

Beaches of islands of tractability: Algorithms for parsimony and minimum perfect phylogeny haplotyping problems

Leo van Iersel¹, Judith Keijsper¹, Steven Kelk², Leen Stougie^{1,2}

(1) Technische Universiteit Eindhoven (TU/e)

(2) Centrum voor Wiskunde en Informatica (CWI), Amsterdam

Email: S.M.Kelk@cwi.nl

Web: <http://homepages.cwi.nl/~kelk>

- **Parsimony Haplotyping (PH)**

- A genotype is modelled as a string over the alphabet $\{0,1,2\}$.
- A haplotype is modelled as a string over the alphabet $\{0,1\}$.
- Two haplotypes h_1, h_2 **resolve** a genotype g iff:-
 - At each site where g has a 0, h_1 and h_2 both have a 0;
 - At each site where g has a 1, h_1 and h_2 both have a 1;
 - At each site where g has a 2, h_1 and h_2 are different i.e. 0/1 or 1/0.
- If g is resolved by h_1 and h_2 we write $g = h_1 + h_2$. But in general a genotype may be resolved by many different pairs of haplotypes.
- This leads to a natural optimisation problem...



Parsimony Haplotyping (PH)

Input: A set of n genotypes G , each of length m ;

Output: The smallest possible set of length- m haplotypes H such that each genotype is resolved by some pair of the haplotypes (in which case we say that H resolves G .)

- Toy example: suppose the input G is: 122, 201, 022.
- A smallest possible set of haplotypes that resolves G has size 4, e.g. 101, 001, 110, 010.

$$\begin{array}{r} 101 \\ 110+ \\ \hline 122 \end{array} \quad \begin{array}{r} 001 \\ 101+ \\ \hline 201 \end{array} \quad \begin{array}{r} 010 \\ 001+ \\ \hline 022 \end{array}$$



Adding a phylogenetic restriction to the PH model

Minimum Perfect Phylogeny Haplotyping (MPPH)

Input: A set of n genotypes G , each of length m

Output: The smallest possible set of length- m haplotypes H such that each genotype is resolved by some pair of the haplotypes **and such that the haplotypes H permit a perfect phylogeny. Or 'null' if no solutions exist.**

- A set of haplotypes permits an (undirected) **perfect phylogeny** iff the haplotypes can be placed at the leaves of an unrooted evolutionary tree, where each site mutates at most once.
- Well-known fact: assuming the haplotypes H are arranged as the rows of a matrix, H permits a perfect phylogeny iff the following **forbidden submatrix** F does not appear:

00
01
10
11



Bounded instances of PH and MPPH

- Much work has been done on PH, but very little on MPPH. Both problems are, in general, NP-hard. (The problem of determining whether ANY perfect phylogeny solution exists, PPH, is linear-time solvable.)
- Inspired by (amongst others) the paper “Islands of tractability for parsimony haplotyping” (Sharan, Halldórsson, Istrail - 2005) we wanted to explore the interface between ‘hard’ and ‘easy’ instances of these problems.
- $PH(j,k)$ is the PH problem where each genotype has at most j 2s per row, and (assuming the input genotypes are given as the rows of a matrix) at most k 2s per column.
- A ‘*’ denotes no restriction e.g. $PH(3,*)$ is the problem with no restriction on the number of 2s per column, but at most three 2s per genotype.
- Same definition for MPPH.



Before and after our paper

Parsimony Haplotyping (PH)

- PH(4,3) is APX-hard (Sharan, Halldórsson, Istrail – 2005)
 - PH(3,*) is APX-hard (Lancia, Pinotti, Rizzi - 2004.)
 - PH(2,*) is in P (Lancia et al, independently Cilibrasi et al - 2005.)
 - PH(3,3) is APX-hard
 - PH(*,1) is in P
- } our results

Minimum Perfect Phylogeny Haplotyping (MPPH)

- NP-hard in general (Bafna, Gusfield, Hannenhalli, Yooseph - 2004.)
 - MPPH(3,3) is APX-hard.
 - MPPH(2,*) is in P, by reduction to PH(2,*)
 - MPPH(*,1) is in P, by reduction to PH(*,1)
- } our results



- For both PH and MPPH, the main open problem left is $(*,2)$.
- Sharan et al showed that $PH(*,2)$ is in P for 'clique' instances.
- We found the analogous result for $MPPH(*,2)$. Surprisingly complicated!

