The History Bound and ILP

Julia Matsieva and Dan Gusfield

UC Davis

March 15, 2017

Bad News for Tree Huggers

Two captive *Heliconius* species mate, illustrating the rampant hybridization among plants and animals in nature.

SHAKING UP THE TREE OF LIFE

Species were once thought to keep to themselves. Now, hybrids are turning up everywhere, challenging evolutionary theory

By **Elizabeth Pennisi**

In 2010, a comparison between the genomes of a Neandertal and people today settled what anthropologists and geneticists had debated for decades: Our ancestors had indeed mated with their archaic cousins, producing hybrid children. They, in turn, had mated with other modern humans, leaving their distant descendants—us—with a permanent Neandertal legacy. Not long afterward, DNA from another archaic human population, the Denisovans, also showed up in the modern human genome, telling a similar story (*Science*, 23 December 2011, p. 1629).

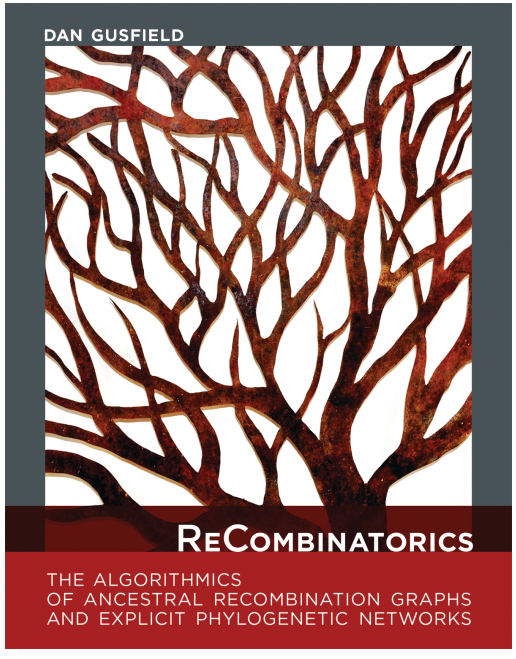
For researchers and the public alike, this evidence of interbreeding among distinct human populations—so different some still argue there was more than one species—created a shock wave. Suddenly hybridization "just captured our imagination," says Michael Arnold, an evolutionary biologist at the University of Georgia in Athens. "That genomic information overturned the assumption that everyone had."

The techniques that revealed the Neandertal and Denisovan legacy in our own genome are now making it possible to peer into the genomic histories of many organisms to check for interbreeding. The result: "Almost every genome study where people use sensitive techniques for detecting hybridization, we find [it]—we are finding hybridization events where no one expected them," says Loren Reiseberg, an evolutionary biologist at the University of British Columbia in Vancouver, Canada.

More Bad News

Far more convincingly even than the (also highly convincing) fossil evidence, the evidence from comparisons among genes is converging, rapidly and decisively, on a single great tree of life. ...
It is the consistency of agreement among all the different genes in the genome that gives us confidence, not only in the historical accuracy of the consensus tree itself, ...
R. Dawkins, The Greatest Show on Earth

So there is no "Species Tree - Gene Tree" disagreement, and no need to resolve discordance.


DAN GUSFIELD
RECOMBINATORICS
THE ALGORITHMICS
OF ANCESTRAL RECOMBINATION GRAPHS
AND EXPLICIT PHYLOGENETIC NETWORKS

Recombination in Meiosis

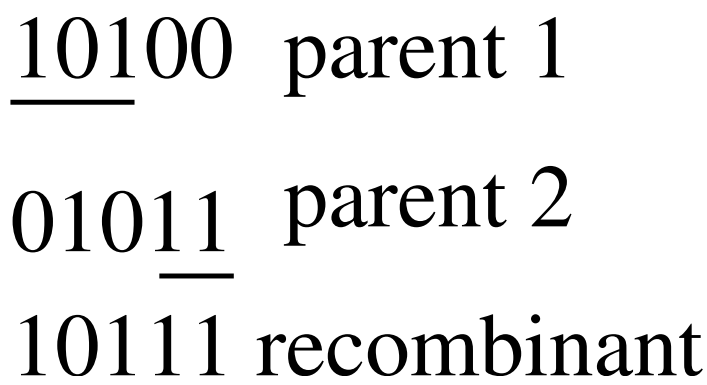


Figure : Crossing-over in Meiosis. A single-crossover recombination, and a recombinant sequence. The prefix (underlined) contributed by parental sequence 1 consists of the first three characters of sequence 1. The suffix (underlined) contributed by parental sequence 2 consists of the last two characters of sequence 2.

