Order of Magnitude Comparisons of Distance

Ernest Davis*
Courant Institute
New York, New York

December 17, 1997

Abstract

Order of magnitude reasoning — reasoning by rough comparisons of the sizes of quantities — is often called "back of the envelope calculation", with the implication that the calculations are quick though approximate. This paper exhibits an interesting class of constraint sets in which order of magnitude reasoning is demonstrably much faster than ordinary quantitative reasoning. Specifically, we present a polynomial-time algorithm that can solve a set of constraints of the form "Points a and b are much closer together than points c and d." We prove that this algorithm can be applied if "much closer together" is interpreted either as referring to an infinite difference in scale or as referring to a finite difference in scale, as long as the difference in scale is greater than the number of variables in the constraint set. We also prove the first-order theory over such constraints is decidable.

1 Introduction

Order of magnitude reasoning — reasoning by rough comparisons of the sizes of quantities — is often called "back of the envelope calculation", with the implication that the calculations are quick though approximate. Previous AI work on order of magnitude reasoning (e.g. [Raiman, 90], [Mavrovouniotis and Stephanopoulos, 90], [Davis, 90], [Weld, 90]) however, has focussed on its expressive power and inferential structure, not on its computational leverage.

In this paper we exhibit an interesting class of constraint sets in which order of magnitude reasoning is demonstrably much faster than ordinary quantitative reasoning. Specifically, given a set of constraints of the form "Points a and b are much closer together than points c and d," the consistency of such a set can be determined in low-order polynomial time. By contrast, it is easily shown that solving a set of constraints of the form "The distance from a to b is less than or equal to the distance from c to d" in one dimension is NP-complete, and in higher dimensions is as hard as solving an arbitrary set of algebraic constraints over the reals.

In particular, the paper presents the following results:

1. The algorithm "solve_constraints($\mathcal{S}$)" solves a systems of constraints of the form "Points a and b are infinitely closer than points c and d" in polynomial time (Section 4).
2. An improved version of the algorithm runs in time $O(\max(n^2\alpha(n), ne, s)$ where n is the number of variables, e is the number of edges mentioned in the constraint set, and s is the size of the constraint set. (Section 5.1).

*This research has been supported by NSF grant #IRI-9625859.

3. An extended version of the algorithm allows the inclusion of non-strict constraints of the form "Points a and b are not infinitely further apart than points c and d." The running time for this modified algorithm is slower than that of solve_constraints, but still polynomial time. (Section 5.2)

4. The same algorithm can be applied to constraints of the form "The distance from a to b is less than $1/B$ times the distance from c to d," where B is a given finite value, as long as B is greater than the number of variables in the constraint set. (Section 6)

5. The first-order theory over such constraints is decidable. (Section 7)

As preliminary steps, we first discuss order-of-magnitude spaces (section 2) and then present a data structure called a *cluster tree*, which expresses order-of-magnitude distance comparisons (section 3)

2 Order-of-magnitude spaces

An order-of-magnitude space, or *om-space*, is a space of geometric points. Any two points are separated by a distance. Two distances d and e are compared by the relation $d \ll e$, meaning "Distance d is infinitesimal compared to e" or, more loosely, "Distance d is much smaller than e."

For example, let $\Re^*$ be the non-standard real line with infinitesimals. Let $\Re^{*m}$ be the corresponding m-dimensional space. Then we can let a point of the om-space be a point in R^{*m}. The distance between two points a, b is the Euclidean distance, which is a non-negative value in $\Re^*$. The relation $d \ll e$ holds for two distances d, e, if d/e is infinitesimal.

The distance operator and the comparator are related by a number of axioms, specified below. The most interesting of these is called the *om-triangle inequality*: If ab and bc is both much smaller than xy, then ac is much smaller than xy. This combines the ordinary triangle inequality "The distance ac is less than or equal to distance ab plus distance bc" together with the rule from order-of-magnitude algebra, "If $p \ll r$ and $q \ll r$ then $p + q \ll r$."

It will simplify the exposition below if, rather than talking about distances, we talk about orders of magnitude. These are defined as follows. We say that two distances d and e have the *same* order of magnitude if neither $d \ll e$ nor $e \ll d$. In $\Re^*$ this is the condition that d/e is finite: neither infinitesimal nor infinite. (In [Raiman, 90] this is notated "d Co e".) By the rules of the order-of-magnitude calculus, this is an equivalence relation. Hence we can define *an order of magnitude* to be an equivalence class of distances under the relation "same order of magnitude". For two points a, b, we define the function $\mathrm{od}(a, b)$ to be the order of magnitude of the distance from a to b. For two orders of magnitude p, q, we define $p \ll q$ if, for any representatives $d \in p$ and $e \in q$, $d \ll e$. By the rules of the order-of-magnitude calculus, if this holds for any representatives, it holds for all representatives. The advantage of using orders-of-magnitude and the function "od", rather than distances and the distance function, is that it allows us to deal with logical equality rather than the equivalence relation "same order of magnitude".

For example, in the extended real line, let δ be a positive infinitesimal value. Then values such as $\{1, 100, 2 - 50\delta + 100\delta^2 \ldots\}$, are all of the same order of magnitude, $o1$. The values $\{\delta, 1.001\delta, 3\delta + e^{-1/\delta} \ldots\}$ are of a different order of magnitude $o2 \ll o1$. The values $\{1/\delta, 10/\delta + \delta^5 \ldots\}$ are of a third order of magnitude $o3 \gg o1$.

We now give the formal definition:

Definition 1: An *order-of-magnitude space (om-space)* Ω consists of:

- A set of points $\mathcal{P}$;
- A set of orders of magnitude $\mathcal{D}$;
- A distinguished value $0 \in \mathcal{D}$;
- A function "$\mathrm{od}(a, b)$" mapping two points $a, b \in \mathcal{P}$ to an order of magnitude;
- A relation "$d \ll e$" over two orders of magnitude $d, e \in \mathcal{D}$

satisfying the following axioms:

A.1 For any orders of magnitude $d, e \in \mathcal{D}$, exactly one of the following holds: $d \ll e$, $e \ll d$, $d = e$.

A.2 For $d, e, f \in \mathcal{D}$, if $d \ll e$ and $e \ll f$ then $d \ll f$.
(Transitivity. Together with A.1, this means that $\ll$ is a total ordering on orders of magnitude.)

A.3 For any $d \in \mathcal{D}$, not $d \ll 0$.
(0 is the minimal order of magnitude.)

A.4 For points $a, b \in \mathcal{P}$, $\mathrm{od}(a, b) = 0$ if and only if $a = b$.
(The function od is positive definite.)

A.5 For points $a, b \in \mathcal{P}$, $\mathrm{od}(a, b) = \mathrm{od}(b, a)$.
(The function od is symmetric.)

A.6 For points $a, b, c \in \mathcal{P}$, and order of magnitude $d \in \mathcal{D}$,
if $\mathrm{od}(a, b) \ll d$ and $\mathrm{od}(b, c) \ll d$ then $\mathrm{od}(a, c) \ll d$.
(The om-triangle inequality.)

A.7 There are infinitely many different orders of magnitude.

A.8 For any point $a_1 \in \mathcal{P}$ and order of magnitude $d \in \mathcal{D}$, there exists an infinite set $a_2, a_3 \ldots$ such that $\mathrm{od}(a_i, a_j) = d$ for all $i \neq j$.

The example we have given above of an om-space, non-standard Euclidean space, is wild and woolly and hard to conceptualize. Here are two simpler examples of om-spaces:

I. Let δ be an infinitesimal value. We define a point to be a polynomial in δ with integer coefficients, such as $3 + 5\delta - 8\delta^5$. We define an order-of-magnitude to be a power of δ. We define $\delta^m \ll \delta^n$ if $m > n$; for example, $\delta^6 \ll \delta^4$. We define $\mathrm{od}(a, b)$ to be the smallest power of δ in $a - b$. For example, $\mathrm{od}(1 + \delta^2 - 3\delta^3, 1 - 5\delta^2 + 4\delta^4) = \delta^2$.

II. Let N be an infinite value. We define a point to be a polynomial in N with integer coefficients. We define an order of magnitude to be a power of N. We define $N^p \ll N^q$ if $p < q$; for example, $N^4 \ll N^6$. We define $\mathrm{od}(a, b)$ to be the largest power of N in $a - b$. For example, $\mathrm{od}(1 + N^2 - 3N^3, 1 - 5N^2 + 4N^4) = N^4$.

It can be shown that any om-space either contains a subset isomorphic to (I) or a subset isomorphic to (II).

We will use the notation "$d \underline{\ll} e$" as an abbreviation for "$d \ll e$ or $d = e$".

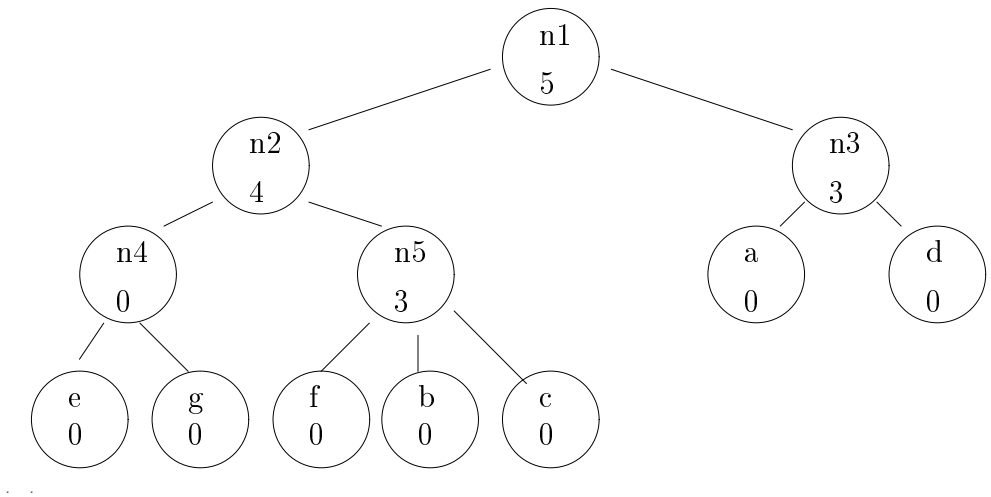

Figure 1: Cluster tree

3 Cluster Trees

Definition 2:

Let P be a finite set of points in an om-space. A non-empty subset $C \subset P$ is called a *cluster* of P if for every $x, y \in C$, $z \in P - C$, $\mathrm{od}(x, y) \ll \mathrm{od}(x, z)$. If C is a cluster, the diameter of C, denoted "odiam(C)" is the maximum value of $\mathrm{od}(x, y)$ for $x, y \in C$.

Note that the set of any single element of P is trivially a cluster of P. The entire set P is likewise a cluster of P. The empty set is by definition not a cluster of P.

Lemma 1: If C and D are clusters of P, then either $C \subset D$, $D \subset C$, or C and D are disjoint.

Proof: Suppose not. Then let $x \in C \cap D$, $y \in C - D$, $z \in D - C$. Since C is a cluster, $\mathrm{od}(x, y) \ll \mathrm{od}(x, z)$. Since D is a cluster, $\mathrm{od}(x, z) \ll \mathrm{od}(x, y)$. Thus we have a contradiction.

