High-Performance Algorithm Engineering for Computational Phylogenetics

Bernard M.E. Moret

moret@cs.unm.edu

Department of Computer Science
University of New Mexico
Albuquerque, NM 87131

Overview

- Phylogenies
- Computational Phylogenetics
- An Example: Quartet-Based Methods
- An Example: Gene-Order Phylogenies
    - Breakpoint phylogeny
    - Inversion and other genomic distance measures
    - GRAPPA: a high-performance software tool for reconstructing phylogenies from gene-order data
    - What did we do to make it one billion times faster than its predecessor?

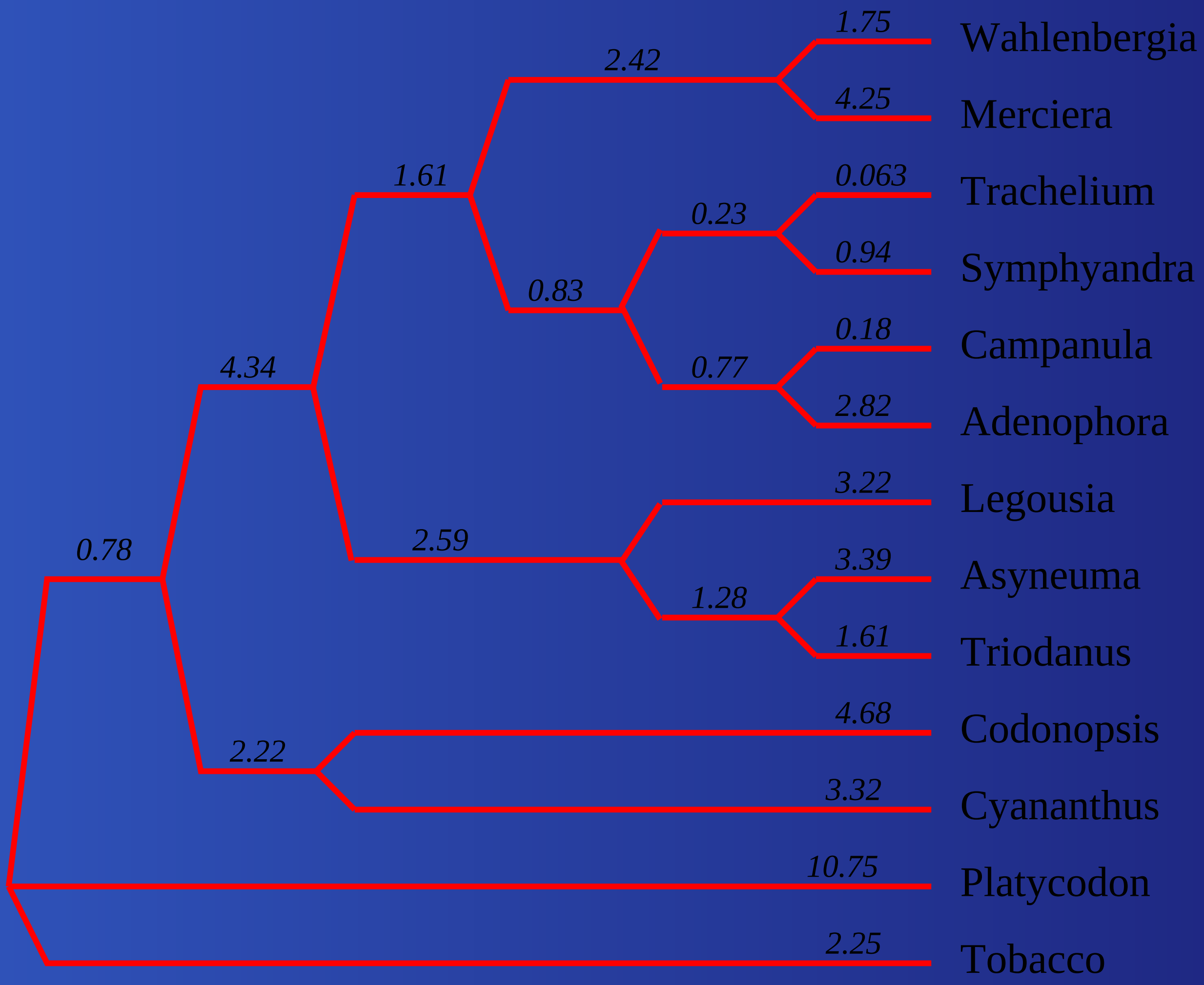
Phylogenies

A phylogeny is a reconstruction of the evolutionary history of a collection of organisms; it usually takes the form of a tree.