Evolution of Sequences with Mutation and Recombination

The mutation model is the *infinite sites model*: One mutation per site in the history of the sequences. This is the same as in the Perfect Phylogeny Model.

But, *Not* every set of binary sequences can be derived on a Perfect Phylogeny. The NASC is known as *Compatibility* in the phylogenetics world, and the *Four Gametes Condition* in population genetics.

In addition to mutation, *recombinations* are allowed and can create new sequences. Every set of binary sequences can be generated this way.

Ancestral Recombination Graph (ARG)

The full evolutionary history that derives a set of sequences M, with one mutation per site, and recombinations allowed, is represented by an *Ancestral Recombination Graph (ARG)*.

An ARG

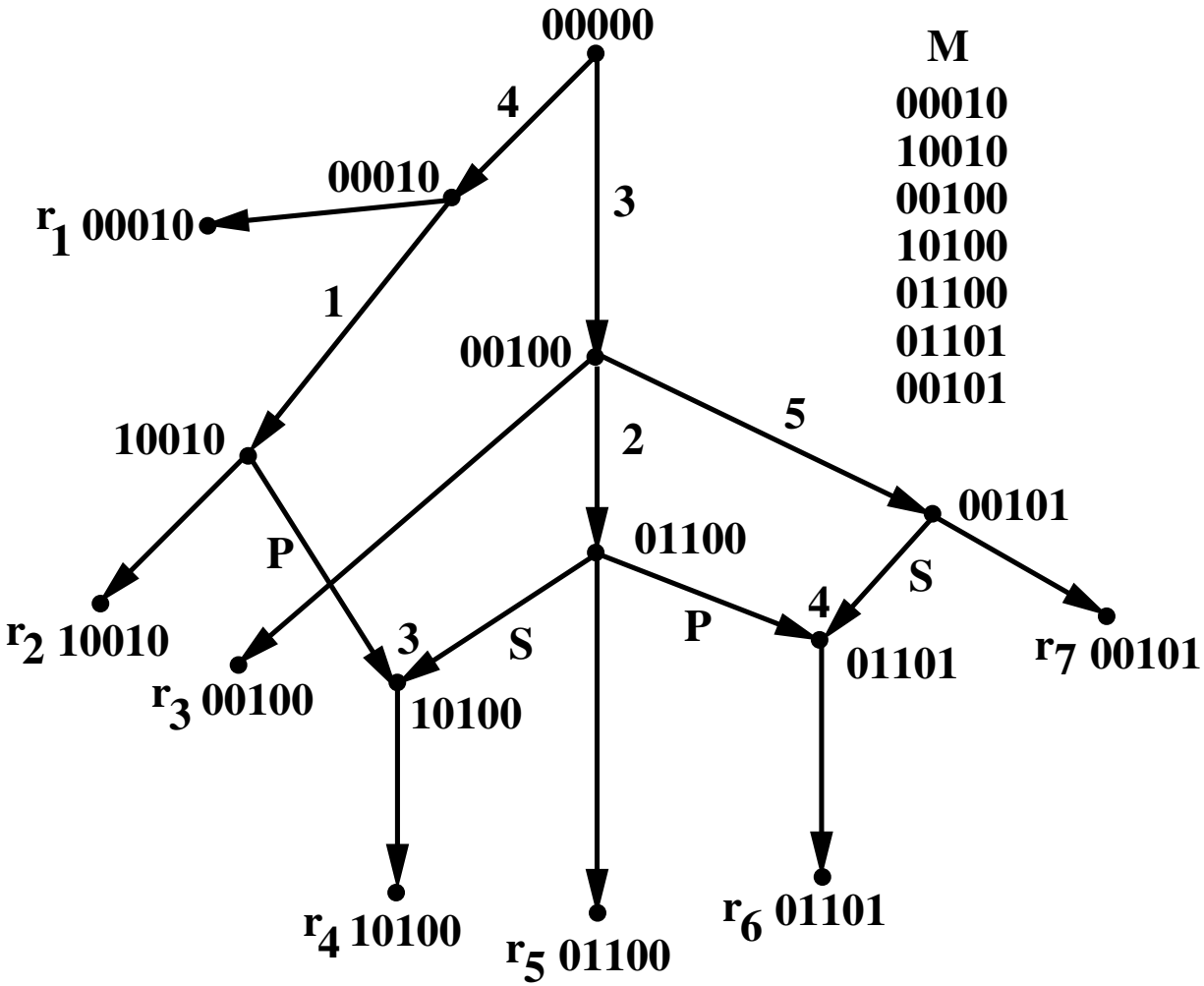


Figure : An ARG N with two recombination nodes, each representing a single-crossover recombination event. The matrix of sequences M that are derived by N is shown at the right.