By virtue of lemma 1, the clusters of a set P form a tree. We now develop a representation of the o.m. relations in P by constructing a tree whose nodes correspond to the clusters of P, labelled an indication of the relative size of each cluster.

Definition 3: A *cluster tree* is a tree T such that

- Every leaf of T is a distinct symbol.
- Every internal node of T has at least two children.
- Each internal node of T is labelled with a non-negative value. Two or more nodes may be given the same value. (For the purposes of sections 4-6, labels may be taken to be non-negative integers; in section 7, it will be useful to allow rational labels.)
- Every leaf of the tree is labelled 0.
- The label of every internal node in the tree is less than the label of its parent.

For any node N of T, the field "N.symbols" gives the set of symbols in the leaves in the subtree of T rooted at N, and the field "N.label" gives the integer label on node N.

Thus, for example, in figure 1, n3.label=3 and n3.symbols = { a, d }; n1.label = 5 and n1.symbols = $\{a, b, c, d, e, f, g\}$.

As we shall see, the nodes of the tree T represents the clusters of a set of points, and the labels represent the relative sizes of the diameters of the clusters.

Definition 4: A *valuation* over a set of symbols is a function mapping each symbol to a point in an om-space. If T is a cluster tree, a valuation over T is a valuation over T.symbols. If N is any node in T and Γ is a valuation over T, we will write $\Gamma(N)$ as an abbreviation for $\Gamma(N.\text{symbols})$.

We now define how a cluster tree T expresses the o.m. relations over a set of points P.

Definition 5: Let T be a cluster tree and let Γ be a valuation over T. Let $P = \Gamma(T)$, the set of points in the image of T under Γ. We say that $\Gamma \models T$ (read Γ *satisfies* or *instantiates* T) if the following conditions hold:

i. For any internal node N of T, $\Gamma(N)$ is a cluster of P.

ii. For any cluster C of P, there is a node N such that $C = \Gamma(N)$.

iii. For any nodes M and N, if M.label $<$ N.label then $\text{odiam}(\Gamma(M)) \ll \text{odiam}(\Gamma(N))$.

iv. If $\text{label}(M) = 0$, then $\text{odiam}(M) = 0$. (That is, all children of M are assigned the same value under Γ.)

The following algorithm generates an instantiation Γ given a cluster tree T:

procedure instantiate(**in** T : cluster tree; Ω : an om-space)
return : array of points indexed on the symbols of T

variable $G[N]$: array of points indexed on the nodes of T;

Let k be the number of internal nodes in T;
Choose $\delta_0 = 0 \ll \delta_1 \ll \delta_2 \ll \ldots \ll \delta_k$ to be $k+1$ different orders of magnitude;
/* Such values can be chosen by virtue of axiom A.7 */
pick a point $x \in \Omega$;
$G[T] := x$;
instantiate1$(T, \Omega, \delta_1 \ldots \delta_k, G)$;
return the restriction of G to the symbols of T.
end instantiate.

instantiate1(**in** N : a node in a cluster tree; Ω : an om-space; $\delta_1 \ldots \delta_k$: orders of magnitude;
in out G : array of points indexed on the nodes of T)
if N is not a leaf **then**
let $C_1 \ldots C_p$ be the children of N;
$\mathbf{x}_1 := G[N]$;
$q := N$.label;
pick points $x_2 \ldots x_p$ such that
for all $i, j \in 1 \ldots p$, if $i \neq j$ then $\text{od}(x_i, x_j) = \delta_q$;
/* Such points can be chosen by virtue of axiom A.8 */
for $i = 1 \ldots p$ do
$G[C_i] := x_i$;
instantiate1$(C_i, \Omega, \delta_1 \ldots \delta_k, G)$
endfor
endif end instantiate1.

Thus, we begin by picking orders of magnitude corresponding to the values of the labels. We pick an arbitrary point for the root of the tree, and then recur down the nodes of the tree. For each node N, we place the children at points that all lie separated by the desired diameter of N. The final placement of the leaves is then the desired instantiation.

Lemma 2: If T is a cluster tree and Ω is an om-space, then instantiate(T, Ω) returns an instantiation of T.

Proof: Let $\delta_0 = 0$. For any node N, if i=N.label, we define $\Delta(N) = \delta_i$. The proof then proceeds in the following steps:

i. For any nodes M,N, if M is a descendant of N in T then $\mathrm{od}(G[M], G[N]) \underline{\ll} \Delta(N)$. **Proof:** If M is a child of N, then this is immediate from the construction of $x_2 \ldots x_p$ in instantiate1. Else, let $N = N_1, N_2 \ldots N_q = M$ be the path from N to M through T. By the definition of a cluster tree, it follows that N_i.label $<$ N.label, for $i > 1$ and therefore $\Delta(N_i) \ll \Delta(N)$. Thus $\mathrm{od}(G[M], G[N]) \underline{\ll}$ (by the o.m.-triangle inequality) $\max_{i=1\ldots q-1}(\mathrm{od}(G[N_{i+1}], G[N_i]) \underline{\ll} \max_{i=1\ldots q-1}(\Delta(N_i))$ (since N_{i-1} is the child of N_i) $\underline{\ll} \Delta(N)$).

ii. Let N be a node in T; let C_1 and C_2 be two distinct children of N; and let M_1 and M_2 be descendants of C_1 and C_2 respectively. Then $\mathrm{od}(G[M_1], G[M_2])) = \Delta(N)$. **Proof:** By the construction of $x_2 \ldots x_p$ in instantiate1(N), $\mathrm{od}(G[C_1], G[C_2]) = \Delta(N)$. By part (i.), $\mathrm{od}(G[M_1], G[C_1]) \underline{\ll} \Delta(C_1) \ll \Delta(N)$ and likewise $\mathrm{od}(G[M_2], G[C_2]) \ll \Delta(N)$. Hence, by axiom A.6, $\mathrm{od}(G[M_1], G[M_2]) = \Delta(N)$.

iii. Let a and b be any two leaves in T, and let N be the least common ancestor in T of a and b. Then $\mathrm{od}(G[a], G[b]) = \Delta(N)$. **Proof:** Immediate from (ii).

iv. For any node N, $\mathrm{odiam}(\Gamma(N)) = \Delta(N)$. **Proof:** From (iii), any two leaves descending from different children of N are at a distance of order $\Delta(N)$, and no two leaves of N are at a distance of order greater than $\Delta(N)$.

v. For any node N, $\Gamma(N)$ is a cluster of $\Gamma(T)$. **Proof:** Let a and b be leaves of N, and let c be a leaf of $T - N$. Let I be the common ancestor of a and b in T and let J be the common ancestor of a and c. Then I is either N or a descendant of N and J is a proper ancestor of N. Therefore by part (i), $\Delta(I) \ll \Delta(J)$. But by (iii), $\mathrm{od}(\Gamma(a), \Gamma(b)) = \Delta(I) \ll \Delta(J) = \mathrm{od}(\Gamma(a), \Gamma(c))$.

vi. For any internal nodes N, M if M.label $<$ N.label then $\mathrm{odiam}(\Gamma(M)) \ll \mathrm{odiam}(\Gamma(N))$. **Proof:** Immediate from (iv) and the construction of Δ.

∎

Moreover, it is clear that any instantiation Γ of T can be generated in this way (Given an instantiation Γ, just pick $G[N]$ at each stage to be Γ of some symbol of N.)

Two further results will be useful.

Lemma 3: Let T be a cluster tree and let Γ be an instantiation of T. Let a and b be symbols of T. Let N be the least common ancestor of a and b in T. Then $\mathrm{od}(\Gamma(a), \Gamma(b)) = \mathrm{odiam}(\Gamma(N))$.

Proof: Since $\Gamma(a)$ and $\Gamma(b)$ are elements of $\Gamma(N)$, it follows from the definition of odiam that $\mathrm{od}(\Gamma(a), \Gamma(b)) \underline{\ll} \mathrm{odiam}(\Gamma(N))$. Suppose the inequality were strict; that is, $\mathrm{od}(\Gamma(a), \Gamma(b)) \ll \mathrm{odiam}(\Gamma(N))$. Then let C be the set of all the symbols c of T such that $\mathrm{od}(\Gamma(a), \Gamma(c)) \underline{\ll} \mathrm{od}(\Gamma(a), \Gamma(b))$. Then $\mathrm{odiam}(\Gamma(C)) = \mathrm{od}(\Gamma(a), \Gamma(b)) \ll \mathrm{odiam}(\Gamma(N))$. It is easily shown that $\Gamma(C)$ is a cluster in $\Gamma(T)$. Therefore, by property (ii) of definition 5, there must be a node M such that M.symbols $=$ C. Now, M is certainly not an ancestor of N, since $\mathrm{odiam}(\Gamma(M)) \ll \mathrm{odiam}(\Gamma(N))$ but M.symbols

contains both a and b. But this contradicts the assumption that N was the least common ancestor of a and b. ∎

Corollary 4: Let T be a cluster tree and let Γ be an instantiation of T. Let a, b, c, d be symbols of T. Let N be the least common ancestor of c and d in T, and let M be the least common ancestor of a and b in T. Then $\text{od}(\Gamma(a), \Gamma(b)) \ll \text{od}(\Gamma(c), \Gamma(d))$ if and only if $M.\text{label} < N.\text{label}$.

Proof: Immediate from lemma 2 and property (iii) of definition 5 of instantiation.

4 Constraints

We now consider a system $\mathcal{S}$ of constraints of the form $\text{od}(a, b) \ll \text{od}(c, d)$.

Let T be a cluster tree. We will say that $T \vdash \mathcal{S}$ (read "T satisfies $\mathcal{S}$") if every instantiation of T satisfies $\mathcal{S}$. In this section, we develop an algorithm for finding a cluster tree that satisfies a given set of constraints.

The algorithm works along the following lines: Suppose we have a solution satisfying $\mathcal{S}$. Let D be the diameter of the solution. If $\mathcal{S}$ contains a constraint $\text{od}(a, b) \ll \text{od}(c, d)$ then, since $\text{od}(c, d)$ is certainly no more than D, it follows that $\text{od}(a, b)$ is much smaller than D. We label ab as a "short" edge.

If two points u and v are connected by a path of short edges, then by the triangle inequality the edge uv is also short (i.e. much shorter than D). Thus, if we compute the connected components H of all the edges that have been labelled short, then all these edges in H can likewise be labelled short. For example, in table 1, edges vz, wx, and xy can all be labelled "short".

On the other hand, as we shall prove below, if an edge is not in the set H, then there is no reason to believe that it is much shorter than D. We can, in fact, safely posit that it is the same o.m. as D. We label all such edges "long".

We now can assume that any connected component of points connected by short edges is a cluster, and a child of the root of the cluster tree. The root of the cluster tree is then given the largest label. Its children will be given smaller labels. Each "long" edge now connects symbols in two different children of the root. Hence, any instantiation of the tree will make any long edge longer than any short edge.

If no edges are labelled "long" — that is, if H contains the complete graph over the symbols — then there is an inconsistency; all edges are much shorter than the longest edge. For instance, in table 2, since vw, wx, and xy are all much smaller than zy, it follows by the triangle inequality that vy is much smaller than zy. But since we also have the constraints that zy is much smaller than vz and that vz is much smaller than vy, we have an inconsistency.

