

Supertree Construction with Constraint Programming

Ian P. Gent¹, Patrick Prosser², Barbara M. Smith³, and Wu Wei²

¹ School of Computer Science, University of St. Andrews, Scotland.
ipg@dcs.st-and.ac.uk

² Department of Computing Science, University of Glasgow, Scotland.
pat@dcs.gla.ac.uk

³ School of Computing and Mathematics, University of Huddersfield, England.
b.m.smith@hud.ac.uk

Abstract. A central goal of systematics is the construction of a *tree of life*, where the tree represents the relationship between all living things. The leaf nodes of the tree correspond to species and the internal nodes to hypothesized species, assumed to be extinct, where species have diverged. One problem that biologist face is to assemble a *supertree* from many smaller trees that have overlapping leaf sets. Polytime algorithms have been proposed for this problem [10, 14]. We present a simple constraint encoding of this problem. This is based on the observation that any rooted tree can be considered as being min-ultrametric when we label interior nodes with their depth in that tree. That is, any path from the root to a leaf corresponds to a strictly increasing sequence. We present a min-ultrametric constraint for rooted binary trees, resulting in what we believe to be the first constraint encoding of species trees. A simple modification of this encoding allows us to generate parsimonious trees when our data is over-constrained.

1 Introduction

In 1859 Charles Darwin published the most controversial book of its time, *The Origins of Species* [7]. Darwin proposed a theory that all living things have a common ancestor, and that new species arise by a splitting process, called specialisation. Darwin argued that the history of all species could be laid out as a rooted tree, with existing species as the leaves of that tree, and internal nodes representing points where two species diverged. By following a path from the root to a leaf, we would get the evolutionary history of a species.

One of the goals of biology¹, is the construction of *The Tree of Life*. This tree would be a tree of all living things. Clearly, this is an ambitious goal, and one most likely to be reached via a sequence of small steps. One such process is to build trees for a number of species, and then combine these trees into a supertree [10, 14]. In a biological study a few species, say S_1 , might then be represented

¹ Systematics in particular (sometimes referred to as phylogenetics); the study of the pattern of relationships among taxa.

as a rooted tree T_1 . A further study might produce a second tree T_2 from a different set of species S_2 . The two trees might then be combined to give a new tree T_3 . The construction of T_3 is trivial if S_1 and S_2 are disjoint, but if there is a non-empty intersection then construction will be more difficult.

There are two problems of interest. First, the decision problem: given two species trees T_1 and T_2 can they be combined into one tree T_3 ? When this question is answered in the negative we get our second problem: produce the best, i.e. most parsimonious tree. Algorithms already exist for both of these problems. For the decision problem we have the polytime algorithm *OneTree* [5, 10]. For the optimisation problem Semple and Steel [14], and more recently Page [12], propose greedy polytime algorithms. In this paper we present a constraint programming solution to the decision problem and the optimisation problem. That is, we encode the decision and optimisation problem as constraint satisfaction problems (CSP) [16]. Our encodings take a radically different approach to solving these problems, and represents a new perspective on these problems.

In the next sections we will present the problem of constructing a supertree, and review the existing algorithms [5, 10]. We then present a somewhat intuitive encoding of the problem as a constraint program, and show that this encoding is naive resulting in unacceptable complexity. We then show how we can view all rooted trees as being symbolically ultrametric [4, 3], and with this knowledge propose a good encoding for the decision problem. We show how this can be extended to handle the optimisation problem in a fair and non-greedy way.

2 Species Trees and Supertrees

In a fully resolved species tree each internal node has degree 3, with the exception of the root. An internal node has one parent and two children, and the root has two children and no parent. Consequently a fully resolved species tree with n leaf nodes has $n - 1$ internal nodes. In Figure 1 we have three species, namely a , b , and c . Species a and b are more closely related to one another than they are to c . More specifically, we say that *the most recent common ancestor of a and b is greater than the most recent common ancestor of a and c (equally b and c)*, where the most recent common ancestor of two leaf nodes a and b is the internal node furthest from the root that has a and b as descendants. We compare most recent common ancestors by measuring their distance from the root. That is,

$$mrca(a, b) > mrca(a, c) \tag{1}$$

$$mrca(a, b) > mrca(b, c) \tag{2}$$

$$mrca(a, c) = mrca(b, c) \tag{3}$$

Note, that in Figure 1 we have labeled the two interior nodes. Generally, interior nodes are anonymous and we introduce this labeling only to explain the equations above. We see that the most recent common ancestor of a and b is interior node Y , i.e. $mrca(a, b) = Y$. Furthermore, $mrca(a, b) = mrca(b, a) = Y$ (i.e. the relation is symmetric) and from equation (3) $mrca(a, c) = mrca(b, c) = X$. From

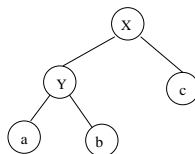


Fig. 1. A species tree, where species a and b are more closely related to each other than they are to species c . This small tree can also be represented as the rooted triple $((a, b), c)$.

equation (1) we have $mrca(a, b) > mrca(a, c)$ (i.e. $Y > X$) and $mrca(a, b) > mrca(b, c)$ (i.e. yet again $Y > X$ from equation (2)).