$Rmin(M)$

Given M we want an ARG that derives M using the *minimum* number of recombination nodes. That minimum number is denoted $Rmin(M)$.

Finding $Rmin(M)$ is NP-hard, and the fastest algorithms guaranteed to compute $Rmin(M)$ take *Super-Exponential* time.

So we want effective algorithms to computer *Lower Bounds* on $Rmin(M)$.

Lower Bounds on $Rmin(M)$

We have examined about fifteen lower bounds on $Rmin(M)$. The best (both empirically, and by some theoretical results) is based on the *History Bound* created in 2002 by Myers and Griffiths.

Initially, the History Bound was only defined *proceduraly* – it is what the Myers-Griffiths algorithm produces. That algorithms runs in $\Omega(n!)$ time, for n taxa. Not satisfying.

Bafna and Bansal showed that computing the History Bound is NP-hard, and gave an $\Theta(2^n)$ time algorithm to compute it – *despite* having *no* static, non-procedural, functional definition for the History Bound (Amazing!).

A non-procedural definition of the History Bound?

At the meeting "The future of phylogenetic networks" at Leiden about five years ago, at the end of my talk, I stated the open (to me) problem of finding a non-procedural definition of the History Bound. Leo van Iersel, Steven Kelk, Celine Scornavacca, Chris Whidden immediately lept to their feet and declared that they already knew it, but hadn't written a proof or a paper.

After more than a year had passed, I asked Julia Matsieva, an MS student then, to try to verify their ideas. She wrote a formal proof, and developed related algorithms; and we (minus Leo) together published the non-procedural definition, recently in TCBB.

So what is the History Bound?

I won't discuss the procedural definition of the original Griffiths-Myers algorithm, or the Bafna-Bansal version, but just say that they are matrix-centric and at first do not seem to have anything to do with networks.

Instead I will explain the non-procedural definition. First, I have to define *reticulation networks*.

The most important definition

Let M be a binary matrix. A *Directed Acyclic Graph (DAG)* $\mathcal{D}$ *displays* character (column) c of M if

a) There is some tree $\mathcal{T}_c$ embedded in $\mathcal{D}$ that reaches *all* of the leaves of $\mathcal{D}$, and

b) There is an edge e in $\mathcal{T}_c$, such that the set of leaves of $\mathcal{T}_c$ that are reachable from e is *exactly* the taxa (rows) with character c (value 1 in column c).

The taxa with character c is called a "cluster". So M defines a set of clusters.