The algorithm then iterates, at the next smaller scale. Since we have now taken care of all the constraints $\text{od}(a, b) \ll \text{od}(c, d)$, where cd was labelled "long", we can drop all those from $\mathcal{S}$. Let D now be the greatest length of all the edges that remain in $\mathcal{S}$. If a constraint $\text{od}(a, b) \ll \text{od}(c, d)$ is in the new $\mathcal{S}$, then we know that $\text{od}(a, b)$ is much shorter than D, and we label it "short". We continue as above. The algorithm halts when all the constraints in $\mathcal{S}$ have been satisfied, and $\mathcal{S}$ is therefore empty; or when we encounter a contradiction, as above.

We now give the formal statement of this algorithm. The algorithm uses an undirected graph over the variable symbols in $\mathcal{S}$. Given such a graph G, and a constraint C of the form $\text{od}(a, b) \ll \text{od}(c, d)$, we call refer to the edge ab as the "short" of C, and to the edge cd as the "long" of C. The shorts of the system $\mathcal{S}$ is the set of all shorts of the constraints of $\mathcal{S}$ and the longs of $\mathcal{S}$ is the set of all the longs of the constraints. An edge may be both a short and a long of $\mathcal{S}$ if it appears on one

side in one constraint and on the other in another constraint.

procedure solve_constraints(**in** $\mathcal{S}$: a system of constraints of the form $\mathrm{od}(a,b) \ll \mathrm{od}(c,d)$).
return either a cluster tree satisfying $\mathcal{S}$ if $\mathcal{S}$ is consistent;
or false if $\mathcal{S}$ is inconsistent.

type: A node N of the cluster tree contains
pointers to the parent and children of N;
the field N.label, holding the integer label;
and the field N.symbols, holding the list of symbols in the leaves of N.

variables: m is an integer;
C is a constraint in $\mathcal{S}$;
H, I are undirected graphs;
N, M are nodes of T;

begin if $\mathcal{S}$ contains any constraint of the form, "$\mathrm{od}(a,b) \ll \mathrm{od}(c,c)$" **then return false**;
$m :=$ the number of variables in $\mathcal{S}$;
initialize T to contain the root N;
N.symbols:= the variables in $\mathcal{S}$;

repeat $H :=$ the connected components of the shorts of $\mathcal{S}$;
if H contains all the edges in $\mathcal{S}$ **then return(false) endif**;
for each current leaf N of T do
if not all vertices of N are connected in H **then**
N.label $:= m$;
for each connected component I of N.symbols in H **do**
construct node M as a new child of N in T;
M.symbols:= the vertices of I;
endfor endif endfor
$\mathcal{S} :=$ the subset of constraints in $\mathcal{S}$ whose long is in H;
$m := m - 1$;
until $\mathcal{S}$ is empty;

for each leaf N of T
N.label $:= 0$;
if N.symbols has more than one symbol
then create a leaf of N for each symbol in N.symbols;
label each such leaf 0;
endif endfor end solve_constraints.

Tables 1 and 2 give two examples of the working of procedure solve_constraints.

We now prove the correctness of algorithm solve_constraints. We will assume throughout that the two variables in the long of any constraint in $\mathcal{S}$ are distinct.

$\mathcal{S}$ contains the constraints

1. $\mathrm{od}(w, x) \ll \mathrm{od}(v, w)$.
2. $\mathrm{od}(x, y) \ll \mathrm{od}(y, z)$.
3. $\mathrm{od}(v, z) \ll \mathrm{od}(w, y)$.

The algorithm proceeds as follows:
Initialization:
The tree is initializes to a single node with $n1$.
$n1$.symbols:= $\{ v, w, x, y, z \}$

First iteration:
The shorts of $\mathcal{S}$ are $\{ wx, xy, vz \}$.
Computing the transitive closure, H is set to $\{ wx, xy, wy, vz \}$.
$n1$.label := 5;
Two children of $n1$ are created:
$n11$.symbols:= w, x, y
$n12$.symbols:= v, z
As vw is not in H, delete constraint #1 from $\mathcal{S}$.
As yz is not in H, delete constraint #2 from $\mathcal{S}$.
$\mathcal{S}$ now contains just constraint #3.
Second iteration:
The shorts of $\mathcal{S}$ are $\{ vz \}$.
The connected components H is just $\{vz\}$.
$n11$.label := 4;
Three children of $n11$ are created:
$n111$.symbols:= w;
$n112$.symbols:= x;
$n113$.symbols := z;
As wy is not in H, delete constraint #3 from $\mathcal{S}$.
$\mathcal{S}$ is now empty.

Cleanup:
$n12$.label := 0;
Two children of $n12$ are created:
$n121$.symbols:= v;
$n122$.symbols:= z;

(See figure 2.)

Table 1: Example of computing a cluster tree

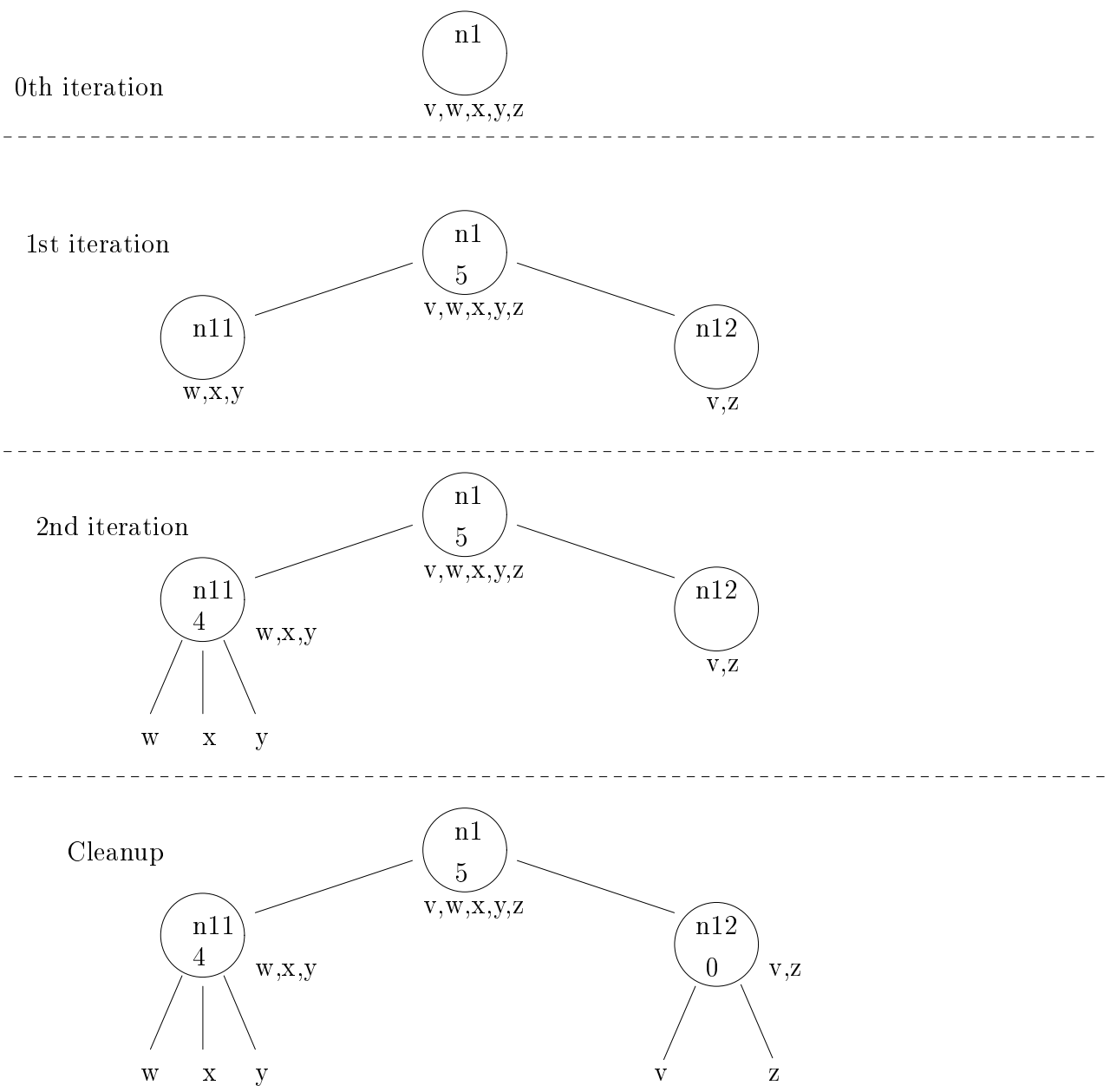

Figure 2: Building a cluster tree

$\mathcal{S}$ contains the constraints

$\mathrm{od}(v,w) \ll \mathrm{od}(z,y)$.
$\mathrm{od}(w,x) \ll \mathrm{od}(z,y)$.
$\mathrm{od}(x,y) \ll \mathrm{od}(z,y)$.
$\mathrm{od}(z,y) \ll \mathrm{od}(v,z)$.
$\mathrm{od}(v,z) \ll \mathrm{od}(v,y)$.

The algorithm proceeds as follows:
Initialization:
The tree is initializes to a single node with $n1$.
$n1$.symbols:= $\{ v, w, x, y, z \}$

First iteration:
The shorts of $\mathcal{S}$ are $\{ vw, vx, xy, zy, vz \}$.
H is set to its connected components, which is the complete graph over v, w, x, y, z.
The algorithm exits returning **false**

Table 2: Example of determining inconsistency

Lemma 5: Let $\mathcal{S}$ be any set of constraints of the form $\mathrm{od}(a,b) \ll \mathrm{od}(c,d)$. Let H be the transitive closure of the shorts of $\mathcal{S}$. If $\mathcal{S}$ is consistent, then not every edge of $\mathcal{S}$ is in H.

Proof: Let Γ be a valuation satisfying $\mathcal{S}$. Find an edge pq in $\mathcal{S}$ for which $\mathrm{od}(\Gamma(p),\Gamma(q))$ is maximal. Now, if ab is a short of $\mathcal{S}$ — that is, there is a constraint $\mathrm{od}(a,b) \ll \mathrm{od}(c,d)$ in $\mathcal{S}$ — then $\mathrm{od}(\Gamma(a),\Gamma(b)) \ll \mathrm{od}(\Gamma(c),\Gamma(d)) \lll \mathrm{od}(\Gamma(p),\Gamma(q))$.

Now, let ab be any edge in H, the transitive closure of the shorts of $\mathcal{S}$. Then there is a path $a_1 = a, a_2 \ldots a_k = b$ such that the edge $a_i a_{i+1}$ is a short of $\mathcal{S}$ for $i = 1 \ldots k-1$. Thus, by the om-triangle inequality, $\mathrm{od}(\Gamma(a),\Gamma(b)) \lll \max_{i=1..k-1}(\mathrm{od}(\Gamma(a_i),\Gamma(a_{i+1})) \ll \mathrm{od}(\Gamma(p),\Gamma(q))$. Hence $pq \neq ab$, so pq is not in H. ∎

Lemma 6: The values of $\mathcal{S}$ and H in any iteration are supersets of their values in any later iteration.