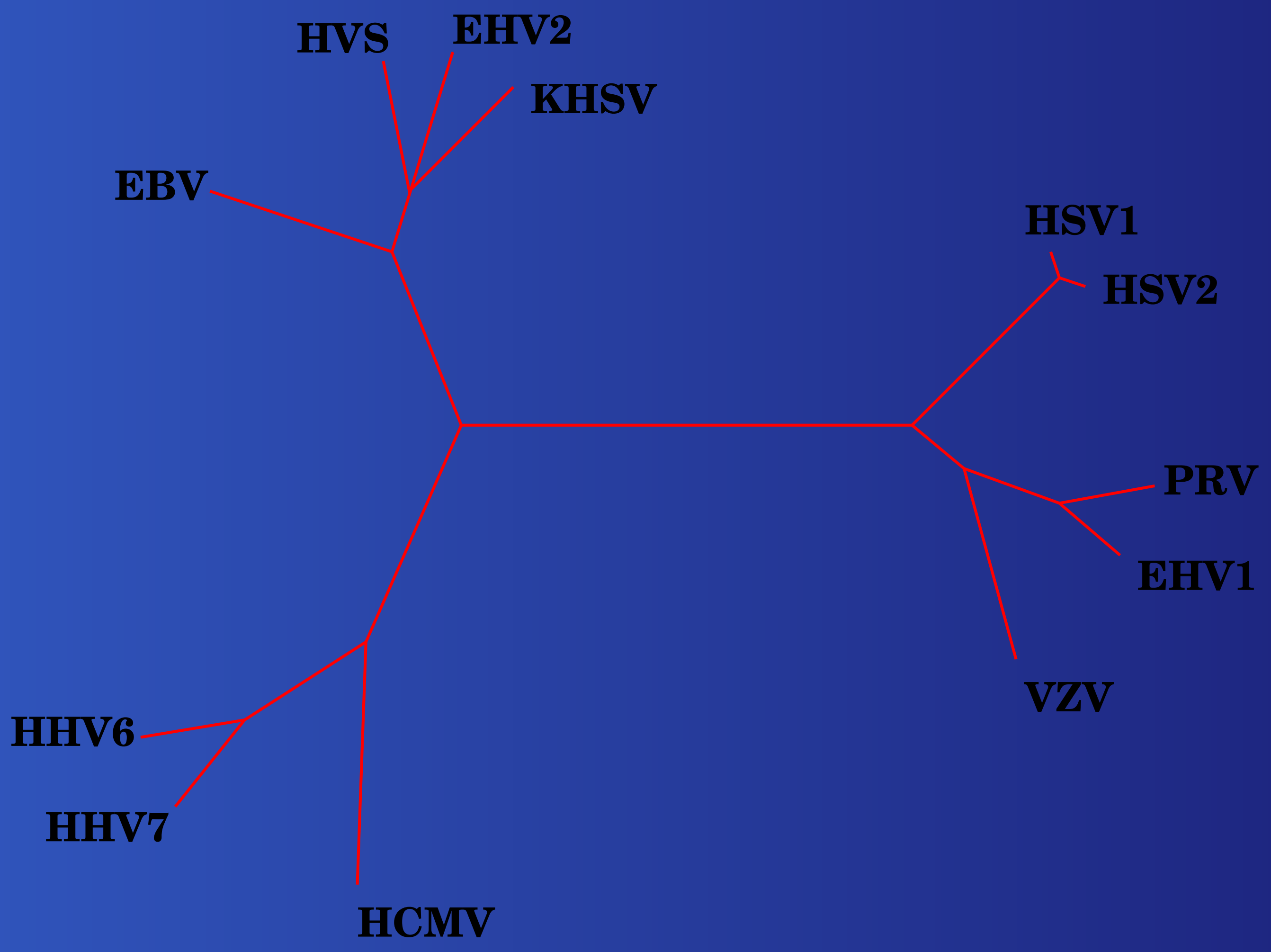
Modern organisms are placed at the leaves and ancestral organisms occupy internal nodes.