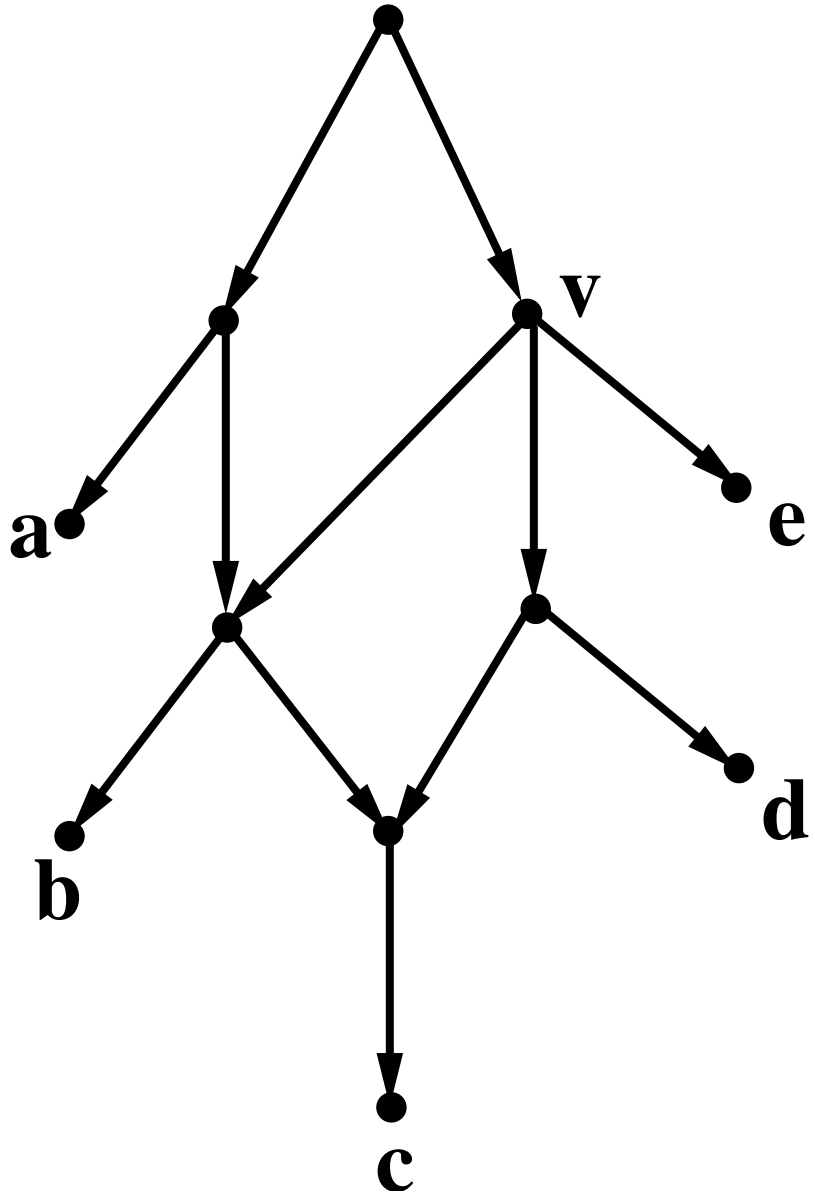


Figure : DAG $\mathcal{D}$.