Proof: $\mathcal{S}$ is reset to a subset of itself at the end of each iteration. H is defined in terms of $\mathcal{S}$ in a monotonic manner. ∎

Lemma 7: $\mathcal{S}$ cannot be the same in two successive iterations of the main loop.

Proof: by contradiction. Suppose that $\mathcal{S}$ is the same in two successive iterations. Then H will be the same, since it is defined in terms of $\mathcal{S}$. H is constructed to contain all the shorts of $\mathcal{S}$, Since the resetting of $\mathcal{S}$ at the end of the first iteration does not change $\mathcal{S}$, H must contain all the longs as well. Thus, H contains all the edges in $\mathcal{S}$. But that being the case, the algorithm should have terminated with failure at the beginning of the first iteration. ∎

Lemma 8: Algorithm solve_constraints always terminates.

Proof: By lemma 7, if the algorithm does not exit with failure, then on each iteration some constraints are removed from $\mathcal{S}$. Hence, the number of iterations of the main loop is at most the original size of $\mathcal{S}$. Everything else in the algorithm is clearly bounded. (We will improve on this bound below.) ∎

Lemma 9: If algorithm solve_constraints returns **false**, then $\mathcal{S}$ is inconsistent.

Proof: If the algorithm returns **false**, then the transitive closure of the shorts of $\mathcal{S}$ contains all the

edges in $\mathcal{S}$. By lemma 5, $\mathcal{S}$ is inconsistent.

Lemma 10: If constraint C of form $\mathrm{od}(a,b) \ll \mathrm{od}(c,d)$ is in the initial value of $\mathcal{S}$, and edge cd is in H in some particular iteration, then constraint C is in $\mathcal{S}$ at the start of that iteration.

Proof: Suppose that C is deleted from S on some particular iteration. Then edge cd, the long of C, cannot be in H in that iteration. That is, it is not possible for edge cd to persist in H in an iteration after C has been deleted from $\mathcal{S}$. Note that, by lemma 6, once cd is eliminated from H, it remains out of H. ∎

Lemma 11: The following loop invariant holds: At the end of each loop iteration, the values of L.symbols where L is a leaf in the current state of the tree are exactly the connected components of H.

Proof: In the first iteration, T is initially just the root R, containing all the symbols, and a child of R is created for each connected component of H.

Let T_i and H_i be values of T and H at the end of the ith iteration. Suppose that the invariant holds at the end of the kth iteration. By lemma 6, H_{k+1} is a subset of H_k. Hence, each connected component of H_{k+1} is a subset of a connected component of H_k. Moreover, each connected component J of H_k is either a connected component of H_{k+1} or is partitioned into several connected components of H_{k+1}. In the former case, the leaf of T_k corresponding to J is unchanged and remains a leaf in T_{k+1}. In the latter case, the leaf corresponding to J gets assigned one child for each connected component of H_{k+1} that is a subset of J. Thus, the connected components of H_{k+1} correspond to the leaves of T_{k+1}. ∎

Lemma 12: If procedure solve_constraints does not return **false**, then it returns a well-formed cluster tree T.

Proof: Using lemma 11, and the cleanup section of solve_constraints which creates the final leaves for symbols, it follows that every symbol in $\mathcal{S}$ ends up in a single leaf of T. As m is decremented on each iteration, and as no iteration adds both a new node and children of that node, it follows that the label of each internal node is greater than the label of its father. Hence the constraints on cluster trees (definition 3) are satisfied. ∎

Lemma 13: Let a, b be two distinct symbols in $\mathcal{S}$ and let T be the cluster tree returned by solve_constraints for $\mathcal{S}$. Let N be the least common ancestor of a, b in T. Then either N is assigned its label on the first iteration when the edge ab is not in H, or the edge ab is in the final value of H when the loop is exited and N is assigned its label in the final cleanup section.

Proof: As above, let H_i be the value of H in the ith iteration.

If N is the root, then it is assigned its label in the first iteration. Clearly, a and b, being in different subtrees of N, must be in different connected components of H_1.

Suppose N is assigned its label in the kth iteration of the loop for $k > 1$. By lemma 11, at the end of the previous iteration, N.symbols was a connected component of H_{k-1}, and it therefore contained the edge ab. Since N is the least common ancestor of a, b, it follows that a and b are placed in two different children of N; hence, they are in two different connected components of H_k. Thus the edge ab cannot be in H_k.

Suppose N is assigned its label in the cleanup section of the algorithm. Then by lemma 11, N.symbols is a connected component of the final value of H. Hence the edge ab was in the final value of H. ∎

Lemma 14: Let $\mathcal{S}$ initially contain constraint C of form $\mathrm{od}(a,b) \ll \mathrm{od}(c,d)$. Suppose that solve_constraints($\mathcal{S}$) returns a cluster tree T. Let M be the common ancestor of a, b in T and let N be the common ancestor of c, d. Then M.label $<$ N.label.

Proof: Suppose N is given a label in a given iteration. By lemma 13, cd is eliminated from H in that same iteration. By lemma 10, constraint C must be in $\mathcal{S}$ at the start of the iteration. Hence ab is a short of S in the iteration, and is therefore in H. Hence M is not given a label until a later iteration, and therefore is given a lower label.

It is easily seen that cd cannot be in H in the final iteration of the loop, and hence N is not assigned its label in the cleanup section. ∎

Lemma 15: Suppose that solve_constraints($\mathcal{S}$) returns a cluster tree T. Then any instantiation of T satisfies the constraints $\mathcal{S}$.

Proof: Immediate from lemma 14 and corollary 4.

Theorem 1: The algorithm solve_constraints($\mathcal{S}$) returns a cluster tree satisfying $\mathcal{S}$ if $\mathcal{S}$ is consistent, and returns **false** if $\mathcal{S}$ is inconsistent.

Proof: If solve_constraints($\mathcal{S}$) returns **false**, then it is inconsistent (lemma 9). If it does not return **false**, then it returns a cluster tree T (lemma 12). Since T has an instantiation (lemma 2) and since every instantiation of T is a solution of $\mathcal{S}$ (lemma 15), it follows that $\mathcal{S}$ is consistent and T satisfies $\mathcal{S}$. ∎

There may be many cluster trees that satisfy a given set of constraints. Among these, the cluster tree returned by algorithm solve_constraints has an important property: it has the fewest possible labels consistent with the constraints. In other words, it uses the minimum number of different orders of magnitude of any solution. Therefore, the algorithm can be used to check the satisfiability of a set of constraints in an om-space that violates axiom A.7 and has only finitely many different orders of magnitude. If the algorithm returns T and T has no more different labels than the number of different orders of magnitude in the space, then the constraints are satisfiable. If T uses more labels than the space has orders of magnitude, then the constraints are unsatisfiable.

The proof is easier to present if we rewrite algorithm solve_constraints in the following form, which returns only the number of different non-zero labels used, but does not actually construct the cluster tree.[1]

function num_labels($\mathcal{S}$);
if $\mathcal{S}$ is empty **then return**(0)
else return(1 + num_labels(reduce_constraints($\mathcal{S}$)))

function reduce_constraints($\mathcal{S}$)
H := connected components of the shorts of $\mathcal{S}$;
if H contains all the edges in $\mathcal{S}$ **then return(false)** to top-level
else return(the set of constraints in $\mathcal{S}$ whose long is in H)

It is easily verified that the sequence of values of $\mathcal{S}$ in successive recursive calls to num_labels is the same as the sequence of values of $\mathcal{S}$ in the main loop of solve_constraints. Therefore num_labels returns the number of different non-zero labels in the tree constructed by solve_constraints.

We now show the following results.

Lemma 16: If $\mathcal{S}_1$ and $\mathcal{S}_2$ are consistent sets of constraints, and $\mathcal{S}_1 \supset \mathcal{S}_2$ then reduce_constraints($\mathcal{S}_1$) $\supseteq$ reduce_constraints($\mathcal{S}_2$).

Proof: Immediate by construction. The value of H in the case of $\mathcal{S}_1$ is a superset of its value in the case of $\mathcal{S}_2$, and hence reduce_constraints($\mathcal{S}_1$) is a superset of reduce_constraints($\mathcal{S}_2$).

[1] The reader may wonder why this simpler algorithm was not presented before the more complicated algorithm solve_constraints. The reason is that the only proof we have found that the system of constraints is consistent if num_labels does not return **false** relies on the relation between num_labels and the constructive solve_constraints.

Lemma 17: If $\mathcal{S}_1$ and $\mathcal{S}_2$ are consistent sets of constraints, and $\mathcal{S}_1 \supset \mathcal{S}_2$ then num_labels($\mathcal{S}_1$) $\geq$ num_labels($\mathcal{S}_2$).

Proof by induction on num_labels($\mathcal{S}_2$). If num_labels($\mathcal{S}_2$) = 0, the statement is trivial. Suppose that the statement holds for all $\mathcal{S}'$, where num_labels($\mathcal{S}'_2$) = k, and let num_labels($\mathcal{S}_2$) = k+1. Then k+1 = num_labels($\mathcal{S}_2$) = 1 + num_labels(reduce_constraints($\mathcal{S}_2$)), so k =num_labels(reduce_constraints($\mathcal{S}_2$)). Now, suppose $\mathcal{S}_1 \supset \mathcal{S}_2$. By lemma 16 reduce_constraints($\mathcal{S}_1$) $\supset$ reduce_constraints($\mathcal{S}_2$). But then by the inductive hypothesis num_labels(reduce_constraints($\mathcal{S}_1$)) $\geq$ num_labels(reduce_constraints($\mathcal{S}_2$)), so num_labels($\mathcal{S}_1$) $\geq$ num_labels($\mathcal{S}_2$). ∎

Lemma 18: Let $\mathcal{S}$ be a set of constraints, and let Γ be a solution of $\mathcal{S}$. For any graph G over the symbols of $\mathcal{S}$, let nd(G, Γ) be the number of different non-zero values of od(a, b) where edge ab is in G. Let edges($\mathcal{S}$) be the set of edges in $\mathcal{S}$. Then nd(edges($\mathcal{S}$), Γ) $\geq$ num_labels($\mathcal{S}$).

Proof: by induction on num_labels($\mathcal{S}$). If num_labels($\mathcal{S}$) = 0, then the statement is trivial. Suppose for some k, the statement holds for all $\mathcal{S}'$ where num_labels($\mathcal{S}'$) = k, and suppose num_labels($\mathcal{S}$) = $k+1$. Let pq be the edge in $\mathcal{S}$ of maximal length. For any set of edges E, let small-edges(E, Γ) be the set of all edges ab in E for which od($\Gamma(a), \Gamma(b)$) $\ll$ od($\Gamma(p), \Gamma(q)$). Since small-edges(E) contains edges of every order of magnitude in E except the order of magnitude of pq, it follows that nd(small-edges(E, Γ), Γ) = nd(E, Γ) $-$ 1. Let G be the complete graph over all the symbols in $\mathcal{S}$. By the same argument as in lemma 5, small-edges(G, Γ) $\supset H$, where H is the connected components of the shorts of $\mathcal{S}$, as computed in reduce($\mathcal{S}$). Let $\mathcal{S}'$ be the set of constraints whose longs are in small-edges(G, Γ). It follows that $\mathcal{S}' \supset$ reduce($\mathcal{S}$). Now small-edges(G, Γ) $\supset$ edges($\mathcal{S}'$) $\supset$ edges(reduce($\mathcal{S}$)). Hence nd(edges($\mathcal{S}$), Γ) = nd(G, Γ) = nd(small-edges(G, Γ), Γ) + 1 $\geq$ nd(edges(reduce($\mathcal{S}$))) + 1 $\geq$ (by the inductive hypothesis) nl(reduce($\mathcal{S}$)) + 1 = nl($\mathcal{S}$). ∎

Theorem 2: Out of all solutions to the set of constraints $\mathcal{S}$, the instantiations of solve_constraints($\mathcal{S}$) have the fewest number of different values of od(a, b), where a, b range over the symbols in $\mathcal{S}$.