The edges of the tree denote evolutionary relationships.

12 Species of *Campanulaceae*



Herpes Viruses that Affect Humans



Reconstructing Phylogenies

Reconstructing phylogenies is a major component of modern research programs in many areas of biology and medicine:

- *pharmaceutical research for drug discovery*
- *understanding rapidly mutating viruses (such as HIV)*
- *designing genetically enhanced organisms (rice, wheat)*
- *explaining and predicting gene expression*
- *most centrally, understanding genomic evolution*

Computational Phylogenetics

- Is extremely computation-intensive.

Computational Phylogenetics

- Is extremely computation-intensive.
- Sequence data (RNA, DNA, aminoacid, and protein) has been used for over 20 years and is well understood, but methods do not scale up and sequences from different genes yield different trees.

Computational Phylogenetics

- Is extremely computation-intensive.
- Sequence data (RNA, DNA, aminoacid, and protein) has been used for over 20 years and is well understood, but methods do not scale up and sequences from different genes yield different trees.
- Genomic data (gene order and content of whole genomes) provides information on deep relationships, but is *much harder* to analyze than sequence data.

Computational Phylogenetics

- Is extremely computation-intensive.
- Sequence data (RNA, DNA, aminoacid, and protein) has been used for over 20 years and is well understood, but methods do not scale up and sequences from different genes yield different trees.
- Genomic data (gene order and content of whole genomes) provides information on deep relationships, but is *much harder* to analyze than sequence data.
- Using mixed data remains uncharted territory.

Comput. Phylo.: Theory vs. Applics

Methods developed by algorithm designers are rarely used by biologists: optimization criteria are chosen for algorithmic reasons more than biological ones.

Methods used by biologists are typically *ad hoc* and offer no guarantees: parameters are set with little understanding of their effects on efficiency or quality.

Getting the two groups to work together requires an atmosphere of mutual respect for each group's research goals and methodologies.

Comput. Phylo.: Theory vs. Applics

Computer scientists and biologists agree that *statistical consistency* (guarantee of convergence to “true tree” given enough data) is very important.

But: the *rate* of convergence is crucial, because consistency is asymptotic, yet data are finite.

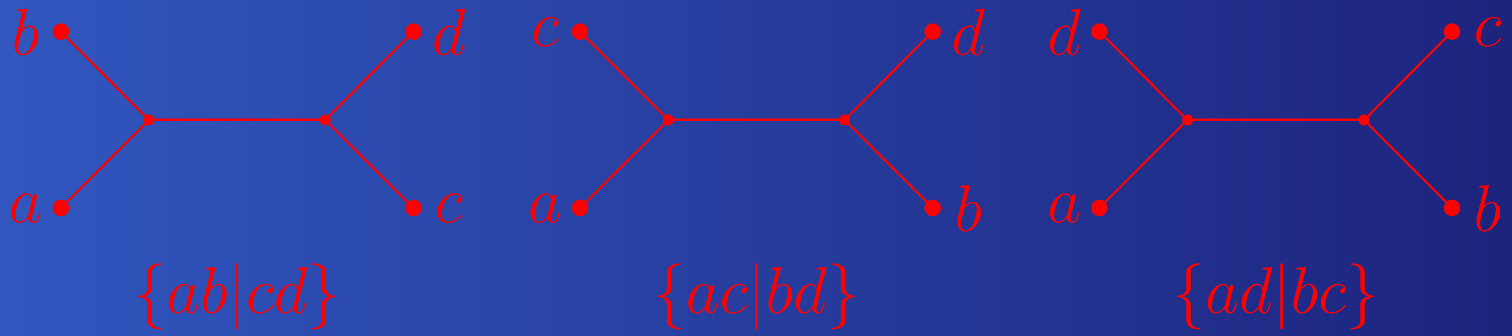
On the other hand, bounded data (# genes, # characters in sequences, # species, etc.) is good news: we can use tailored algorithms, fixed-parameter approaches, etc.