Species trees are frequently presented as a collection of rooted triples, of the form $((a, b), c)$ meaning that $mrca(a, b) > mrca(a, c)$, $mrca(a, b) > mrca(b, c)$, and $mrca(a, c) = mrca(b, c)$. The *BreakUp* algorithm of [10] takes as input a species tree and delivers as a result a set of rooted triples that define that tree. Therefore we might have the two triples as shown in Figure 2, $((a, b), c)$ and $((b, c), d)$. These can then be resolved into the third tree shown, with 4 leaf nodes and 3 internal nodes. An algorithm that does this is the *OneTree* algorithm of [10, 5].

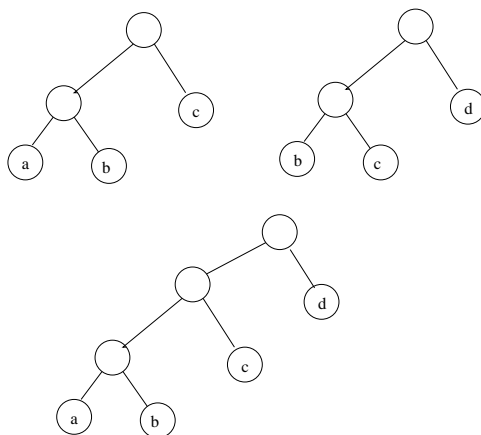


Fig. 2. Two rooted triples $((a, b), c)$ and $((b, c), d)$ and the resultant supertree

2.1 Algorithm *BreakUp*

The *BreakUp* algorithm [10] takes as input a tree T and delivers a set of rooted triples and fans that define that tree². For ease of exposition, we describe the algorithm for the case when the tree T is bifurcating, i.e. ignoring fans. The algorithm finds the deepest interior node in the tree v . Interior node v has two leaf nodes, a and b . *BreakUp* finds the parent of v , call it node w . From w we find the sibling of v , call it u . From u find any leaf node c . *BreakUp* then writes out the triple $((a, b), c)$, deletes leaf nodes a and b , and renames interior node v to become the new leaf node labeled with b . The algorithm is then applied to the reduced tree and terminates when T is a triple or less. Figure 3 shows two trees, both from Thorley's PhD thesis [15]. The trees are presented in bracket notation, as a set of rooted triples as a result of *BreakUp*, and as a cladogram drawn using the TreeView software [11].

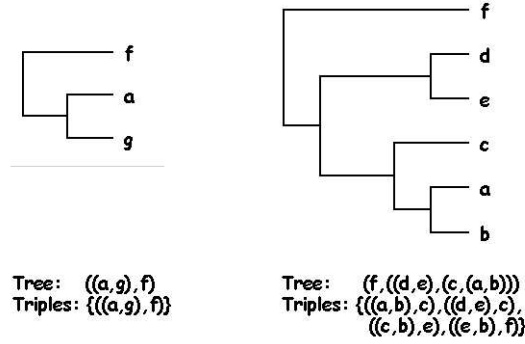


Fig. 3. Cladograms of the trees $(f, (a, g))$ and $(f, ((d, e), (c, (a, b))))$, originally figure 36 in Thorley's PhD thesis [15]. Algorithm *BreakUp* produces the triple $((a, g), f)$ for the left hand tree and the four triples $((a, b), c)$, $((d, e), c)$, $((c, b), e)$, $((e, b), f)$ for the tree on the right. The cladograms were produced using Rod Page's TreeView software [11].

2.2 Algorithm *OneTree*

The *OneTree* algorithm presented in [10] and in [5], is a specialisation of the *Build* algorithm of [1]. *OneTree* takes as input a set of rooted triples R and a set of species S and delivers a supertree that contains the species in S respecting the triples in R . The rooted triples will have been produced by processing a number of trees with the *BreakUp* algorithm, and the set S is the union of the leaf sets of those trees.

² A fan is a set of unresolved species. For example, in the fan $\{a, b, c\}$ the three species are allowed to share a parent. In this paper we will assume that all species have been resolved, and fans do not exist. However, we will return to this later in the paper.