Proof: Immediate from lemma 18.

Corollary 19: Let Ω have all the properties of an om-space except that it has only k different orders of magnitude. A system of constraints $\mathcal{S}$ has a solution in Ω if and only if the tree returned by solve_constraints($\mathcal{S}$) uses no more than k different labels.

Proof: Immediate from theorems 1 and 2. ∎

5 Extensions and Consequences

5.1 An efficient implementation of solve_constraints

It is possible to implement algorithm solve_constraint somewhat more efficiently than the naive encoding of the above description. The key is to observe that the transitive closure H does not have to be computed explicitly; it suffices to compute it implicitly using merge-find sets (union-find sets). Combining this with suitable back pointers from edges to constraints, we can formulate a more efficient version of the algorithm.

We use the following data structures and subroutines:

- Each node N of the cluster tree contains pointers to its parents and children; a field N.label, holding the integer label; a field N.symbols, holding the list of symbols in the leaves of N; and a field N.mfsets, holding a list of the MFSETS for the symbols in N.
- An edge E in the graph over symbols contains its two endpoints, each of which is a symbol;

a field E.shorts, a list of the constraints in which E appears as a short; and a field E.longs, a list of the constraints in which E appears as a long.

- A constraint C has two fields, C.short and C.long, both of them edges. It also has pointers into the lists C.short.shorts and C.long.longs, enabling C to be removed in constant time from the constraint lists associated with the individual edges.
- We will use the disjoint-set forest implementation of MFSETs [Cormen, Leiserson, and Rivest, 1990, p. 448] with merging smaller sets into larger and path-compression. Thus, each MFSET is a upward-pointing tree of symbols, each node of the tree being a symbol. The tree as a whole is represented by the symbol at the root. A symbol A has then the following fields:
  - A.parent is a pointer to the parent in the MFSET tree.
  - A.cluster_leaf is a pointer to the leaf in the cluster tree containing A.
  - If A is the root of the MFSET then A.size holds the size of the MFSET.
  - If A is the root of the MFSET, then A.symbols holds all the elements of the MFSET.
  - If A is the root of the MFSET then A.leaf_ptr holds a pointer to the pointer to A in N.mfsets where $N = A$.cluster_leaf.

We can now describe the algorithm.

procedure solve_constraints1(**in** $\mathcal{S}$: a system of constraints of the form $\mathrm{od}(a,b) \ll \mathrm{od}(c,d)$).
return either a cluster tree T satisfying $\mathcal{S}$ if $\mathcal{S}$ is consistent;
or false if $\mathcal{S}$ is inconsistent.

Variables: m is an integer;
a, b are symbols;
C is a constraint in $\mathcal{S}$;
H is an undirected graph;
E, F are edges;
P is an MFSET;
N, M are nodes of T;

0. **begin** if $\mathcal{S}$ contains any constraint of the form, "$\mathrm{od}(a,b) \ll \mathrm{od}(c,c)$" **then return false**;
1. $H := \emptyset$;
2. **for** each constraint C in $\mathcal{S}$ with short E and long F **do**
3. add E and F to H;
4. add C to E.shorts and to F.longs **endfor**;
5. $m :=$ the number of variables in $\mathcal{S}$;
6. initialize T to contain the root N;
7. N.symbols := the variables in $\mathcal{S}$;

8. **repeat for** each leaf N of T, INITIALIZE_MFSETS(N);
9. **for** each edge $E = ab$ in H **do**
10. **if** E.shorts is non-empty and FIND(a) $\neq$ FIND(b) **then**
11. MERGE(FIND(a), FIND(b)) **endif endfor**
12. **if every** edge $E = ab$ in H satisfies FIND(a) = FIND(b)
13. **then return(false) endif**
14. **for** each current leaf N of T do
15. **if** N.mfsets has more then one element **then**
16. **for** each mfset P in N.mfsets **do**
17. construct node M as a new child of N in T;
18. M.symbols:= P.symbols;
19. **endfor endif endfor**
20. **for** each edge $E = ab$ in H **do**
21. **if** FIND(a) $\neq$ FIND(b) **then**
22. **for** each constraint C in E.longs **do**
23. delete C from $\mathcal{S}$;
24. delete C from E.longs;
25. delete C from C.short.shorts **endfor**
26. delete E from H **endif endfor**
27. $m := m - 1$;
28. **until** $\mathcal{S}$ is empty;

29. **for** each leaf N of T
30. N.label := 0;
31. **if** N.symbols has more than one symbol
32. **then** create a leaf of N with label 0 for each symbol in N.symbols;
33. **endif endfor end** solve_constraints1.

```
procedure INITIALIZE_MFSETS(N : node)
var A : symbol;
N.mfsets := ∅
for A in N.symbols do
    A.parent := null;
    A.cluster_leaf := N;
    A.symbols := { A };
    A.size := 1;
    N.mfsets := cons(A,N.mfsets);
    A.leaf_ptr := N.mfsets;
endfor end INITIALIZE_MFSETS

procedure MERGE(in A, B : symbol)
if A.size > B.size then swap(A, B);
A.parent := B;
B.size := B.size + A.size;
B.symbols := B.symbols ∪ A.symbols;
Using A.leaf_ptr, delete A from N.mfsets;
end MERGE

procedure FIND(in A : symbol) return symbol;
var R : symbol;
if A.parent = null then return A
    else R := FIND(A.parent);
        A.parent := R; /* Path compression */
        return(R)
end FIND.
```

Let n be the number of symbols in $\mathcal{S}$; let e be the number of edges; and let s be the number of constraints. Note that $n/2 \leq e \leq n(n-1)/2$ and that $e/2 \leq s \leq e(e-1)/2$. The running time of solve_constraint1 can be computed as follows. As each iteration of the main loop 8-28 splits at least one of the connected components of H, there can be at most $n-1$ iterations. The MERGE-FIND operations in the **for** loop 9-11 take together time at most $O(\max(n\alpha(n), e)$ where $\alpha(n)$ is the inverse Ackermann's function. Each iteration of the inner **for** loop lines 16-18 creates one node M of the tree. Therefore, there are only $O(n)$ iterations of this loop over the entire algorithm. Lines 14, 15 of the outer **for** loop require at most n iterations in each iteration of the main loop. The **for** loop 22-26 is executed exactly once in the course of the entire execution of the algorithm for each constraint C, and hence takes at most time $O(s)$ over the entire algorithm. Steps 20-21 require time $O(e)$ in each iteration of the main loop. It is easily verified that the remaining operations in the algorithm take no more time than these. Hence the overall running time is $\mathrm{O}(\max(n^2\alpha(n), ne, s)$.

5.2 Adding non-strict inequalities

The algorithm solve_constraints can be modified to deal non-strict comparisons of the form $\mathrm{od}(a,b) \lessapprox \mathrm{od}(c,d)$ by, intuitively, marking the edge ab as "short" on each iteration if the edge cd has been found to be short.

Specifically, in algorithm solve_constraints, we make the following changes:

- The algorithm takes two parameters: $\mathcal{S}$, the set of strict constraints, and $\mathcal{W}$, the set of non-strict constraints.

- Replace the line

  $H :=$ the connected components of the shorts of $\mathcal{S}$

  with the following code:

  1. $H :=$ the shorts of $\mathcal{S}$;
  2. **repeat** $H :=$ the connected components of H;
  3. **for** each weak constraint $\mathrm{od}(a,b) \lessapprox \mathrm{od}(c,d)$
  4. **if** cd is in H then add ab to H **endif endfor**
  5. **until** no change has been made to H in the last iteration.

The proof that the revised algorithm is only a slight extension of the proof of theorem 1. We need the following new lemmas and proofs:

Lemma 20: Let $\mathcal{S}$ be a set of strict comparisons, and let $\mathcal{W}$ be a set of non-strict comparisons. Let H be the set of edges output by the above code. If $\mathcal{S} \cup \mathcal{W}$ is consistent, then there is an edge in $\mathcal{S}$ that is not in H.

Proof: As in the proof of lemma 5, let Γ be a valuation satisfying $\mathcal{S} \cup \mathcal{W}$ and let pq be an edge in $\mathcal{S}$ such that $\mathrm{od}(\Gamma(p), \Gamma(q))$ is maximal. We wish to show that, for every edge $ab \in H$, $\mathrm{od}(\Gamma(a), \Gamma(b)) \ll \mathrm{od}(\Gamma(p), \Gamma(q))$, and hence $ab \neq pq$. Proof by induction: suppose that this holds for all the edges in H at some point in the code, and that ab is now to be added to H. There are three cases to consider.

- ab is added in step [1]. Then, as in lemma 5, there is a constraint $\mathrm{od}(a,b) \ll \mathrm{od}(c,d)$ in $\mathcal{S}$. Hence $\mathrm{od}(\Gamma(a), \Gamma(b)) \ll \mathrm{od}(\Gamma(c), \Gamma(d)) \lessapprox \mathrm{od}(\Gamma(p), \Gamma(q))$.
- ab is added in step [2]. Then there is a path $a_1 = a, a_2 \ldots a_k = b$ such that the edge $a_i a_{i+1}$ is in H for $i = 1 \ldots k-1$. By the inductive hypothesis, $\mathrm{od}(a_i, a_{i+1}) \ll \mathrm{od}(p,q)$. By the om-triangle inequality, $\mathrm{od}(\Gamma(a), \Gamma(b)) \lessapprox \max_{i=1..k-1}(\mathrm{od}(\Gamma(a_i), \Gamma(a_{i+1})) \ll \mathrm{od}(\Gamma(p), \Gamma(q))$.
- ab is added in step [4]. Then there is a constraint $\mathrm{od}(a,b) \lessapprox \mathrm{od}(c,d)$ in $\mathcal{W}$ such that cd is in H. By the inductive hypothesis, $\mathrm{od}(\Gamma(c), \Gamma(d)) \ll \mathrm{od}(\Gamma(p), \Gamma(q))$.

∎

Lemma 21: Let $\mathcal{W}$ contain the constraint $\mathrm{od}(a,b) \lessapprox \mathrm{od}(c,d)$. Suppose that the algorithm returns a cluster tree T. Let M be the least common ancestor of a and b in T, and let N be the least common ancestor of c and d. Then $\mathrm{label}(M) \leq \mathrm{label}(N)$.

Proof: By lemma 13, N is assigned a label in the first iteration where H does not include the edge cd. In all previous iterations, since cd is in H, ab will likewise be put into H. Hence M does not get assigned a label before N, so $\mathrm{label}(M) \leq \mathrm{label}(N)$.