Testing: Quartet-Based Methods

The CS community has devised a variety of reconstruction methods based on *quartets*:

quartet: an unrooted binary tree on four taxa

If the taxa are $\{a, b, c, d\}$, we can use $\{ab|cd\}$ to denote the quartet that pairs a with b and c with d.



Quartet Methods

- A quartet $\{ab|cd\}$ *agrees* with a tree T if all four of its taxa are leaves of T and the subtree induced in T by the four taxa is the quartet itself.

Quartet Methods

- A quartet $\{ab|cd\}$ *agrees* with a tree T if all four of its taxa are leaves of T and the subtree induced in T by the four taxa is the quartet itself.
- If $Q(T)$ denotes the set of all quartets that agree with T, then T is uniquely characterized by $Q(T)$ and can be reconstructed from $Q(T)$ in polynomial time.

Quartet Methods

- A quartet $\{ab|cd\}$ *agrees* with a tree T if all four of its taxa are leaves of T and the subtree induced in T by the four taxa is the quartet itself.
- If $Q(T)$ denotes the set of all quartets that agree with T, then T is uniquely characterized by $Q(T)$ and can be reconstructed from $Q(T)$ in polynomial time.
- Errors occur, so quartet methods reconstruct an edge when at most some fraction α of the quartets disagree with that edge.

Testing Quartet Methods

- We need to test for datasets where we know the true tree in order to measure accuracy, hence simulations.

Testing Quartet Methods

- We need to test for datasets where we know the true tree in order to measure accuracy, hence simulations.
- But simulations must assume a model for tree generation and a model for sequence evolution, all for each fixed tree size and sequence length. The parameter space is enormous!

Testing Quartet Methods

- We need to test for datasets where we know the true tree in order to measure accuracy, hence simulations.
- But simulations must assume a model for tree generation and a model for sequence evolution, all for each fixed tree size and sequence length. The parameter space is enormous!
- We ran 20,000 separate datasets in our study (SODA'01, to appear in J. Algs.), using over 20 CPUs nearly nonstop for 6 months.

Testing Quartet Methods

And are we satisfied?

Of course not. . . Our models were among the simplest (technically, most of our events were i.i.d.), the size of our trees was limited by running times (we went up to 80 taxa only, because some quartet methods take $\Theta(n^7)$ time), etc.

As a point of reference: some biologists routinely have to deal with up to a thousand taxa, while a few want to reconstruct the “Tree of Life” with hundreds of thousands of taxa.

An Example: the Bluebell Family

Jansen's group at UT Austin provided full gene sequences for the chloroplasts of 12 species of Campanulaceae (Bluebells), plus tobacco.