The *OneTree* algorithm answers the decision problem by producing a tree if one exists, or the empty tree if none exists. The algorithm starts construction with the root of the tree, using the rooted triples to divide the leafs into disjoint subsets, where the leafs in a subset lie in the same subtree attached to the root. If there is only one triple in R then the tree is defined by that triple. Otherwise the algorithm constructs a graph G using R as follows. Construct in G the edges $\{(a, b) \mid ((a, b), c) \in R \wedge \{a, b, c\} \subseteq S\}$. If G is a single component there is no tree, and *OneTree* delivers the empty tree. Otherwise an internal node is created, v . For each component S_i in G we collect in R_i the set of rooted triples in R with leafs in S_i . *OneTree* is then called recursively on each pair (S_i, R_i) , and the resultant subtrees are then attached to v . Note, that this may result in a non-bifurcating tree, as interior nodes may have degree greater than three. In Figure 4 two trees (from Figure 3) are combined to produce a supertree. First, the two initial trees are broken up into rooted triples using the *BreakUp* algorithm. These triples are then passed to *OneTree* along with the set of species $\{a, b, c, d, e, f, g\}$, and a supertree $(f, ((d, e), ((c, (a, b)), g)))$ is produced. Note that there are 9 possible supertrees that respect those triples [15]. The *OneTree* algorithm is of complexity $O(m.n)$ where we have n leaf nodes and m triples. For a complete description see [10].

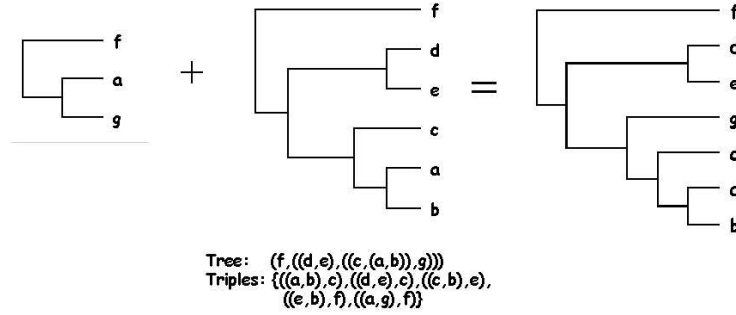


Fig. 4. *OneTree* applied to the two trees $(f, (a, g))$ and $(f, ((d, e), (c, (a, b))))$, giving the supertree $(f, ((d, e), ((c, (a, b)), g)))$. *OneTree* takes as input the set of rooted triples $R = \{((a, g), f), ((a, b), c), ((d, e), c), ((c, b), e), ((e, b), f)\}$ and the set of species $S = \{a, b, c, d, e, f, g\}$

3 An Intuitive Encoding of Trees

We now present our first constraint encoding of the decision problem, does a supertree exist? In one respect it is similar to the *OneTree* algorithm, i.e. if there is a tree it is delivered. Our encoding is presented in two parts. First, we have a constraint encoding that builds a binary tree with $n - 1$ interior nodes and n leaf nodes. We then have an addition to this such that the rooted triples are respected.

3.1 Generating a binary tree

To generate a binary tree with $n - 1$ interior nodes and n leaf nodes, we have the following constraint encoding

- We have a set of constrained integer variables $X = \{x_0, x_1, \dots, x_{2n-2}\}$, such that x_i is a node in the tree, either a leaf or internal. Each variable has a domain $\{0, \dots, n - 2\}$, with the exception of x_0 , which has a domain with one value, 0.
- We divide X into two subsets, X_{le} the set of leaf nodes and X_{in} the set of internal nodes. Nodes 0 to $n - 2$ correspond to the set of internal nodes X_{in} and nodes $n - 1$ to $2n - 2$ correspond to the set of leaf nodes X_{le} . The root of the tree is x_0 .
- If a node (leaf or internal) x_i has value j then node j is the parent of node i . (The root node is its own parent.)
- For each internal node i , we have a set variable C_i representing the children nodes. The possible values of such a variable are taken from $\{1, \dots, 2n - 2\}$, i.e. all nodes can be child nodes, apart from the root node. The cardinality of C_i is constrained to be exactly 2.
- For each node (whether a leaf node or an internal node) we have a set variable A_i representing its ancestors. The root node has only one ancestor (itself), so $A_0 = \{0\}$. We also have constraints to ensure that if node j is the parent of node i , then $A_i = \{j\} \cup A_j$.
- To avoid cycles, we use the the following constraints: if $i \in A_j$ then $j \notin A_i$, for all pairs i, j .

The solutions of the constraint satisfaction problem (CSP) just described are arbitrary rooted binary trees. However, there are very many of them: given any labeled tree, we can permute all the labels of the internal nodes (except the root node) and all the labels of the leaf nodes.

3.2 Species Trees

We now want to find binary trees which obey stated constraints on the leaf nodes. For a triple $((a, b), c)$ we need to ensure that there is a node which is an ancestor of both a and b , but is not an ancestor of c .

This can be modeled in the above encoding by introducing a new integer variable $v_{((a,b),c)}$ for each triple. The value assigned to such a variable will be an internal node which is an ancestor of a and b and not of c . The domain of $v_{((a,b),c)}$ is $\{1, \dots, n - 2\}$, i.e. the value can be any internal node other than the root node, since the root node is an ancestor of every other node and therefore of c . The constraint on $v_{((a,b),c)}$ is: $v_{((a,b),c)} \in A_a \ \& \ v_{((a,b),c)} \in A_b \ \& \ v_{((a,b),c)} \notin A_c$.