- Here I sketch our polynomial-time algorithms for:
 - $PH(*,1)$
 - $MPPH(*,1)$
 - $MPPH(*,2)$ on 'clique' instances.



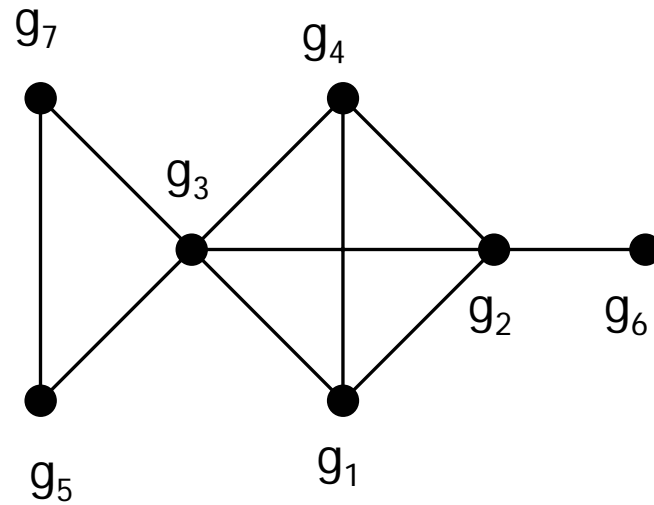
Compatibility and consistency

- Two genotypes g_1 and g_2 are said to be **compatible** iff at each site where they are non-equal, one of the two genotypes has a 2. For example **020** and **210** are compatible but **020** and **120** are not. (Incompatible genotypes can never share haplotypes.)
- The **Compatibility Graph** $\text{Comp}(G)$ of a set of genotypes G has:
 - a vertex for every genotype g in G ;
 - an edge between two vertices g_1, g_2 iff g_1 and g_2 are compatible.
- A haplotype h is said to be **consistent** with g iff at each site where g and h are non-equal, g has a 2. For example **010** is consistent with **022** but **110** is not. If g_1 and g_2 are compatible and h is consistent with both, we write $g_1 \sim_h g_2$.



g_1	0	0	1	0	2	0	1
g_2	2	0	2	0	0	0	1
g_3	0	0	1	2	0	0	1
g_4	0	0	1	0	0	0	2
g_5	0	0	1	1	0	2	1
g_6	1	2	0	0	0	0	1
g_7	0	0	1	1	0	0	1

Example of an input genotype matrix G



Compatibility graph $\text{Comp}(G)$



The $PH(, 1)$ compatibility graph has a special structure*

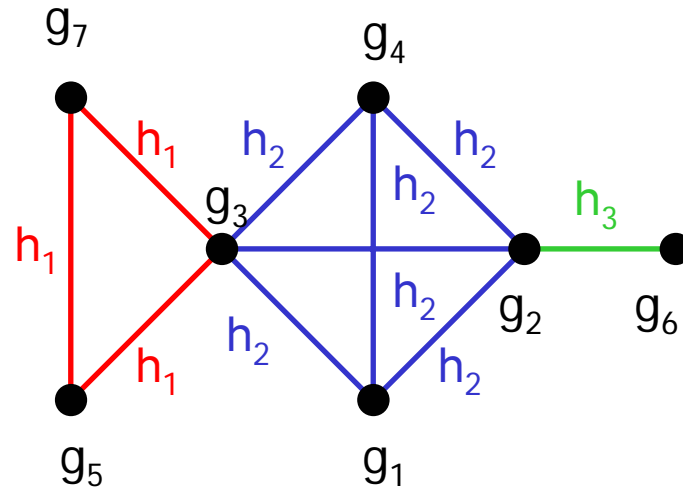
In $PH(*, 1)$, the following facts hold:

- (1) If two genotypes g_1 and g_2 are compatible, then there is **precisely one** haplotype h that is consistent with both of them. (At each column, read off the non-2 element.) So each edge in $\text{Comp}(G)$ corresponds to a unique haplotype.
- (2) The compatibility graph $\text{Comp}(G)$ is a 1-sum of cliques, and is thus **chordal**.