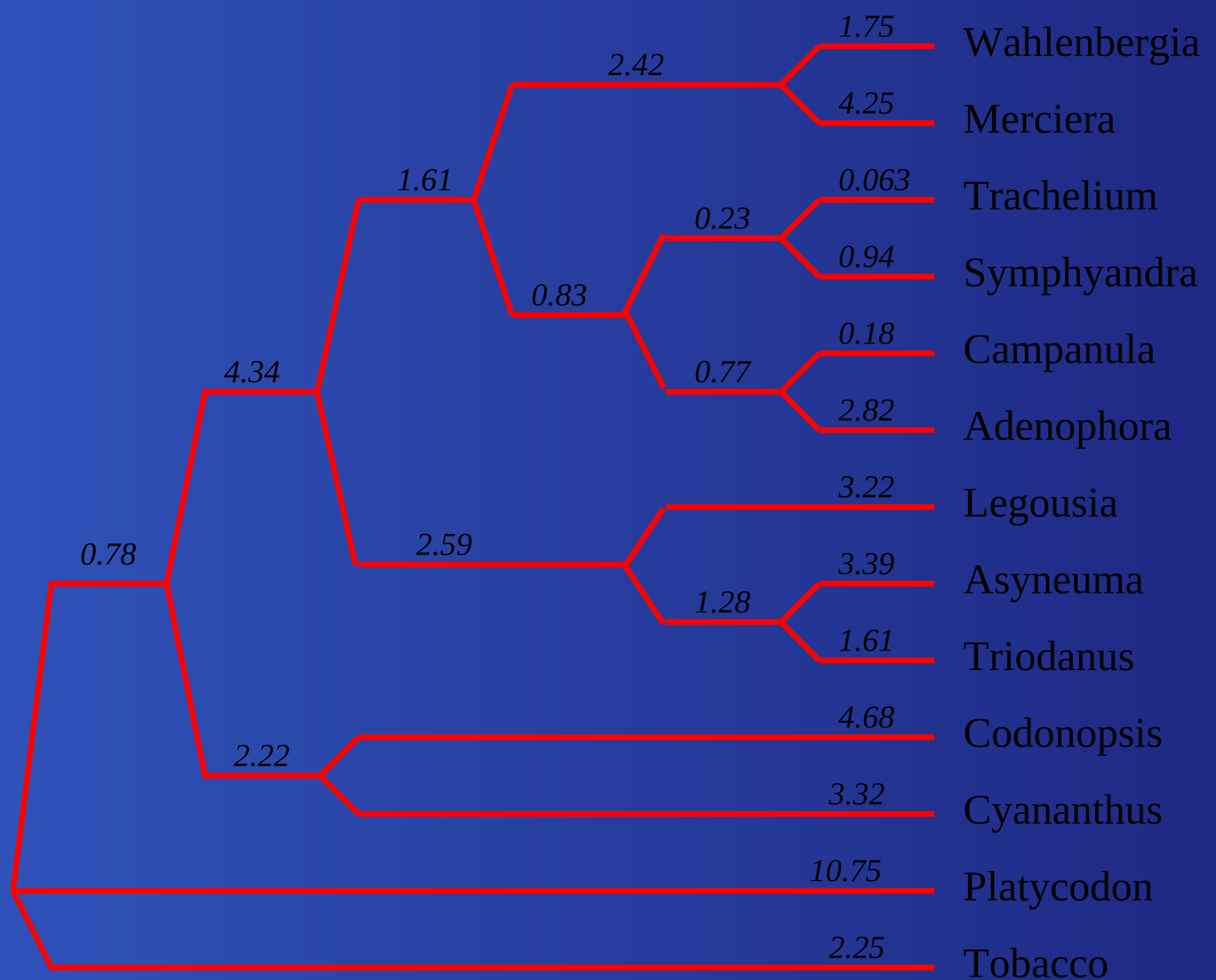
A chloroplast is a semi-independent organism that lives within plant cells and allows them to photosynthesize.

Chloroplasts have a single chromosome with about 120 genes.

Optimization target: reconstruct the phylogeny with the least total number of genomic changes.

An application of Occam's razor; biologists call this the principle of parsimony.

12 Species of *Campanulaceae*



The Bluebell Family (cont'd)

We reimplemented a tool due to D. Sankoff and M. Blanchette using algorithm engineering.

Results: *a speed-up by* **five to six orders of magnitude** *in the serial part of the code and a total speed-up by nearly* **one billion** *when run on the 512-processor* Los Lobos *supercluster at UNM.*

Reasons: *cache-awareness, detailed code optimization, better combinatorial optimization, better bounding, and parallelization.*

Breakpoint Analysis: An Overview

An iterative improvement procedure *over every tree*:

```
Initially label all internal nodes with gene orders
Repeat
    For each internal node $v$, with neighbors $A$, $B$, and $C$, do
        Solve the MPB on $A, B, C$ to yield label $m$
        If relabelling $v$ with $m$ improves the score of $T$, then do it
until no internal node can be relabelled
```

Median Problem for Breakpoints

Given 3 gene orders, represented as 3 signed permutations π_1, π_2, and π_3, find a 4th permutation π_m that minimizes the sum of the distances

$$d(\pi_1, \pi_m) + d(\pi_1, \pi_m) + d(\pi_1, \pi_m)$$

where each distance is the number of *breakpoints*, i.e., the number of adjacencies present in one permutation but not in the other.