Note that we do not do anything to enforce that $v_{((a,b),c)}$ is the *most recent* common ancestor of a and b ; it can be any ancestor of a and b as long as it is not also an ancestor of c . We can then find possible species trees correctly representing the information given by a two-stage search process: first assign values to the variables in X and then to the variables $v_{((a,b),c)}$.

3.3 Behaviour of the encoding

The encoding can be made more efficient, in particular by removing symmetries. However, performance is unacceptable. Take for example a simple problem with the following set of rooted triples $R = \{((a, b), c), ((b, c), d), ((c, d), a)\}$. The encoding will build a tree made up of the 3 interior nodes and then place the 4 leaf nodes across this. In the worst case this will take $4!$ steps to conclude that no configuration of the leaf nodes is compatible with the arrangement of the interior nodes. The search process will then try another arrangement of the interior nodes, eventually exhausting this process to prove that there is no tree that satisfies R . The *OneTree* algorithm would construct a single component graph with 4 vertices and report failure.

4 Ultrametric Trees and Ultrametric Matrices

Definition 1. *An ultrametric tree T is a rooted tree with n uniquely labeled leaves and interior nodes labeled with values (real or integer) such that any path from the root to a leaf node constitutes a strictly decreasing sequence. In a **min-ultrametric tree** interior nodes are labeled with values such that any path from root to a leaf is a strictly increasing sequence [8].*

An ultrametric tree can be represented by an ultrametric matrix D , and an ultrametric matrix can be represented by an ultrametric tree.

Definition 2. *Let D be an $n \times n$ symmetric matrix. D is an **ultrametric matrix** if there are at most $n - 1$ distinct values within D , and for any three indices i , j , and k there is a tie for the maximum of $D_{i,j}$, $D_{i,k}$, and $D_{j,k}$.*

Given an ultrametric matrix D , $D_{i,j}$ is an integer value corresponding to the time of divergence of species i and j . The diagonal values are all zero, i.e. an existing species does not diverge from itself. We are then to build a bifurcating species tree T with n leaves and $n - 1$ interior nodes such that the interior nodes of T are labeled with values from the matrix D , and on a path from the root to a leaf node the labels on the interior nodes of that path are strictly decreasing.

The element $D_{i,j}$ is the most recent common ancestor of species i and j , labeled with the time that i and j diverged³. In Figure 5 we have an ultrametric matrix D and the corresponding ultrametric tree T (these figures are taken from [8] page 450). Looking at the tree and matrix we see that the most recent common ancestor of A and D is labeled with the value 5, and the most recent common ancestor of E and B is labeled with value 8. The leaf nodes might be considered as species, and the values on the interior nodes the number (let's say) of millions of years that have passed since species diverged (i.e. species D diverged from species E 5 million years ago).

In [7], it is proved firstly that an ultrametric tree has an ultrametric matrix. Secondly, it is shown, using a constructive proof, that an ultrametric matrix

³ Gusfield [8] calls this the lowest common ancestor, lca, although many papers refer to this as mrca.

has an ultrametric tree. The proof gives an algorithm of complexity $O(n^2)$ for constructing an ultrametric tree.

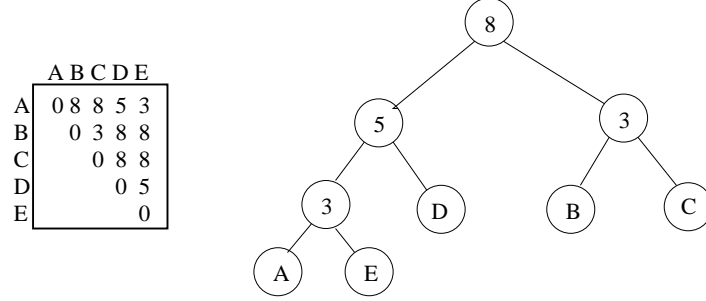


Fig. 5. The ultrametric matrix D and its ultrametric tree T

Any rooted tree can be considered as being ultrametric. We can do this by labeling interior nodes with their height, where the height of a leaf node is zero, and the height of an interior node is one plus the maximum of the heights of its children. For a min-ultrametric tree we label interior nodes with their depth in the tree, where the root has a depth of zero, and the depth of any other interior node is one plus the depth of its parent node. A rooted triple $((a, b), c)$ has an obvious interpretation in a min-ultrametric tree, where interior nodes are labeled with their depth in the tree. As previously noted, $((a, b), c)$ is equivalent to the relations (1), (2), and (3). Therefore the most recent common ancestor of a and b is at a greater depth in the tree than the most recent common ancestor of a and c .

5 The Min-Ultrametric Constraint

We now present a constraint encoding which provides a unique representation of trees up to symmetry of either renaming of internal nodes, or swapping of left and right. It does this by not encoding those aspects of trees and therefore not needing to break any symmetry. It also makes it trivial to add rooted triples.