The remainder of the proof of the correctness of the revised algorithm is exactly the same as the proof of theorem 1.

Optimizing this algorithm for efficiency is a little involved, not only because of the new operations that must be included, but also because there are now four parameters — n, the number of symbols; e, the number of edges mentioned; s, the number of strict comparison; and w, the number of non-strict comparisons — and the optimal implementation varies depending on their relative sizes. In particular, either s or w, though not both, may be much smaller than n, and each of these cases requires special treatment for optimal efficiency. The best implementation we have found for

the case where both s and w are $\Omega(n)$ has a running time of $O(\max(n^3, nw, s))$. The details of the implementation are straightforward and not of sufficient interest to be worth elaborating here.

An immediate consequence of this result is that a couple of problems of inference are easily computed:

- To determine whether a constraint C is the consequence of a set of constraints $\mathcal{S}$, form the set $\mathcal{S} \cup \neg C$ and check for consistency. If $\mathcal{S} \cup \neg C$ is inconsistent then $\mathcal{S} \models C$. Note that the negation of the constraint $\mathrm{od}(a,b) \ll \mathrm{od}(c,d)$ is the constraint $\mathrm{od}(c,d) \underline{\ll} \mathrm{od}(a,b)$.
- To determine whether two sets of constraints are logically equivalent, check that each constraint in the first is a consequence of the second, and vice versa.

6 Finite order of magnitude comparison

In this section, it is demonstrated that algorithm solve_constraints can be applied to systems of constraints of the form "$\mathrm{dist}(a,b) < \mathrm{dist}(c,d)$ / B" for finite B in ordinary Euclidean space as long as the number of symbols in the constraint network is smaller than B.

We could be sure immediately that some such result must apply for finite B. It is a fundamental property of the non-standard real line that any sentence in the first-order theory of the reals that holds for all infinite values holds for any sufficiently large finite value, and that any sentence that holds for some infinite value holds for arbitrarily large finite values. Hence, since the answer given by algorithm solve_constraints works over a set of constraints $\mathcal{S}$ when the constraint "$\mathrm{od}(a,b) \ll \mathrm{od}(c,d)$" is interpreted as "$\mathrm{od}(a,b) < \mathrm{od}(c,d)/B$ for infinite B", the same answer must be valid for sufficiently large finite B. What is interesting is that we can find a simple characterization of B in terms of $\mathcal{S}$; namely, that B is larger than the number of symbols in $\mathcal{S}$.

The proof is not very different from the proof of theorem 1. First, to avoid confusion, we will use a four-place predicate "$\mathrm{much_closer}(a,b,c,d)$" rather than the form "$\mathrm{od}(a,b) \ll \mathrm{od}(c,d)$" as we are not going to give an interpretation to "od" as a function. We fix a finite value $B > 1$, and interpret "$\mathrm{much_closer}(a,b,c,d)$" to mean "$\mathrm{dist}(a,b) < \mathrm{dist}(c,d)$ / B."

We next redefine what it means for a valuation to instantiate a cluster tree:

Definition 6: Let T be a cluster tree and let Γ be a valuation on the symbols in T. We say that $\Gamma \vdash T$ if the following holds: For any symbols a, b, c, d in T, let M be the least common ancestor of a, b and let N be the least common ancestor of c, d. If $M.\mathrm{label} < N.\mathrm{label}$ then $\mathrm{much_closer}(a,b,c,d)$.

We next prove the analogue of Lemma 2:

Lemma 22: Any cluster tree T has an instantiation in Euclidean space $\Re^m$ of any dimensionality m.

Proof: Let n be the number of symbols in T.

We modify procedure instantiate as follows:

procedure instantiate(**in** T : cluster tree; Ω : an om-space; B : real);
return : array of points indexed on the symbols of T;

Let n be the number of nodes in T;
$\alpha := 2 + 2n + Bn$;
Choose $\delta_1, \delta_2 \ldots \delta_n$ such that $\delta_i < \delta_{i+1}/\alpha$;
pick a point $x \in \Omega$;
$G[T] := x$;
instantiate1$(T, \Omega, \delta_1 \ldots \delta_n, G)$;
return the restriction of G to the symbols of T.
end instantiate.

instantiate1(**in** N : a node in a cluster tree; Ω : an om-space; $\delta_1 \ldots \delta_n$: orders of magnitude;
in out G : array of points indexed on the nodes of T)
if N is not a leaf **then**
let $C_1 \ldots C_p$ be the children of N;
$\mathbf{x}_1 := G(N)$;
$q := N$.label;
pick points $x_2 \ldots x_p$ such that
for all $i, j \in 1 \ldots p$, if $i \neq j$ then $\delta_q \leq \text{dist}(x_i, x_j) < n\delta_q$
/* This is possible since $p \leq n$. */
for $i = 1 \ldots p$ do
$G(C_i) := x_i$;
instantiate1$(C_i, \Omega, \delta_1 \ldots \delta_n, G)$
endfor
endif end instantiate1.

The proof that this returns an instantiation of T is essentially the same the proof of Lemma 2, except that we now have to keep track of real quantities. For any node N, if i=N.label, we define $\Delta(N) = \delta_i$. The proof then proceeds in the following steps:

i. For any $i < j$, $\delta_i < \delta_j/\alpha^{j-i}$. Immediate by construction.

ii. For any nodes M,C, if M is a descendant of C in T then $\text{dist}(G[M], G[C]) < \alpha n \Delta(C)/(\alpha - 1)$.
Proof: Let $C = C_0, C_1 \ldots C_r = M$ be the path from C to M through T. Then
$\text{dist}(G[M], G[C]) \leq$ (by the triangle inequality) $\sum_{i=0\ldots r-1} \text{dist}(G[C_{i+1}], G[C_i]) \leq$
$\sum_{i=0\ldots r-1}(n\Delta(C)/\alpha^i) < (\alpha/\alpha - 1)(n\Delta(C))$.

iii. Let N be a node in T; let C_1 and C_2 be two children of N; and let M_1 and M_2 be descendants of C_1 and C_2 respectively. Then
$\Delta(N)(1 - 2n/(\alpha - 1)) < \text{dist}(G[M_1], G[M_2])) < n\Delta(N)(1 + 2/(\alpha - 1))$
Proof: By the triangle inequality,
$\text{dist}(G[C_1], G[C_2]) \leq \text{dist}(G[C_1], G[M_1]) + \text{dist}(G[M_1], G[M_2]) + \text{dist}(G[M_2], G[C_2])$.
Thus, $\text{dist}(G[C_1], G[C_2]) - \text{dist}(G[C_1], G[M_1]) - \text{dist}(G[M_2], G[C_2]) \leq \text{dist}(G[M_1], G[M_2])$.
Also, by the triangle inequality,
$\text{dist}(G[M_1], G[M_2]) \leq \text{dist}(G[C_1], G[C_2]) + \text{dist}(G[C_1], G[M_1]) + \text{dist}(G[M_2], G[C_2])$.
By construction, $\Delta(N) < \text{dist}(G[C_1], G[C_2]) < n\Delta(N)$,
and by part (ii), for $i = 1, 2$, $\text{dist}(G[M_i], G[C_i]) < \alpha n \Delta(C)/(\alpha - 1) < n\Delta(N)/(\alpha - 1)$
as $\Delta(C) < \Delta(N)/\alpha$.

iv. For any symbols a, b, c, d in T, let P be the least common ancestor of a, b and let N be the least common ancestor of c, d. If P.label $<$ N.label then much_closer$(G[a], G[b], G[c], G[d])$.

Proof: By part (iii), $\mathrm{dist}(G[a], G[b]) < n\Delta(P)(1 + 2/(\alpha - 1))$
and $\mathrm{dist}(G[c], G[d]) > \Delta(N)(1-2n/(\alpha-1))$. Since $\Delta(P) < \Delta(N)/\alpha$ and since $\alpha = 2+2n+Bn$, it follows by straightforward algebra that $\mathrm{dist}(G[a], G[b]) < \mathrm{dist}(G[c], G[d])\ /\ B$.

∎

We next prove the analogue of lemma 5.

Lemma 23: Let $\mathcal{S}$ be a set of constraints over n variables of the form "$\mathrm{dist}(a, b) < \mathrm{dist}(c, d)\ /\ B$", where $B > n$. If $\mathcal{S}$ is consistent, then there is some edge in $\mathcal{S}$ which is not in the transitive closure of the shorts of $\mathcal{S}$.

Proof: Let Γ be a valuation satisfying $\mathcal{S}$. Let pq be the edge in $\mathcal{S}$ for which $\mathrm{dist}(\Gamma(p), \Gamma(q))$ is maximal. Now, if ab is a short of $\mathcal{S}$ — that is, there is a constraint $\mathrm{much_closer}(a, b, c, d)$ in $\mathcal{S}$ — then $\mathrm{dist}(\Gamma(a), \Gamma(b)) < \mathrm{dist}(\Gamma(c), \Gamma(d))/B \leq \mathrm{dist}(\Gamma(p), \Gamma(q))/B$.

Now, let ab be any edge in H, the transitive closure of the shorts of $\mathcal{S}$. Then there is a simple path $a_1 = a, a_2 \ldots a_k = b$ such that the edge $a_i a_{i+1}$ is a short of $\mathcal{S}$ for $i = 1 \ldots k-1$. Note that $k \leq n$. Then, by the triangle inequality,

$$\mathrm{dist}(\Gamma(a), \Gamma(b)) \leq \mathrm{dist}(\Gamma(a_1), \Gamma(a_2)) + \mathrm{dist}(\Gamma(a_2), \Gamma(a_3)) + \ldots + \mathrm{dist}(\Gamma(a_{k-1}), \Gamma(a_k)) \leq (k-1)\mathrm{dist}(\Gamma(p), \Gamma(q))\ /\ B < \mathrm{dist}(\Gamma(p), \Gamma(q))$$

Hence $pq \neq ab$, so pq is not in H. ∎

Theorem 3: Let $\mathcal{S}$ be a set of constraints over n variables of the form "$\mathrm{dist}(a, b) < \mathrm{dist}(c, d)\ /\ B$", where $B > n$. The algorithm solve_constraints($\mathcal{S}$) returns a cluster tree satisfying $\mathcal{S}$ if $\mathcal{S}$ is consistent over Euclidean space, and returns **false** if $\mathcal{S}$ is inconsistent.

Proof: Note that the semantics of the constraints "$\mathrm{much_closer}(a, b, c, d)$" enters into the proof of Theorem 1 only in lemmas 2 and 5. The remainder of the proof of Theorem 1 has to do purely with the relation between the structure of $\mathcal{S}$ and the structure of the tree. Hence, since we have shown that the analogues of lemmas 2 and 5 hold in a set of constraints of this kind, the same proof can be completed in exactly the same way. ∎

It may be noted that the proof of lemma 22 does not depend on any relation between n and B. Therefore, if solve_constraints($\mathcal{S}$) returns a tree T, then $\mathcal{S}$ is consistent and T satisfies $\mathcal{S}$ regardless of the relation between n and B. However, it is possible for $\mathcal{S}$ to be consistent and solve_constraints($\mathcal{S}$) to return **false** if $n > B$. On the other hand, the proof of lemma 23 holds in any metric space. Hence, if $B > n$ and solve_constraints($\mathcal{S}$) returns **false** then $\mathcal{S}$ is inconsistent in any metric space. However, there are metric spaces other then $\Re^m$ in which the cluster tree returned by solve_constraints may have no instantiation.