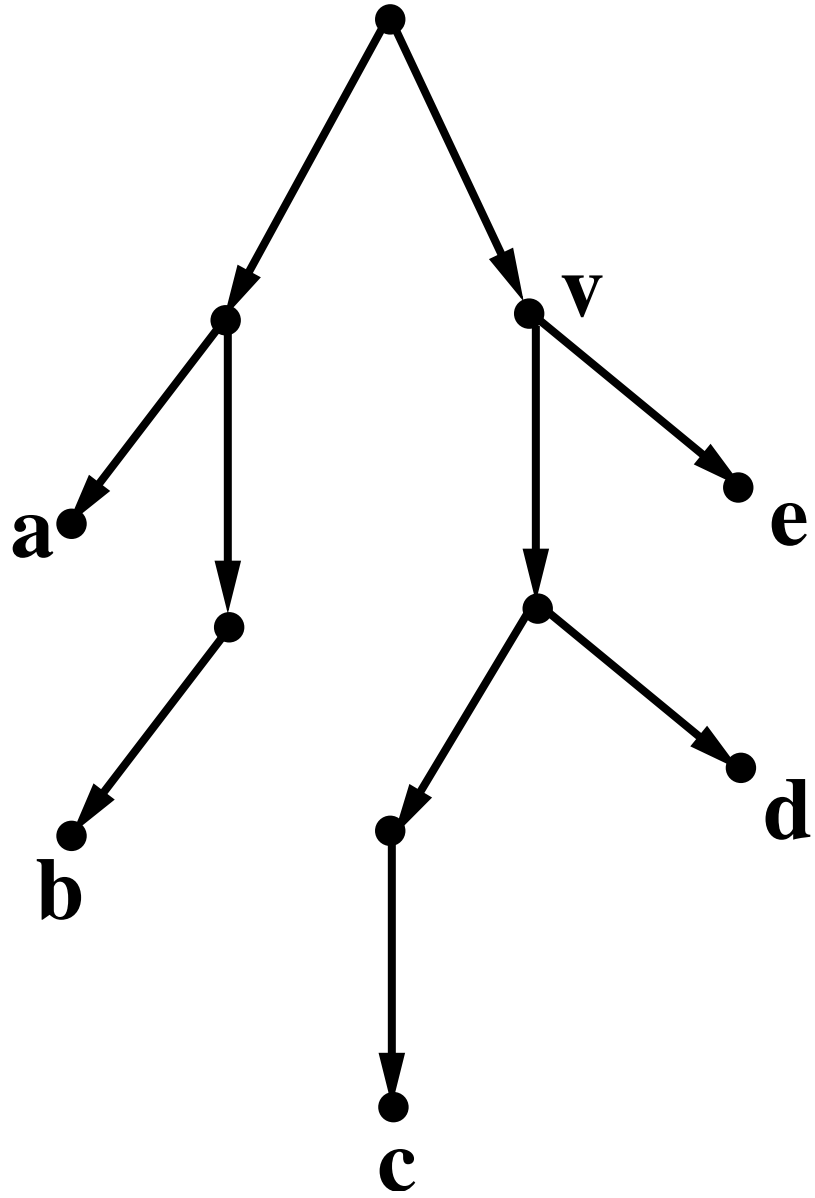


Figure : A subtree of $\mathcal{D}$ that reaches all leaves. The tree displays $\{c, d, e\}$, and other clusters.

Reticulation Networks

Given M, a DAG that displays *all* of the characters of M is called a *Reticulation Network* for M.

Unlike an ARG, where the maximum in-degree of any node is at most two, in a reticulation network the in-degree is unconstrained.

Any node in a reticulation network with in-degree more than one is called a *reticulation* node.

Finally, the non-procedural definition

Given M, the History Lower Bound on the minimum number of recombinations needed in any ARG that creates M, is *exactly* the minimum number of reticulation nodes needed in any reticulation network for M.

So the minimum number of reticulation nodes needed is computable in exponential time, even though we don't know how to compute $Rmin(M)$ in exponential time.

It is easily verified that an ARG for M is also a reticulation network for M, so it is no surprise that the minimum number of reticulation nodes needed in a reticulation network for M is a lower bound on $Rmin(M)$. The surprise is that this lower bound is exactly what the History Bound is.

OK, Now What?

Having a non-procedural, functional definition should allow us to reason more deeply and productively about the History Bound, and to find *alternative* ways to compute it.

We tried *Top-Down* branching, search algorithms that work well - much faster in practice than the $\Theta(2^n)$ method of Bafna and Bansal.

And, we tried three different approaches using *Integer Linear Programming (ILP)*. They are all bad. I will discuss one that illustrates a general idea.

Designing Networks with ILP

- → Constructing a reticulation network for a matrix M, using the minimum number of reticulation nodes is very particular *network design* problem.
- → The general network design problem is very related to *Multi-Commodity Network Flow* problems.
- → ILP works well for Multi-Commodity flow.
- → Ergo - ILP should be good for computing an optimal reticulation network for M.

How to design a reticulation network $\mathcal{D}$ using network flow, and ILP

Start with a *Super-Network* H that contains all the edges, and more, that might be in the optimal network $\mathcal{D}$.

A super-network H: If M has n taxa and m characters, then the following graph works: Create a complete graph with $m + n$ "internal" nodes, i.e., where there is an edge between each pair of nodes; then add a root node r and a directed edge from r to each of the internal nodes; then add n leaves corresponding to the n input taxa in M, and add a directed edge from each of the internal nodes to each of the leaves.

Then, the network design problem is to choose a *subset* of the edges of H for $\mathcal{D}$. For each edge (i, j) in H we use a *binary* ILP variable $X(i, j)$, where we choose (i, j) for $\mathcal{D}$, if and only if the value of $X(i, j)$ is set to 1.

How to satisfy the first requirement for $\mathcal{D}$

Recall: A DAG $\mathcal{D}$ *displays* character c of M if
a) There is some tree $\mathcal{T}_c$ embedded in $\mathcal{D}$ that reaches *all* of the leaves of $\mathcal{D}$,
and b) ....

To satisfy a) we think of the all-zero node in H as a *source* with n units of commodity c; and think of each node in H that is labeled by a taxon in M as a *sink*, with a demand for one unit of commodity c. Each other node in H must satisfy the normal Kirkoff *flow conservation* constraint. Implementing flow as an ILP is standard.

A flow F_c from the source to the sinks selects a subset of edges of H that almost satisfies a) for character c. But F_c must specify a tree (which also forces the flow to be integral). For that, we add constraints that for any node v in H, *at most* one edge into v has flow in F_c. How to do that?