The simple idea which achieves this is to encode the *depth* of the most recent common ancestors in the tree. We have a $n \times n$ two dimensional array D of constrained integer variables. Each variable $D_{i,j}$ takes a value in the range 0 to $n-1$. Since the array is symmetric, $D_{i,j} = D_{j,i}$, for all i and j , and we arbitrarily set the diagonal $D_{i,i} = 0$. The value assigned to $D_{i,j}$ represents the depth of the most recent common ancestor of leaf nodes i and j . For example, in the tree $(a, (b, (c, d)))$ of Figure 6, we have $D_{a,b} = D_{a,c} = D_{a,d} = 0$, $D_{b,c} = D_{b,d} = 1$ and $D_{c,d} = 2$. Notice that there is no indication of whether a is to the left or right of b .

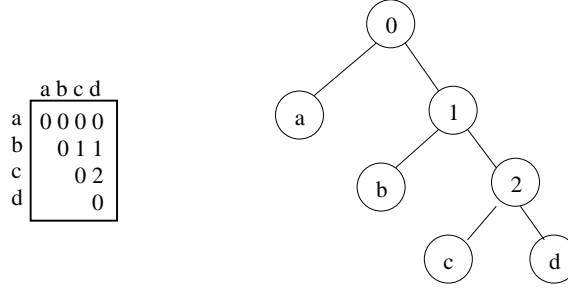


Fig. 6. The tree $(a, (b, (c, d)))$ made min-ultrametric by labeling interior nodes with their depth

To encode the constraints on D , we start with the constraint for the triple $((a, b), c)$. This means that a is closer to b than it is to c , or equivalently that:

$$\text{triple}(a, b, c) \equiv [(D_{a,c} = D_{b,c}) \wedge (D_{a,b} > D_{b,c}) \wedge (D_{a,b} > D_{a,c})] \quad (4)$$

This encodes the equations (1), (2), and (3). We can then guarantee that D is a min-ultrametric matrix by demanding that:

$$\forall a \forall b \forall c [(a \neq b \neq c) \wedge (\text{triple}(a, b, c) \vee \text{triple}(b, c, a) \vee \text{triple}(c, a, b))] \quad (5)$$

i.e. one of these triples holds for every combination of three species a , b , and c . This successfully encodes that the minimum value of the three variables $D_{a,b}$, $D_{b,c}$, $D_{c,a}$ is shared by two of them and not the third, the defining property of a min-ultrametric matrix where the resultant tree is bifurcating.

This can also be viewed from a geometric perspective. We can consider the indices a , b and c as being vertices of a triangle, and the matrix elements $D_{a,b}$, $D_{a,c}$ and $D_{b,c}$ as being the length of the edges of that triangle. Triangles are then forced to be isosceles, and equilateral triangles are disallowed.

We require more than just that D is a min-ultrametric matrix; we need to insist that the values in a row of D do not contain any gaps. That is, in the resultant tree any path from the root to a leaf node will be an increasing sequence, and that sequence will have no numeric gaps. For example, a path 0,1,1,3,4 would be illegal as we have a *hole* at depth 2, however 0,1,1,2,3 would be a legal sequence. If there is some b such that $D_{a,b} = 2$, then there have to be values c and d such that $D_{a,c} = 1$ and $D_{a,d} = 0$. This is because the depths reading down from the root to a have to be simply 0, 1, 2, ... We can insist on every integer up to a 's depth occurring because we know that any of a 's ancestor nodes is the most recent common ancestor of a and at least one other leaf node in a bifurcating tree. We do this by introducing arrays of constrained integer variables to count the number of occurrences of each integer in each row of D . We then demand that if the count for i is zero, then the count for $i + 1$ is also zero. Equivalently, if i occurs in a row, and i is greater than 0, then $i - 1$ occurs in that row also, i.e.

$$\forall a \forall b [(D_{a,b} = i \wedge i > 0) \rightarrow \exists c (D_{a,c} = i - 1)] \quad (6)$$

With this encoding, any consistent instantiation of the variables in D is min-ultrametric and has a min-ultrametric tree. The number of possible such instantiations is $\frac{(2n-2)!}{2^{n-1}(n-1)!}$ [13].

Given a set of rooted triples R we can then post each triple as a constraint on the array D . For a triple $((i, j), k) \in R$ we post the constraint $(D_{i,k} = D_{j,k}) \wedge (D_{i,j} > D_{j,k}) \wedge (D_{i,j} > D_{i,k})$, i.e. $triple(i, j, k)$ from equation (4) above. This breaks the three disjunctive constraints already posted between these three variables in (5) above. Consequently, a consistent instantiation of the variables in D will correspond to a species tree that respects the triples in R . Given that consistent instantiation of D we can then process D to construct the tree, using the algorithm in [8]. Therefore, our constraint encoding achieves the same net result as the *OneTree* algorithm, and trivially extends to the enumeration of all species trees (i.e. performing the same function as *AllTrees* [10]). To generate all trees we allow our solving procedure to backtrack whenever it finds a solution, and to continue on to the next solution.