7 The first-order theory

In this section, we show that if the om-space is rich enough then the full first-order language of order-of-magnitude distance comparisons is decidable. Specifically, if the collection of orders of magnitude is dense and unbounded above, then there is a decision algorithm for first-order sentences over the formula, "$\mathrm{od}(W, X) \ll \mathrm{od}(Y, Z)$" that runs in time $O(4^n (n!)^2 s)$ where n is the number of variables in the sentence and s is the length of the sentence.

The basic reason for this is the following: As we have observed in corollary 4, a cluster tree T determines the truth value of all constraints of the form "$\mathrm{od}(a, b) \ll \mathrm{od}(c, d)$" where a, b, c, d are symbols in the tree. That is, any two instantiations of T in any two om-spaces agree on any such

constraint. If we further require that the om-spaces are dense and unbounded, then a much stronger statement holds: Any two instantiations of T over such om-spaces agree on *any* first-order formula free in the symbols of T over the relation "$\mathrm{od}(W,X) \ll \mathrm{od}(Y,Z)$". Hence, it suffices to check the truth of a sentence over all possible cluster trees on the variables in the sentence. Since there are only finitely many cluster trees over a fixed set of variables, this is a decidable procedure.

The result that any such sentence ϕ is decidable could also have been obtained as follows: Let $\alpha(B)$ be the formula open in B that results from replacing every formula inside ϕ of the form "$\mathrm{od}(a,b) \ll \mathrm{od}(c,d)$" by the formula "$\mathrm{dist}(a,b) < \mathrm{dist}(c,d)/B$". Let β be the sentence, "For arbitrarily large B, $\alpha(B)$." Then, by the correspondence between non-standard and standard analysis, β is true over the ordinary real numbers if and only if ϕ is true over the hyper-real numbers. We can then apply Tarski's theorem [1951] to show that β is decidable. In this way we can show the much stronger result that the entire first-order language with addition, multiplication, $<$, and $\ll$ is decidable over the hyperreal numbers.

The much more limited proof we shall give here, however, has three advantages: (a) It is much simpler. (b) It gives a much better running time. (c) It shows that the truth of a formula in this language is the same over *any* dense, unbounded om-space.

Let $\mathcal{L}$ be the first-order language with equality with the single predicate symbol "$\mathrm{much_closer}(a,b,c,d)$" and no function or constant symbols. We will assume without loss of generality that the sentences in $\mathcal{L}$ have been transformed so that the only logical symbols used are $\neg$ (not), $\wedge$ (and) $\exists$ (exists), and quantified variables, and so that no sentence uses the same variable symbol with two different quantifiers. It is easily shown that $\mathcal{L}$ is as expressive as a language with the function symbol "od" and the relation symbol $\ll$.

Definition 7: An om-space Ω with orders of magnitude $\mathcal{D}$ is *dense* if it satisfies the following axiom:

A.9 For all orders of magnitude $\delta_1 \ll \delta_3$ in $\mathcal{D}$, there exists a order of magnitude δ_2 in $\mathcal{D}$ such that $\delta_1 \ll \delta_2 \ll \delta_3$.

Ω is *unbounded above* if it satisfies the following:

A.10 For every order of magnitude δ_1 in $\mathcal{D}$ there exists δ_2 in $\mathcal{D}$ such that $\delta_1 \ll \delta_2$.

If $\mathcal{D}$ is the collection of orders of magnitude in the hyperreal line, then both of these are satisfied. In axiom [A.9], if $0 \ll \delta_1 \ll \delta_3$, choose $\delta_2 = \sqrt{\delta_1 \delta_3}$, the geometric mean. If $0 = \delta_1 \ll \delta_3$, choose $\delta_2 = \delta_3 \delta$ where $\delta \ll 1$. In axiom [A.10] choose $\delta_2 = \delta_1/\delta$ where $\delta \ll 1$.

Definition 8: Let T be a cluster tree. Let $l_0 = 0, l_1, l_2 \ldots l_k$ be the distinct labels in T in ascending order. An *extending label* for T is either (a) l_i for some i; (b) $l_k + 1$ (note that l_k is the label of the root); (c) $(l_{i-1} + l_i)/2$ for some i between 1 and k.

Note that if T has n distinct labels, then there are at most $2n+2$ different extending labels for T.

Definition 9: Let T be a non-empty cluster tree with root R. Let x be a symbol not in T. The cluster tree T' *extends* T *with* x if T' is formed from T by applying one of the following operations (a single application of a single operation).

1. T is the null tree and T' is the tree containing the single node x.
2. T consists of the single node for symbol y. Make a new node M, make both x and y children of M, and set the label of M to be either 0 or 1.

3. For any internal node N of T (including the root), make x a child of N.

4. Let y be a symbol in T, and let N be its father. If N.label $\neq 0$, create a new node M with an extending label for T such that M.label $< N$.label. Make M a child of N, and make x and y children of M.

5. Let C be an internal node of T and let N be its father. Create a new node M with an extending label for T such that C.label $< M$.label $< N$.label. Make M a child of N and make x and C children of M.

6. Let R be the root of T. Create a new node M such that M.label $= R$.label $+ 1$. Make R and x children of M. Thus M is the root of the new tree T'.

(See figure 3)

Note that if T is a tree of n symbols and at most $n-1$ internal nodes then

- There are $n-1$ ways to carry out step 3.
- There are n possible ways to choose symbol y in step 4, and at most $2n$ for the label on M in each.
- There are at most $n-1$ different choices for C in step 5, and at most $2n$ choices for the label on M in each.
- There is only one way to carry out step 6.

Hence, there are less than $4n^2$ different extensions of T by x. (This is almost certainly an overestimate by at least a factor of 2, but the final algorithm is so entirely impractical that it is not worthwhile being more precise.)

Lemma 24: Let T be a cluster tree and let Γ be a valuation over om-space Ω satisfying T. Let x be a symbol not in T, let a be a point in Ω, and let Γ' be the valuation $\Gamma \cup \{x \to a\}$. Let T' be the cluster tree corresponding to Γ'. Then T' is an extension of T by x.

Proof: If T is the empty tree, the statement is trivial. If T contains the single symbol y, then if $a = \Gamma(y)$ then operation (2) applies with M.label=0; if $a \neq \Gamma(y)$ then operation (2) applies with M.label=1.

Otherwise, let y be the symbol in T such that $\mathrm{od}(\Gamma(y), a)$ is minimal. (We will deal with the case of ties in step (D) below.) Let F be the father of y in T.

Let D=od$(\Gamma(y), a)$. Let V be the set of all orders of magnitude od$(\Gamma(p), \Gamma(q))$, where p and q range over symbols in T. We define L to be the *suitable label for* D as follows: If $D \in V$, then L is the label in T corresponding to D. If D is larger than any value in V then L is the label of the root of T plus 1. If $D \notin V$, but some value in V is larger than D, then let D_1 be the largest value in V less than D; let D_2 be the smallest value in V greater than D; let L_1, L_2 be the labels in T corresponding to D_1, D_2; and let $L = (L_1 + L_2)/2$.

One of the following must hold:

A. $\Gamma(y) = a$, and F.label=0. Then apply operation (3) with $N = F$.

B. $\Gamma(y) = a$ and F.label $\neq 0$. Then apply operation (4) with M.label $= 0$.

C. $\Gamma(y) \neq a$, but od$(\Gamma(y), a)$ is less than od$(\Gamma(z), a)$ for any other symbol $z \neq y$ in T. Apply operation (4) with M.label set to the suitable value for D in T.

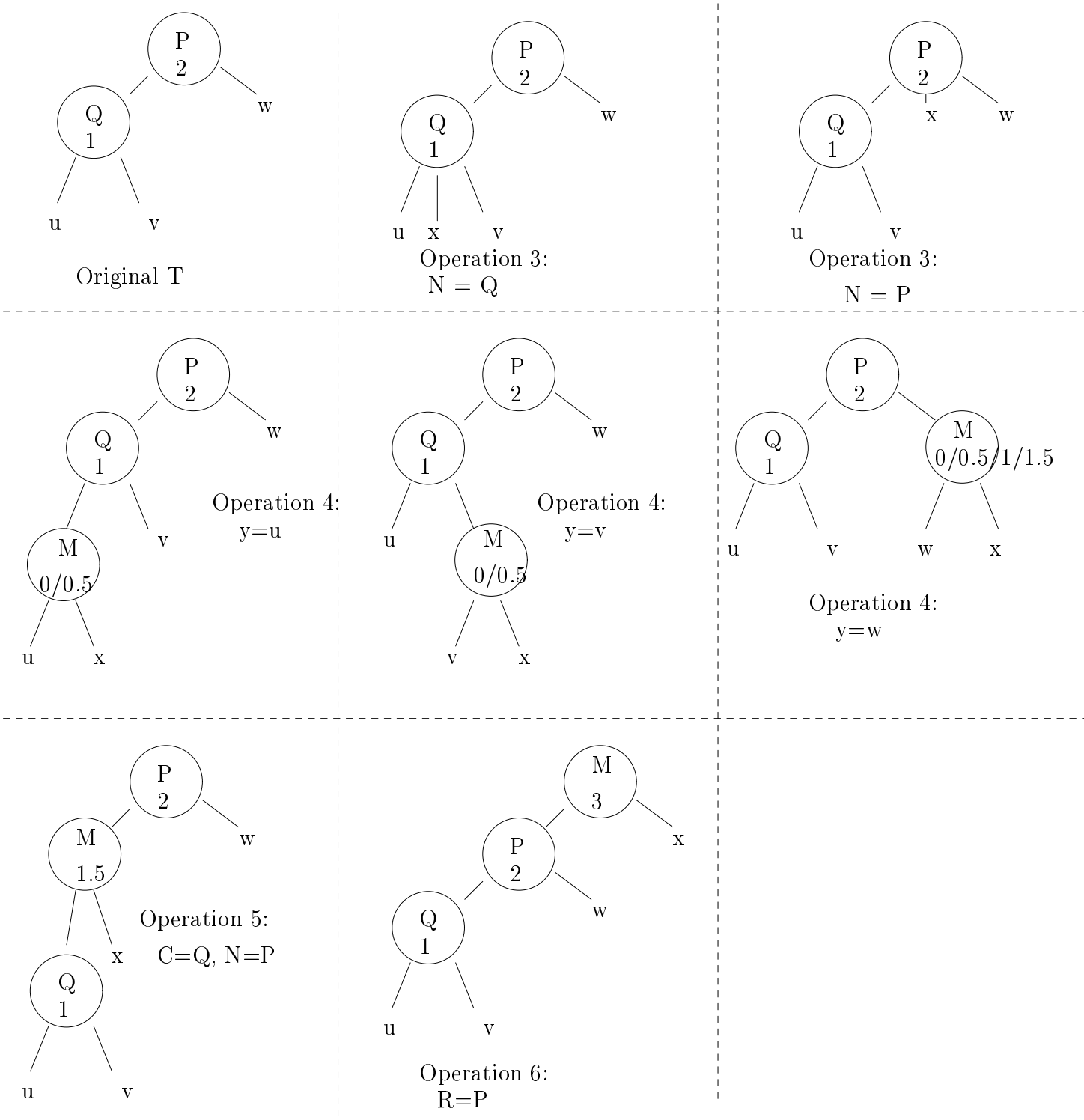

Figure 3: Extensions of a cluster tree

D. There is more than one value $y_1 \ldots y_k$ for which $\mathrm{od}(\Gamma(y_i, a) = D$. It is easily shown that in this case there is an internal node Q such that $y_1 \ldots y_k$ is just the set of symbols in the subtree of Q. There are three cases to consider:

D.i D=odiam($\Gamma(Q$.symbols)). Then apply operation (3) with $N = Q$.