Related to (1), the following is also true:

- (3) Given any mutually compatible set of genotypes (which thus appear as a clique c in the compatibility graph), there is precisely **one** haplotype that is consistent with all of them. (At each column, read off the non-2 element.) We call this the **clique haplotype** h_c for that clique c .

g_1	0	0	1	0	2	0	1
g_2	2	0	2	0	0	0	1
g_3	0	0	1	2	0	0	1
g_4	0	0	1	0	0	0	2
g_5	0	0	1	1	0	2	1
g_6	1	2	0	0	0	0	1
g_7	0	0	1	1	0	0	1



Clique haplotype for red clique (i.e. h_1) is: 0011001

Clique haplotype for blue clique (i.e. h_2) is: 0010001

Clique haplotype for green clique (i.e. h_3) is: 1000001




Algorithm idea:

- The graph $\text{Comp}(G)$ is chordal, and thus has a simplicial vertex. (A vertex whose closed neighbourhood is a clique.) Removing a simplicial vertex (and its incident edges) still leaves a chordal graph.
- We build the solution H by repeatedly 'peeling' vertices away from $\text{Comp}(G)$, each time adding haplotypes to a haplotype set H' (that is initially empty.) Specifically:
 - At every iteration, locate a simplicial vertex (genotype) g . Depending on which haplotypes are already in H' , add (at most) two haplotypes h_1 and h_2 (such that $g = h_1 + h_2$) to H' , and then remove g and its incident edges from $\text{Comp}(G)$.

Repeat until $\text{Comp}(G)$ is empty. Return H' as the final solution H .

- But, for each g , how do we decide which h_1 and h_2 to add to H' ? Tempting to always use the clique haplotypes, but that's not always optimal...