However, there is one notable difference between the trees produced by our encoding and the trees produced by *OneTree*. Our encoding only produces bifurcating trees, whereas *OneTree* may produce trees with interior nodes with more than two children. This happens when the triples do not fully resolve the tree, whereas the constraint encoding automatically forces a resolution. We can relax our constraints to allow this. The disjunctive constraints posted across the matrix D can be relaxed as follows:

$$relTriple(a, b, c) \equiv [(D_{a,c} = D_{b,c}) \wedge (D_{a,b} \geq D_{b,c}) \wedge (D_{a,b} \geq D_{a,c})] \quad (7)$$

In equation (5) replace $triple(a, b, c)$ with our relaxed constraint $relTriple(a, b, c)$. From a geometric perspective, for any three vertices a, b, c , by default we allow triangles to be isosceles or equilateral. Our constraint encoding requires of the order $O(n^2)$ variables, each with a domain of size n , and $O(C_3^n)$ ternary constraints. Figure 7 shows the species tree with its ultrametric matrix, resulting from the constraint encoding of the trees in Figure 4.

6 An Empirical Example

We now present an example using biological data. Two species trees, both of sea birds taken from [9], are presented in Figure 8. The tree on the left has 20 species, and on the right 16 species. The trees have two species in common, *Diomedea epomophora* (the Royal Albatross) and *Thalassarche bulleri* (Buller's Albatross). Together, the trees have 34 species. The two species in common to both trees have been typed in upper case.

The *BreakUp* algorithm was applied to both trees producing 32 rooted triples. The constraint encoding, implemented in Choco and given in the appendix, took about 5 seconds to build C_3^{34} constraints from equation (5), 32 constraints from equation (4) and 34 constraints from equation (6). The first solution was then found in 1 second, and 2 backtracks were performed. No variable or value ordering heuristic was used. That is, the default Choco search was

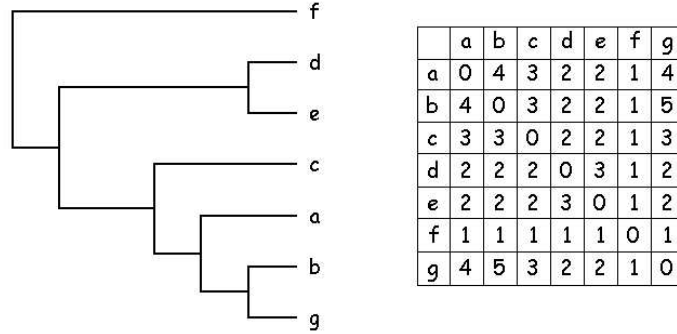


Fig. 7. The min-ultrametric matrix and tree produced from the rooted triples $\{((a, g), f), ((a, b), c), ((d, e), c), ((c, b), e), ((e, b), f)\}$. Note that there are again 9 possible solutions [15]. This tree is comparable to that given in Figure 4

used: first, all disjunctions are resolved; domain splitting then takes place; finally, variables are instantiated. Figure 9 shows the supertree produced using *OneTree* on the left, and on the right the min-ultrametric tree resulting from the constraint encoding. Again, shared species are typed in upper case.

7 Complexity, Extensions, and Optimisation

In general, arc-consistency can be established in $O(ed^r)$ time [2], where there are e constraints, each of arity r , and domain size is d . In the encoding presented $e = O(n^3)$, $d = n$ and $r \leq 3$. Thus arc-consistency can be established in $O(n^5)$. Clearly, this is less efficient than the complexity of *OneTree*. However, we hope that we can improve upon this by designing a global ultrametric constraint.

We have programmed our encodings using ILOG Solver and the Choco toolkit. The Choco program is listed in the appendix. In the Solver implementation, the min-ultrametric constraint is encoded as a *table constraint*. Solver enforces generalized arc consistency on table constraints (which in this case are ternary) using the general arc consistency schema introduced by Bessière and Régin [2]. In an empirical comparison of the two implementations, Solver never backtracked in solving the decision problem. On over-constrained problems Solver reported a failure as soon as a triple constraint was reached which was inconsistent with those already posted: this allowed us to find solutions satisfying the maximum number of constraints with little effort. On the other hand, on some over-constrained problems the Choco implementation had to search in order to show inconsistency ; we believe that the difference is due solely to the higher level of consistency enforced on the min-ultrametric constraints in the Solver implementation. We conjecture that it will always be possible to find a solution or show that there is no solution, without backtracking in either case,