D.ii $D >$ odiam($\Gamma(Q$.symbols)), and Q is not the root. Then apply operation (4) with $C = Q$. Set M.label to be the suitable value for D. (It is easily shown that $D <$ odiam($\Gamma(N$.symbols)), where N is the father of Q.)

D.iii $D >$ odiam($\Gamma(Q$.symbols)), and Q is the root. Apply operation (5).

∎

Lemma 25: Let $A = \{a_1 \ldots a_k\}$ be a finite set of points whose diameter has order-of-magnitude D. Then there exists a point u such that, for $i = 1 \ldots k$, $\mathrm{od}(u, a_i) = D$.

Proof: Let $b_1 = a_1$. By lemma A.8 there exists an infinite collection of points $b_2, b_3 \ldots$ such that $\mathrm{od}(b_i, b_j) = D$. Now, for any value a_i there can be at most one value b_j such that $\mathrm{od}(a_i, b_j) \ll D$; if there were two such values b_{j1} and b_{j2}, then by the om-triangle inequality, $\mathrm{od}(b_{j1}, b_{j2}) \ll D$. Hence, all but k different values of b_j are at least D from any of the a_i. Let u be any of these values of b_j. Then since $\mathrm{od}(u, a_1) = D$ and $\mathrm{od}(a_1, a_i) \not\ll D$ for all i, it follows that $\mathrm{od}(u, a_i) \not\ll D$ for all a_i. Thus, since $\mathrm{od}(u, a_i) \not\ll D$ but not $\mathrm{od}(u, a_i) \ll D$, it follows that $\mathrm{od}(u, a_i) = D$.

Lemma 26: Let T be a cluster tree; let Γ be a valuation over om-space Ω satisfying T; and let T' be an extension of T by x. If Ω is dense and unbounded above, then there is a value a such that the valuation $\Gamma \cup \{x \to a\}$ satisfies T'.

Proof: For operations (1) and (2) the statement is trivial.

Otherwise, let L be an extending label of T. If $L = 0$, then set $D = 0$. If L is in T, then let D be the order of magnitude corresponding to L in T under Γ. If $L_1 < L < L_2$ where L_1 and L_2 are labels of consecutive values in T, then let D_1 and D_2 be the orders of magnitude corresponding to L_1, L_2 in T under Γ. Let D be chosen so that $D_1 \ll D \ll D_2$. If L is greater than any label in the tree, then choose D to be greater than the diameter of the tree under Γ.

If T' is formed from T by operation (3), then using lemma 25 let a be a point such that $\mathrm{od}(a, \Gamma(y)) = \mathrm{odiam}(N)$ for all y in N.symbols.

If T' is formed from T by operation (4), then let a be a point such that $\mathrm{od}(a, \Gamma(y)) = D$.

If T' is formed from T by operation (5), then let a be a point such that $\mathrm{od}(a, \Gamma(y)) = D$ for all y in C.symbols. (Note that, since M.label $< N$.label, $D <$ odiam(N.symbols).)

If T' is formed from T by operation (6), then let a be a point such that $\mathrm{od}(a, \Gamma(y)) = D$ for all y in R.symbols.

In each of these cases, it is straightforward to verify that $\Gamma \cup \{x \to a\}$ satisfies T'. ∎

It should be observed that the conditions on Ω in lemma 26 are necessary, and that the statement is false otherwise. For example, let Ω be the om-space described in example I, section 2, of polynomials in an infinitesimal δ. Then Ω is not unbounded above; there is a maximum order-of-magnitude $O(1)$. Let T be the starting tree of figure 3 (upper-left corner), and let T' be the result of applying operation 6 (middle bottom). Let Γ be the valuation $\{u \to \delta, v \to 2\delta, w \to 1\}$. Then Γ satisfies T, but it cannot be extended to a valuation that satisfies T', as that would require x to be given a value such that $\mathrm{od}(v, w) \ll \mathrm{od}(x, w)$, and no such value exists within Ω. The point of the lemma is that, if Ω is required to be both dense and unbounded above, then we cannot get "stuck" in this way.

Correspondingly, lemma 28 is also not satisfied in this space. Let ϕ be the formula "$\exists_X$ od$(V, W) \ll$ od(W, X)", free in U, V, W. Then the valuation $\{u \to \delta, v \to 2\delta, w \to 1\}$ satisfies T but not ϕ, whereas the valuation $\{u \to \delta^2, v \to 2\delta^2, w \to \delta\}$ satisfies both T and ϕ.

Lemma 27: Let T be a cluster tree. Let X be a variable not among the symbols of T. Let α be an open formula in $\mathcal{L}$, whose free variables are the symbols of T and the variable X. Let ϕ be the formula $\exists_X \alpha$. Let Ω be an om-space that is dense and unbounded above. Then there exists an instantiation Γ of T in Ω that satisfies ϕ if and only if there exists an extension T' of T and an instantiation Γ' of T' that extends Γ and satisfies α.

Proof: Suppose that there exists an instantiation Γ of T that satisfies $\exists_X \alpha$. Then, by definition, there is a point a in Ω such that Γ satisfies $\alpha(X/a)$. That is, the instantiation $\Gamma \cup \{x \to a\}$ satisfies α. Let $\Gamma' = \Gamma \cup \{x \to a\}$. By lemma 24, the cluster tree T' corresponding to Γ' is an extension of T.

Conversely, suppose that there exists an extension T' of T and an instantiation Γ' of T' satisfying α. Let Γ be the restriction of Γ' to the symbols of T. Then clearly Γ satisfies the formula $\exists_X \alpha$. ∎

Lemma 28: Let T be a cluster tree. Let ϕ be an open formula in $\mathcal{L}$, whose free variables are the symbols of T. Assume that the only logical symbols in ϕ are $\neg$ (not), $\wedge$ (and), $\exists$ (exists), $=$ (equals) and variables names, and that the only non-logical symbol is the predicate "much_closer". Let Ω be an om-space that is dense and unbounded above. If one instantiation Γ of T in Ω satisfies ϕ then every instantiation of T in Ω satisfies ϕ.

Proof by structural induction on the form of ϕ. Note that an equivalent statement of the inductive hypothesis is, "For any formula ψ, either ψ is true under every instantiation of T, or ψ is false under every instantiation of T."

Base case: If ϕ is an atomic formula "$X = Y$" or "much_closer(W, X, Y, Z)" then this follows immediately from corollary 4.

Let ϕ have the form $\neg\psi$. If ϕ is true under Γ, then ψ is false under Γ. By the inductive hypothesis, ψ is false under every instantiation of T. Hence ϕ is true under every instantiation of T.

Let ϕ have the form $\psi \wedge \theta$. If ϕ is true under Γ then both ψ and θ are true under Γ. By the inductive hypothesis, both ψ and θ are true under every instantiation of T. Hence ϕ is true under every instantiation of T.

Let ϕ have the form $\exists_X \alpha$. If ϕ is true under Γ then by lemma 27, there exists an extension T' of T and a instantiation Γ' of T' such that α is true under Γ'. By the inductive hypothesis, α is true under every instantiation of T'. Now, if Δ' is an instantiation of T' that satisfies α, and Δ is the restriction of Δ' to the variables in T, then clearly Δ satisfies $\exists_X \alpha$. But by lemma 27, every instantiation Δ of T can be extended to an instantiation Δ' of T'. Therefore, every instantiation of T satisfies ϕ. ∎

Definition 10: Let T be a cluster tree, and let ϕ be a formula open in the variables of T. T *satisfies* ϕ if every instantiation of T satisfies ϕ. By lemma 28, if the om-space is dense and unbounded, this condition is equivalent to the condition that some instantiation of T satisfies ϕ.

Theorem 4: Let T be a cluster tree. Let ϕ be an open formula in $\mathcal{L}$, whose free variables are the symbols of T. Assume that the only logical symbols in ϕ are $\neg$ (not), $\wedge$ (and), $\exists$ (exists), $=$ (equals) and variables names, and that the only non-logical symbol is the predicate "much_closer". Let Ω be an om-space that is dense and unbounded above. Algorithm decide(T, ϕ) returns **true** if T satisfies ϕ and **false** otherwise.

function decide(T : cluster tree; ϕ : formula) **return boolean**
case
ϕ has form $X = Y$: **return** (distance$(X, Y, T) = 0$);
ϕ has form "much_closer(W, X, Y, Z)": **return** distance$(W, X, T) <$ distance(Y, Z, T));
ϕ has form $\neg\psi$: **return not**(decide(T, ψ))
ϕ has form $\psi \wedge \theta$: **return**(decide(T, ψ) **and** decide(T, θ))
ϕ has form $\exists_X \alpha$;
if for some extension T' of T by X, decide(T', α) = **true**
then return true
else return false endif end decide

function distance(X, Y : symbol; T : cluster tree) **return integer**
$N :=$ the common ancestor of X and Y in T;
return(N.label)
end distance

Proof: Immediate from the proof of lemma 28. ∎

Running time: As we have remarked above, for a tree T of size k there are at most $4k^2$ extensions of T to be considered. The total number of cluster trees considered is therefore bounded by $\Pi_{k=1}^{n} 4k^2 = 4^n(n!)^2$. It is easily verified that the logical operators other than quantifiers add at most a factor of s where s is the length of the sentence. Hence the running time is bounded by $O(4^n(n!)^2 s)$.

References

T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms,* MIT Press and McGraw Hill, 1990.

E. Davis (1990). "Order of Magnitude Reasoning in Qualitative Differential Equations". In D. Weld and J. de Kleer (eds.) *Readings in Qualitative Reasoning about Physical Systems*, Morgan Kaufmann. pp. 422-434.

M. Mavrovouniotis and G. Stephanopoulos (1990). "Formal Order-of-Magnitude Reasoning in Process Engineering." In D. Weld and J. de Kleer (eds.) *Readings in Qualitative Reasoning about Physical Systems*, Morgan Kaufmann. pp. 422-434.

O. Raiman (1990). "Order of Magnitude Reasoning." In D. Weld and J. de Kleer (eds.) *Readings in Qualitative Reasoning about Physical Systems*, Morgan Kaufmann. pp. 422-434.

A. Tarski (1951). *A Decision Method for Algebra and Geometry.* University of California Press, Berkeley, CA

D. Weld (1990). "Exaggeration." In D. Weld and J. de Kleer (eds.) *Readings in Qualitative Reasoning about Physical Systems*, Morgan Kaufmann. pp. 422-434.