MPB: an example

Let the (circular) permutations be

$$\begin{array}{rrrr} 1 & -2 & 4 & 3 \\ 1 & 2 & -3 & -4 \\ 2 & -3 & -4 & -1 \end{array}$$

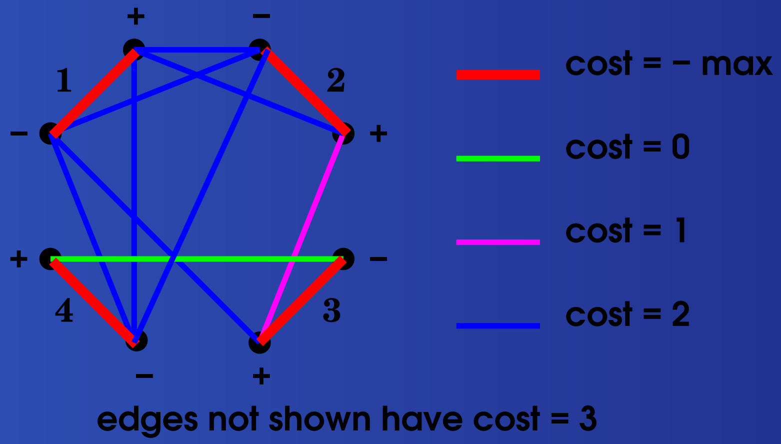
A possible median is $-1\ \ 2\ \ -3\ \ -4$, with cost 5

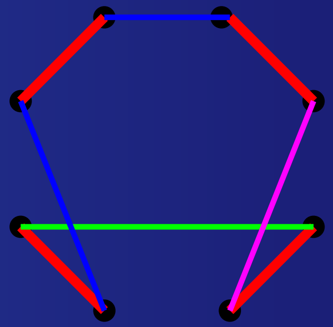
$$d((1\ \ -2\ \ 4\ \ 3),(-1\ \ 2\ \ -3\ \ -4)) = 3$$
$$d((1\ \ 2\ \ -3\ \ -4),(-1\ \ 2\ \ -3\ \ -4)) = 2$$
$$d((2\ \ -3\ \ -4\ \ -1),(-1\ \ 2\ \ -3\ \ -4)) = 0$$

MPB (cont'd)

Sankoff showed to to convert this problem to the Travelling Salesperson Problem.

+1 −2 +4 +3
+1 +2 −3 −4
+2 −3 −4 −1





an optimal solution corresponding to genome +1 +2 −3 −4

Adjacency A B becomes an edge from A to −B

The cost of an edge A −B is the number of genomes that do NOT have the adjacency A B

Re-Engineering: Coding Aspects

- Memory Use

  GRAPPA uses static allocation and has a working set size of 500KB, compared to 12MB for BPAnalysis

Re-Engineering: Coding Aspects

- Memory Use

  GRAPPA uses static allocation and has a working set size of 500KB, compared to 12MB for BPAnalysis

- Cache Awareness

  GRAPPA minimizes pointer dereferencing, has hand-unrolled loops, and re-uses allocated storage

Re-Engineering: Coding Aspects

- Memory Use

  GRAPPA uses static allocation and has a working set size of 500KB, compared to 12MB for BPAnalysis
- Cache Awareness

  GRAPPA minimizes pointer dereferencing, has hand-unrolled loops, and re-uses allocated storage
- Profiling

  Identifies bottlenecks to balance the computation

Hand-Unrolling Loops

```
/* builds adjacency lists for the TSP instance */
for (i = 1; i < num_genes-1 ; i++){
  /* First genome */
  /* g = g1->genes[i]; already set*/
  i1 = -i2; i2 = -g1->genes[i+1];
  /* now we have the endpoints of the edge */
  handle(i1,i2,adj_list,&slot,num_genes);
  /* Second genome: same deal */
  j1 = -j2; j2 = -g2->genes[i+1];
  handle(j1,j2,adj_list,&slot,num_genes);
  /* Third genome: same deal */
  k1 = -k2; k2 = -g3->genes[i+1];
  handle(k1,k2,adj_list,&slot,num_genes); }
```