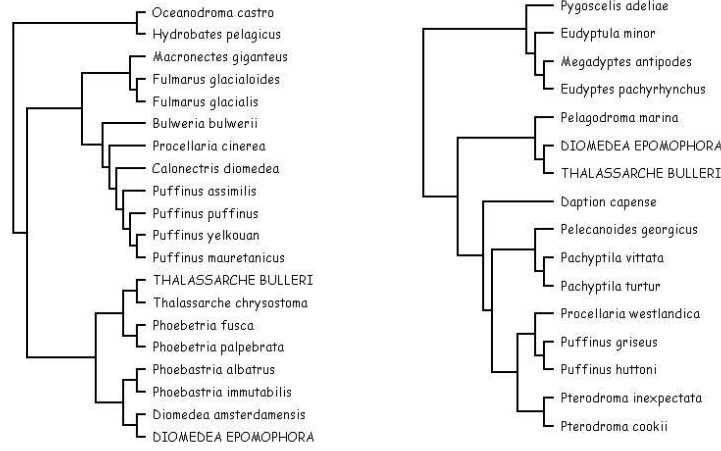


Fig. 8. The two trees are of sea birds. On the left there are 20 species and on the right 16. The trees share two species, namely *Diomedea epomophora* and *Thalassarche bulleri*. Therefore there are 34 unique species in all. The BreakUp algorithm produces 32 rooted triples that define these trees.

with the Solver implementation. In other words, with a suitable instantiation order, enforcing generalized arc consistency on the min-ultrametric constraint will allow inconsistency amongst the triple constraints to be detected immediately, or a consistent solution satisfying all the triple constraints to be found without search.

The encoding has been presented for fully resolved species trees. That is, we have so far assumed that we only have triples, and no fans. A fan is a set of species that have the same parent, i.e. have not been fully resolved. We can easily extend our encoding to cover this case. Rather than forcing all triples i, j, k to form isosceles triangles we can also allow equilateral triangles. We do this by replacing the *triple* constraint (equation (4)) with its relaxed version *relTriple* (equation (7)) and the encoding can then produce non-bifurcating trees. A *fan* constraint $\{a, b, c\}$ is then posted as $D_{a,b} = D_{b,c} = D_{a,c}$. With this relaxation, the constraint encoding will enumerate the same trees as the *AllTrees* algorithm.

We can also extend our encoding such that it builds the most parsimonious tree, i.e. when the problem is over-constrained by triples, find the largest resolved tree that satisfies most triples, or involves most species. For each rooted triple $((i, j), k)$ we introduce a constrained 0/1 variable $P_{i,j,k}$ (P for penalty). We now post rooted triple constraints conditionally, such that the constraint is either satisfied with $P_{i,j,k} = 0$ or the constraint is ignored with $P_{i,j,k} = 1$. The search process is then given the goal of minimising the sum of all the P variables, i.e. minimise the penalties associated with ignoring triples.

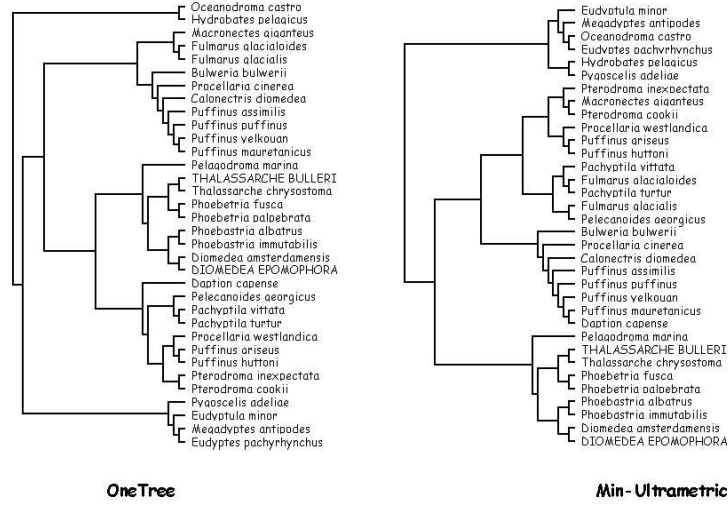


Fig. 9. On the left, the supertree produced using *OneTree* with 34 species of birds and 32 rooted triples, and on the right a tree from the min-ultrametric encoding

8 Conclusion and Future Work

We have presented a new method for building supertrees. This is based on the observations that any tree can be considered min-ultrametric, when we label interior nodes with their depth in that tree, and any min-ultrametric matrix has a corresponding min-ultrametric tree. This then leads us to an encoding of the decision problem as a constraint satisfaction problem. This can then be extended to deal with unresolved triples as well as over-constrained problems. However, our encoding is not as efficient as the conventional algorithms. So, why should we even consider the constraint programming approach?

Constraint programming has a number of strengths, in particular its ability to model rich problems. It is typically easy to add side constraints to a *pure* problem, such that the solution is more realistic. It may be that when we build supertrees there is additional information that we would like to incorporate into our solution, other than just triples and fans⁴. This might be difficult to do in the *OneTree* framework, but relatively easy with constraint programming. When we address the optimisation problem, the constraint encoding allows us to find an optimal tree, subject to whatever criteria we care to encode into our objective function. Although in the worst case this can take exponential time, constraint programming gives us another property that we can exploit: the incorporation of domain knowledge into the search process. We might encode variable and value ordering heuristics such that exponential behaviour is a rare occurrence, and average case behaviour is acceptable.

