dimension so the running time of the algorithm increases with dimension. This problem is just the l_1-metric problem for a space of the dimension of the hypercube. Thus the hypercube-metric problem (without rearrangement) can be solved in $O(nk)$ time, where n is the size of the tree and k is the dimension of the cube.

The discrete metric. This metric is realistic for a single processor if the data structure can be represented in several different ways and there is a fixed cost to translate from one to another. The metric is robust. Every nonempty set is a generalized interval. The generalized intersection $A \overset{*}{\cap} B$ is $A \cap B$ if $A \cap B$ is nonempty, or $A \cup B$ otherwise.

Transposing arrays. As an example of a more complicated metric that is still robust, suppose that arrays may be stored either by rows or by columns. Then some operands must be transposed as well as moved. We can include this possibility in any of these metrics if the cost of transposing simply adds to the cost of moving. The discrete metric with two positions "by rows" and "by columns" is robust, so adding it to a robust metric still gives a robust metric.

The power-of-two-news metric. Here the positions are a k-dimensional grid, and the metric is the length of a shortest path whose step lengths are all powers of two. This metric may be realistic in some cases, such as the "power-of-two news" moves in the Connection Machine. We do not have any results for it.

Toroidal metrics. We could extend the l_1 metrics to the torus, by allowing the extremes of a k-dimensional grid of processors to be connected. We do not have any results for this metric.

String mutation distance. Here the positions are finite strings of symbols over a finite alphabet. The distance between two strings is the smallest number of one-symbol deletions, insertions, and substitutions required to transform one into the other. This metric has nothing to do with parallel computing (presumably), but it has been studied as a model of evolutionary trees of DNA sequences. Sankoff [7] gives an exponential-time algorithm to find an optimal embedding.

4 Algorithms for l_1 and l_∞

For definiteness, we now present explicit pseudocode for the l_1 norm (k-dimensional grids and hypercubes) and the two-dimensional l_∞ norm (nine-point meshes).

The input is the expression tree, represented by a node called *root* which is the top-level operation of the expression, and two node arrays *left-child*(x) and *right-child*(x) that give the two subexpressions combined at node x.

The position of node x is $\pi(x)$, which is a k-vector $(\pi_1(x), \ldots, \pi_k(x))^T$ of coordinates. Initially the positions of the leaves of the tree are given; on output, the positions of all the nodes have been filled in.

The algorithm uses two recursive procedures *find-opt*, which fills in the Opt values from the leaves of the tree up to the root, and *find-pos*, which fills in the positions π from the root down to the leaves.

4.1 Grids and hypercubes

This is the l_1 norm in k dimensions. In this case our algorithm reduces to a method first suggested by Farris [1] in the context of evolutionary trees. We present it here in a somewhat different form.

The Opt values are generalized intervals. In the l_1 norm, these are just k-dimensional rectangles. Such a rectangle is represented by its minimum and maximum coordinate in each dimension. We compute the generalized intersection of two k-dimensional rectangles, $c = a \overset{*}{\cap} b$, as follows. Each dimension of the generalized intersection is computed independently. One dimension of a rectangle is a closed interval on the real line; say dimension i of a is the interval $a_i = [a_i^-, a_i^+]$, and similarly for b and c. If a_i and b_i

overlap then c_i is just their intersection. If a_i and b_i do not overlap, then c_i is the closed interval lying between a_i and b_i. (In every case, c_i^- is the second smallest of the four values $\{a_i^-, a_i^+, b_i^-, b_i^+\}$, and c_i^+ is the second largest.) We omit the code for procedure *general-int*, which does this computation.

In *find-pos*, we need to compute the closest point p of a generalized interval a to a given point $\pi(x)$. Again this done independently in each dimension. In dimension i, if $\pi_i(x)$ lies inside the interval $[a_i^-, a_i^+]$, then $p_i = \pi_i(x)$. If $\pi_i(x)$ lies outside the interval, then p_i is the closer of a_i^- and a_i^+ to $\pi_i(x)$. We omit the code for procedure *closest-point*, which does this computation.

For brevity we assume that all the data structures are global, so the only parameter to the recursive calls is the current node of the tree.

procedure *optimal-evaluation*;
    /* Input is a tree represented by *root*, *left-child*, and *right-child*,
        and positions $\pi(x)$ for the leaves. Output is $\pi(x)$ for all nodes. */
    find-opt(*root*);
    $\pi(root) \leftarrow$ any point in Opt(*root*);
    find-pos(*root*)
    end *optimal-evaluation*;

procedure *find-opt*(**node** x);
    /* Determine the Opt intervals for x and its descendants. */
    if x is a leaf **then**
        $\mathrm{Opt}(x) \leftarrow \{\pi(x)\}$
    else
        $y \leftarrow$ *left-child*(x);
        $z \leftarrow$ *right-child*(x);
        find-opt(y);
        find-opt(z);
        $\mathrm{Opt}(x) \leftarrow$ *general-int*(Opt(y), Opt(z))
    fi **end** *find-opt*;

procedure *find-pos*(**node** x);
    /* Determine the positions for all proper descendants of x. */

/* $\pi(x)$ has already been computed. */
if x is a leaf **then**
    /* Do nothing */
else
    $y \leftarrow$ *left-child*(x);
    $z \leftarrow$ *right-child*(x);
    $\pi(y) \leftarrow$ *closest-point*$(\mathrm{Opt}(y), \pi(x))$;
    $\pi(z) \leftarrow$ *closest-point*$(\mathrm{Opt}(z), \pi(x))$;
    find-pos(y);
    find-pos(z)
fi **end** *find-pos*;

As an example, consider the expression mentioned in Figure 1. Procedure *find-opt* will compute the three rectangles shown in Figure 2 as the Opt values for the three subexpressions. Rectangle C, from $(25, 60)$ to $(30, 80)$, is Opt(*root*), so the root position will be placed arbitrarily in that rectangle. Suppose the root position is placed at $(25, 70)$. Then *find-pos* will place the node for $w \oplus x$ at $(25, 80)$ and the node for $y \odot z$ at $(25, 60)$. The total cost of the moves to those positions is 135.