Hand-Unrolling Loops

```
/* creates circular lists in each direction */
/* first adjacency */
gen1first = gen1 = ingene1[0]; gen2 = ingene1[1];
succ[gen1] = gen2; pred[gen2] = gen1;
pred[-gen1] = -gen2; succ[-gen2] = -gen1;
/* all middle adjacencies */
for (i=2; i<num_genes; i++) {
  gen1 = gen2; gen2 = ingene1[i];
  succ[gen1] = gen2; pred[gen2] = gen1;
  pred[-gen1] = -gen2; succ[-gen2] = -gen1; }
/* last adjacency */
succ[gen2] = gen1first; pred[gen1first] = gen2;
pred[-gen2] = -gen1first; succ[-gen1first] = -gen2;
```

Re-Engineering: Algorithmic Aspects

- Taking Advantage of Special Structures
  The TSP has only 2 nontrivial edges (cost 1 and 2)

Re-Engineering: Algorithmic Aspects

- Taking Advantage of Special Structures

  The TSP has only 2 nontrivial edges (cost 1 and 2)
- Using all Available Information

  TSP bounds can use full local legality test

Re-Engineering: Algorithmic Aspects

- Taking Advantage of Special Structures

  The TSP has only 2 nontrivial edges (cost 1 and 2)
- Using all Available Information

  TSP bounds can use full local legality test
- Lower Bound for Each Tree

  Triangle inequality implies that a tour of the leaves is at most twice the cost of any tree

Re-Engineering: Algorithmic Aspects

- Taking Advantage of Special Structures

  The TSP has only 2 nontrivial edges (cost 1 and 2)
- Using all Available Information

  TSP bounds can use full local legality test
- Lower Bound for Each Tree

  Triangle inequality implies that a tour of the leaves is at most twice the cost of any tree
- Find Best Lower Bound

  Try all orderings in the circular order—cheap

Re-Engineering: Algorithmic Aspects

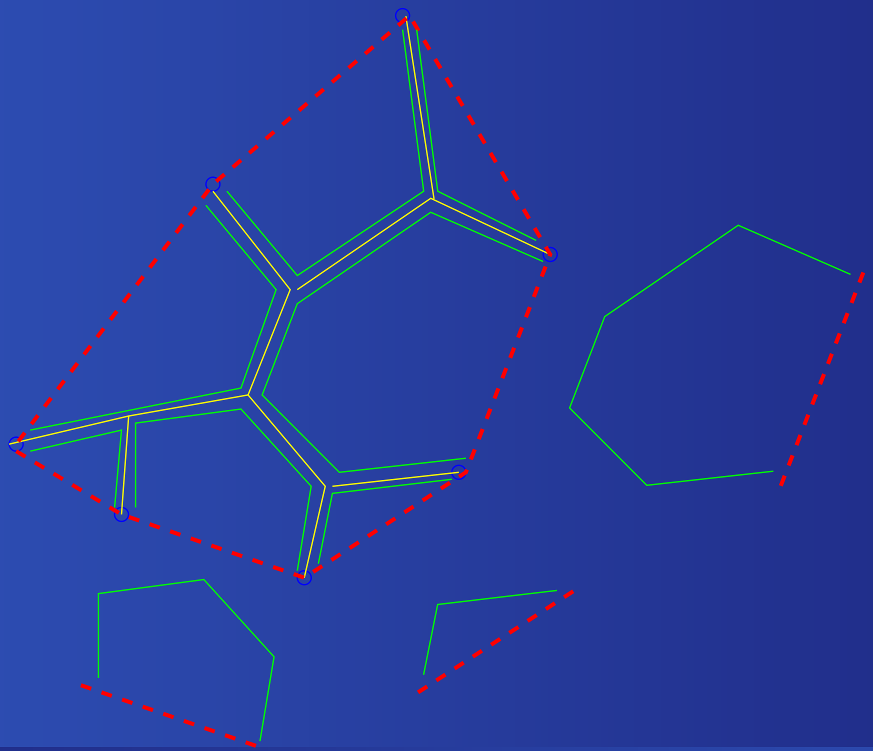
- Taking Advantage of Special Structures
  The TSP has only 2 nontrivial edges (cost 1 and 2)
- Using all Available Information
  TSP bounds can use full local legality test
- Lower Bound for Each Tree
  Triangle inequality implies that a tour of the leaves is at most twice the cost of any tree
- Find Best Lower Bound
  Try all orderings in the circular order—cheap
- Use Bounds More Efficiently
  Layered search strategy