⁴ We might think of this as a generalisation of the constraint trees of [6]

We also expect that our encodings can be improved. In our Choco encoding, for every three indices we post three disjunctive constraints, whereas in our Solver implementation we post a single constraint. We hope to get improved efficiency by writing a special-purpose propagation algorithm for this constraint, rather than relying on the general schema given by Bessière and Régin [2].

Acknowledgments

We would like to thank David Gilbert, Françoise Laburthe, Rod Page, and Evgeny Selensky.

References

1. A.V. Aho, Y. Sagiv, T.G. Szymanski, and J.D. Ullman. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM J. Comput.*, 10(3):405–421, August 1981.
2. C. Bessière and J-C Régin. Arc consistency for general constraint networks: Preliminary results. In *Proceedings of IJCAI'97*, pages 398–404, 1997.
3. Sebastian Böcker and Andreas W.M. Dress. Recovering Symbolically Dates, Rooted Trees from Symbolic Ultrametrics. *Advances in Mathematics*, 138(1):105–125, 1998.
4. Sebastian Böcker and Andreas W.M. Dress. *Symbolic Ultrametrics*. 33rd Winter Seminar, Molecular Biology and Biophysical Chemistry of the Cell, Klosters, Switzerland, 1998.
5. D. Bryant and M. Steel. Extension Operations on Sets of Leaf-labeled Trees. *Advances in Applied Mathematics*, 16:425–453, 1995.
6. M. Constatinescu and D. Sankoff. Tree enumeration modulo a consensus. *Journal of Classification*, 3:349–356, 1986.
7. C.R. Darwin. *The Origin of Species*. John Murray, London, 1859.
8. Dan Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997.
9. M. Kennedy and R.D.M. Page. Seabird supertrees: Combining partial estimates of procellariiform phylogeny. *The Auk, A Quarterly Journal of Ornithology*, 119:88–108, 2002.
10. Meei Pyng Ng and Nicholas C. Wormald. Reconstruction of rooted trees from subtrees. *Discrete Applied Mathematics*, 69:19–31, 1996.
11. R.D.M. Page. TREEVIEW: An application to display phylogenetic trees on personal computers. *Computer Applications in the Biosciences*, 12:357–358, 1996.
12. R.D.M. Page. Modified mincut supertrees. *WABI 2001, Workshop on Algorithms in BioInformatics*, 2002.
13. E. Schröder. Vier combinatorische probleme. *Zeit. für Math. Phys.*, 15:361–376, 1870.
14. Charles Semple and Mike Steel. A supertree method for rooted trees. *Discrete Applied Mathematics*, 105:147–158, 2000.
15. Joseph L. Thorley. Cladistic information, leaf stability and supertree construction. *PhD Thesis, University of Bristol, Department of Biological Sciences*, 2000.
16. E.P.K. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.

Appendix: an implementation in Choco

We present below an encoding of the problem in the Choco constraint programming toolkit⁵. The function *utree*(*n*) delivers a problem, composed of $\frac{n(n-1)}{2}$ bound integer variables and the min-ultrametric constraints. A call to *solve*(*p*, *false*), where *p* : *Problem* := *utree*(*n*), will deliver the first min-ultrametric instantiation of the variables, and a call to *solve*(*p*, *true*) will enumerate all $\frac{(2n-2)!}{2^{n-1}(n-1)!}$ instantiations. To post the triple ((*i*, *j*), *k*) on the problem *p* we call *postTriple*(*p*, *i*, *j*, *k*).

```
[utree(n:integer) : Problem
-> let pb := makeProblem("utree", (n * (n - 1) / 2) + 1),
    X := makeBoundIntVar(pb, "X", 0, 0),
    M := list<array<IntVar>>{make_array(n, IntVar, X) | i in (1 .. n)}
in (for i in (1 .. n - 1)
    for j in (i + 1 .. n)
        (M[i][j] := makeBoundIntVar(pb, "M", 1, n - 1),
         M[j][i] := M[i][j]),
    for i in (1 .. n - 2)
        for j in (i + 1 .. n - 1)
            for k in (j + 1 .. n)
                let A := M[i][k],
                    B := M[k][j],
                    C := M[i][j],
                    U1 := and(A == B, C > B),
                    U2 := and(B == C, A > C),
                    U3 := and(C == A, B > A),
                    Ux := or(U1, or(U2, U3))
                in post(pb, Ux),
    for i in (1 .. n)
        for j in (1 .. n)
            let C := ifThen(occur(j, list!(M[i]))) == 0,
                            occur(j + 1, list!(M[i]))) == 0)
            in post(pb, C),
    X.hook := M,
    pb)]

[postTriple(pb:Problem, i:integer, j:integer, k:integer) : void
-> let M := pb.vars[1].hook
in (post(pb, M[i][k] == M[j][k]),
    post(pb, M[i][j] > M[j][k]),
    post(pb, M[i][j] > M[i][k]))]
```

⁵ The official site of the Choco constraint programming system can be found here <http://www.choco-constraints.net/>