$X_c(i,j)$ is a binary ILP variable indicating whether the directed edge (i,j) is used in the tree $\mathcal{T}_c$, and $T_c(j)$ is a binary variable that records whether node j is a non-root node in $\mathcal{T}_c$. Specifically, we have for c and (i,j): $X_c(i,j) \leq F_c(i,j)$,

$$F_c(i,j) - n \times X_c(i,j) \leq 0,$$

$$T_c(j) = \sum_{i \neq j} X_c(i,j) \leq 1.$$

Of course, we create these inequalities for each c and edge (i,j). The result is a set of chosen edges that satisfies requirement a) for each character c. Then $\mathcal{D}$ is the superposition of those trees. $\mathcal{D}$ is specified by the $X(i,j)$ variables, where for each edge (i,j) in H:

$$X(i,j) \leq \sum_c X_c(i,j)$$

$$\sum_c X_c(i,j) - m \times X(i,j) \leq 0$$

Actually

We can replace

$$\sum_c X_c(i,j) - m \times X(i,j) \leq 0$$

with

$$X(i,j) \geq X_c(i,j)$$

Or

$$F_c(i,j) - n \times X(i,j) \leq 0$$

for each c.
Is one way better than the other? That is an empirical question.

How to minimize the number of reticulation nodes?

$R(j)$ is a binary ILP variable indicating whether node j is a reticulation node in $\mathcal{D}$. Then, for each node j:

$$\sum_{i \neq j} X(i,j) - m \times R(j) \leq 1$$

and we have the objective function:

$$\min \sum_{j} R(j)$$

Now how do we incorporate requirement b)?

This is trickier.

We think of a new source outside of H, that specifies exactly one node v in $\mathcal{T}_c$, not the all-zero node of H, as the "root of the cluster tree" for character c. Then, the unique edge in tree $\mathcal{T}_c$ into v is the edge e needed in requirement b).

To specify the root of the cluster tree for c, we use one binary variable $RCT_c(i)$, for each node i not the root of H.

We also have binary variables $CT_c(i)$ indicating whether or not node i is in the cluster tree for c. Then requirement b) is achieved with the inequalities that follow:

Forward Inequalities for Requirement b)

$$RCT_c(i) \leq T_c(i)$$

Meaning: Node i can be chosen as the root of the cluster tree for c, only if i is in tree $\mathcal{T}_c$.

$$\sum_{i \in H, \text{ and not the root of H}} RCT_c(i) = 1$$

Meaning: Exactly one root of the cluster tree for c is chosen, and it isn't the root of H.

$$RCT_c(i) \leq CT_c(i)$$

Meaning: If i is the chosen root of the cluster tree for c, then i is in the cluster tree for c - duh!

Continuing

Further,
For any node i that is not a node representing a taxon with character c, and any edge (i, j) in H, we want

If $CT_c(i)$ AND $X_c(i, j)$ then $CT_c(j)$, which is implemented as:

$$CT_c(i) + X_c(i, j) - CT_c(j) \leq 1$$

Then for a leaf j which has character c,

$$CT_c(j) = 1,$$

and for a leaf j which does not have character c,

$$CT_c(j) = 0.$$

We Conclude

For each c, the forward inequalities above will select a proper subtree of $\mathcal{T}_c$, consisting of a root i and every node that is reachable from i using edges in $\mathcal{T}_c$. The variable $CT_c(j)$ will be set to 1, if node j is one such node, and since $CT_c(j) = 0$ for leaves which do not have character c, none of those leaves will be in the subtree.

So the subtree contains every leaf representing a taxon that has character c, and is reachable from i, and it does not reach any other leaves.

However, it is not necessarilly true that *every* leaf with character c is reached, so we need the backward (converse) inequalities also.

Backward Inequalities

We need to implement the requirement that $CT_c(j)$ is set to 1, *only if* $RCT_c(j) = 1$, OR for some $i \neq j$, $CT_c(i) = 1$ AND $X_c(i,j)$. For that, we have a new binary variable $Z_c(i,j)$ for each $i \neq j$. This variable will be set to one *only if* the AND clause is satisfied with node i:

$$2 \times Z_c(i,j) - CT_c(i) - X_c(i,j) \leq 0$$

and

$$CT_c(j) - \sum_{i \neq j} Z_c(i,j) - RCT_c(j) \leq 0.$$

Finis

The above ILP formulation is a bit different from what we have implemented, but is (hopefully) mathematically correct. What we implemented takes too long to solve - using Gurobi or Cplex. So, finding a better ILP formulation is an open question.