TSP Lower Bound Computation

```
lb_new = 0;
for (i=-num_genes; i<=num_genes; i++) {
  if (degree[i] == 1) { /* nothing to do for degree 2 *
    node = adj_list[i].next;
    while (node != NULL) {
      j = node->vertex;
      if ((node->status == STAT_AVAIL) && (degree[j]==1
          && ((otherEnd[i] != j) || (picked==ncount-1))
        lb_new += node->weight;
        goto nexti; }
      node = node->next; }
    lb_new += 3; /* no usable edge of lower cost */
  nexti: ; } }
```

Re-Engineering: Tree Lower Bound

The triangle inequality implies that a tour of the leaves is at most twice the cost of any tree (the “twice around the tree” heuristic for TSP).



Re-Engineering: Tree Lower Bound

In biology, distances are often *additive* or nearly so:

$$d(a,c) \approx d(a,b) + d(b,c)$$

which makes for very tight bounds.

There are $n!$ possible tours of the n leaves, each a lower bound on the tree cost; the tour with the largest total distance can be computed by heuristics or brute force (all distances are precomputed).

Re-Engineering: Algorithmic Aspects

Large distances can still defeat bounding, so we devised a layered search:

- Search through all trees over and over again with a target score initialized at the lower bound and increases by 1 after each stage.
- Trees that cannot be eliminated are scored, then stored in a hash table, so next encounter is a lookup.
- Initially most trees need not be scored; we get a threshold effect close to the optimal value, where we have a sharp upper bound.

Re-Engineering: Parallel Aspects

- Efficient Tree Generation

  Avoid multi-precision arithmetic, allow generation from any count with variable gap—provides parallel generation and also sampling of search space

Re-Engineering: Parallel Aspects

- Efficient Tree Generation

  Avoid multi-precision arithmetic, allow generation from any count with variable gap—provides parallel generation and also sampling of search space

- (current) Portable MPI Implementation

  Exploits “embarrassing” parallelism (each processor handles a fraction of the trees)

Re-Engineering: Parallel Aspects

- Efficient Tree Generation
  Avoid multi-precision arithmetic, allow generation from any count with variable gap—provides parallel generation and also sampling of search space
- (current) Portable MPI Implementation
  Exploits “embarrassing” parallelism (each processor handles a fraction of the trees)
- (future) Hybrid Mode Implementation
  Exploits shared-memory parallelism at each node for combinatorial optimization

Algorithmic Benefits of Re-Engineerin

- In re-engineering an algorithm, one comes to understand it better than anyone else—and so one often finds algorithmic improvements.

Algorithmic Benefits of Re-Engineerin

- In re-engineering an algorithm, one comes to understand it better than anyone else—and so one often finds algorithmic improvements.
- We developed the first true linear-time algorithm for computing the inversion distance between two signed permutations (WADS’01, to appear in J. Comput. Biol..

Algorithmic Benefits of Re-Engineerin

- In re-engineering an algorithm, one comes to understand it better than anyone else—and so one often finds algorithmic improvements.
- We developed the first true linear-time algorithm for computing the inversion distance between two signed permutations (WADS'01, to appear in J. Comput. Biol..
- We developed the first family of fast-converging phylogeny reconstruction algorithms for sequence data (SODA'01).

Impact in Computational Biology

- Much faster implementations can alter the practice of research in biology and medicine.
  Reducing the time of an analysis from a year down to an hour makes an enormous difference in the pace and cost of drug discovery and development.

Impact in Computational Biology

- Much faster implementations can alter the practice of research in biology and medicine.
  Reducing the time of an analysis from a year down to an hour makes an enormous difference in the pace and cost of drug discovery and development.
- Fast and accurate analysis software enables researchers to pursue more leads, develop better intuition on small datasets, and form new conjectures about biological mechanisms.
  Even when the software does not scale up to "industrial-strength".