Procedures *general-int* and *closest-point* each take $O(k)$ time. Procedures *find-opt* and *find-pos* are each called once for each node in the tree. Therefore the entire computation takes $O(kn)$ time for an expression with n operands in k dimensions. This is linear for a fixed-dimensional grid.

4.2 Nine-point meshes

A nine-point mesh in two dimensions represents the l_∞ metric. This metric is just a rotated and scaled version of the l_1 metric. Let A and B be the matrices

$$A = \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}, \qquad B = \frac{1}{2}\begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix},$$

which are inverses of each other. Then the optimum l_∞ positions can be found by applying the l_1 procedure *optimal-evaluation* to leaf positions $\pi'(x) = A\pi(x)$. The node positions $\pi'(x)$ returned are then converted back to the original metric by $\pi(x) = B\pi'(x)$.

Again this whole procedure takes time linear in the number of operands of the expression.

5 Other variations

5.1 Commutative and associative rearrangement: One dimension

For the Euclidean metric in one dimension, allowing commutative and associative rearrangement does not make the problem harder. First, if all the operations in the expression are the same then the problem is trivial.

Lemma 3 *Suppose positions in one dimension are specified for the leaves $x_1, \ldots, x_n$ of an expression consisting entirely of one kind of operation. If associative and commutative rearrangements are allowed, the minimum cost of any embedding of any tree with those leaves is the difference between the positions of the leftmost and rightmost leaves. In a minimum-cost embedding, the root may be placed anywhere in the closed interval between the leftmost and rightmost leaves. If the root is outside that interval by a distance of k, then the cost of the tree is at least k more than optimal.*

Proof. Suppose the leaves are numbered so that $\pi(x_1) \leq \pi(x_2) \leq \cdots \leq \pi(x_n)$. Given a position p such that $\pi(x_i) \leq p \leq \pi(x_{i+1})$, we consider the tree corresponding to the parenthesization

$$(\cdots((x_1 + x_2) + x_3) + \cdots + x_i) + (x_{i+1} + \cdots + (x_{n-2} + (x_{n-1} + x_n))\cdots).$$

We place the root at position p, assign each internal node of the left subtree to the same position as its right child, and assign each internal node of the right subtree to the same position as its left child. The cost of this embedding is $\pi(x_n) - \pi(x_1)$.

This cost is clearly optimal, because some path in the tree must join x_1 and x_n. If the root is outside the interval $[\pi(x_1), \pi(x_n)]$ by k, one of the paths in the tree joining the root to x_1 or x_n must have cost at least $\pi(x_n) - \pi(x_1) + k$. ∎

If the expression has more than one kind of operation in it, we can find a minimum-cost embedding in linear time by partitioning the given expression into connected subtrees with only one kind of operation. Then rearrangement is allowed within these homogeneous subtrees but not between them. We compute $\mathrm{Opt}(x)$ in postorder as usual, except that for each homogeneous subtree we use the lemma to find $\mathrm{Opt}(x)$ for its root x. Then we compute π in preorder, using the lemma to place the internal nodes of homogeneous subtrees.

5.2 Commutative and associative rearrangement: Higher dimensions

The problem becomes much harder in two or more dimensions if rearrangement is allowed. The two-dimensional problem in which all operations are the same is the Geometric Steiner Tree problem [3], which is NP-hard for the l_1, l_2, and l_∞ metrics. Many heuristics have been studied; Winter [8] gives a survey.

5.3 Specifying the position of the result

Consider the variation in which the position of the final result is specified and rearrangement is not allowed. We reduce this to the free-root problem as follows: Suppose the given tree is T with root r, and the final result is constrained to have position p. Let T' be T, plus a new root r' whose children are r and a new leaf x'. In T' we require the same leaf positions as in T, except that the root is unconstrained and we require $\pi(x') = p$. An optimal layout for T' gives an optimal layout for T.

Acknowledgements

We gratefully acknowledge helpful conversations with David Eppstein, who pointed out that the hypercube metric is just l_1, and Marshall Bern, whose encyclopediac knowledge of Steiner problems pointed us to the biological literature on the subject.

References

[1] James S. Farris. Methods for computing Wagner trees. *Systematic Zoology*, 19:83–92, 1970.

[2] Walter M. Fitch. Toward defining the course of evolution: Minimum change for a specific tree topology. *Systematic Zoology*, 20:406–416, 1971.

[3] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

[4] J. A. Hartigan. Minimum mutation fits to a given tree. *Biometrics*, 29:53–65, 1973.

[5] F. K. Hwang. A linear time algorithm for full Steiner trees. *Operations Research Letters*, 4:235–237, 1986.

[6] Z. A. Melzak. On the problem of Steiner. *Canadian Mathematics Bulletin*, 4:143–148, 1961.

[7] David Sankoff. Minimal mutation trees of sequences. *SIAM Journal on Applied Mathematics*, 28:35–42, 1975.

[8] Pawel Winter. Steiner problem in networks: A survey. *Networks*, 17:129–167, 1987.