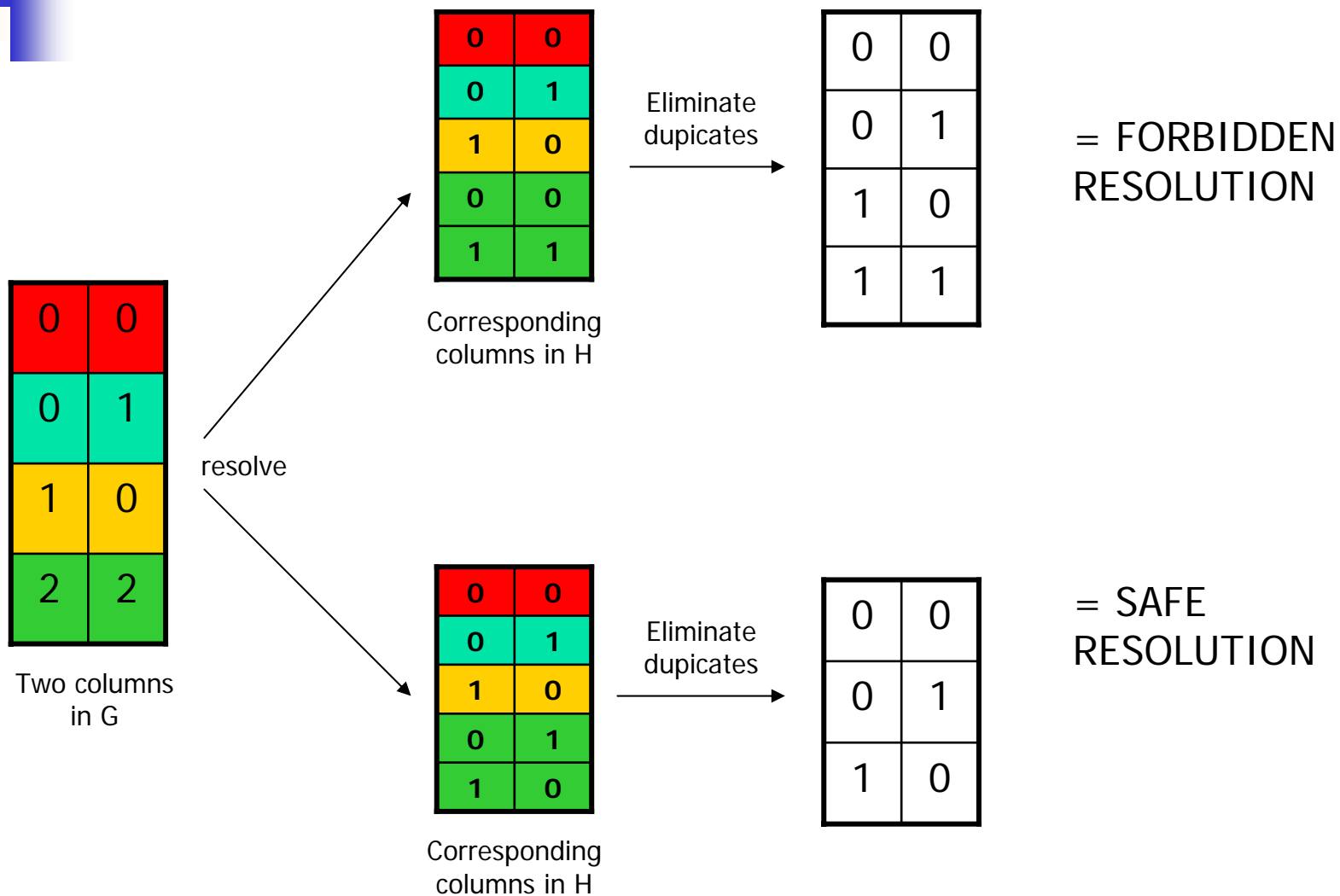


Try each of the following steps in order and stop once a step has been executed. (h_c is the clique haplotype corresponding to g , defined at the start of the algorithm.)

1. If g has no 2s, simply add g to H' .
2. If g is already resolved by some pair of haplotypes in H' , there's no need to add new haplotypes to H' .
3. If just adding the clique haplotype of g to H' allows H' to resolve g , do it.
4. If just adding some non-clique haplotype to H' allows H' to resolve g , do it.
5. If g is not an isolated vertex, add $\{h_c, h\}$ to H' (where $g = h_c + h$.)
6. Add any two haplotypes h_1, h_2 to H' such that $g = h_1 + h_2$.

Optimality of algorithm proved by induction. (Not given here.)

In MPPH(, 1) some resolutions are forbidden...*



Reducing $MPPH(*, 1)$ to $PH(*, 1)$ by discouraging forbidden resolutions

- In $PH(*, 1)$, there will – for each pair of columns – be at most one row that is 22, and if such a row exists there will be no other 2s in those columns.
- Idea: to reduce $MPPH(*, 1)$ to $PH(*, 1)$, we have to discourage such rows from resolving the forbidden way.
- We do this by adding, for each pair of columns where a 22 can be seen, a ‘blocking’ column that biases resolutions in favour of the safe way.

0	0	
0	1	
1	0	
2	2	

becomes →

0	0	0
0	1	1
1	0	1
2	2	1

Idea is that, within $PH(*, 1)$, the 22 might still choose the forbidden resolution (e.g. 00/11 in this case) but the haplotypes used to do this cannot be shared by any other genotypes, because of the extra column. So just as good, if not better, to choose the safe resolution. So (assuming feasibility) there exist optimal solutions to $PH(*, 1)$ where all such 22 resolutions are safe.



MPPH(,2) on clique instances*

- Sharan et al showed that $PH(*,2)$ is in P if the compatibility graph is a clique i.e. if all genotypes are mutually compatible.
- We have proved the same for $MPPH(*,2)$.
- Surprisingly complicated! Main complication is that, unlike the $PH(*,2)$ case, using the all-0 haplotype can sometimes cause forbidden resolutions.
- Here I demonstrate the most important ideas/steps behind the algorithm.



Foundations for the $MPPH(*,2)$ -clique algorithm

- We can assume (by a relabelling argument) that the input matrix is restricted to $\{0,2\}$.
- If there are 3 or more genotypes in the input, the only haplotype that is consistent with all genotypes is the all-0 haplotype (because every column must contain a 0.)
- The **restricted compatibility graph** $\text{ResComp}(G)$ is defined as follows:-
 - A vertex for every genotype g ;
 - An edge between two genotypes g_1 and g_2 iff **there exists some haplotype h not equal to the all-0 haplotype, such that $g_1 \sim_h g_2$.**
(Equivalently, iff there exists some column where g_1 and g_2 both have 2s.)
- For an edge (g_1, g_2) in $\text{ResComp}(G)$, define the **edge haplotype** for that edge as the haplotype having 1s in columns where g_1 and g_2 both have a 2, and 0s everywhere else. (These are useful for avoiding the forbidden submatrix!)

Critical observations

1. If the input G does permit solutions, then every vertex in the restricted compatibility graph has degree at most 2 i.e. consists of paths, cycles and isolated vertices.
2. It is never permitted to resolve a degree-2 vertex in the restricted compatibility graph with the all-0 haplotype.
3. If you resolve a genotype g with haplotypes h_1 and h_2 , and h_1 and h_2 are both consistent with other haplotypes, then h_1 and h_2 are **uniquely defined** (and, for deg-2 vertices, are equal to the adjacent edge haplotypes.)
4. Genotypes that cannot share both their haplotypes, can be thought of as having at least one **private haplotype** that no other genotypes can share.

2	2	2
2	0	0
0	2	0
0	0	2

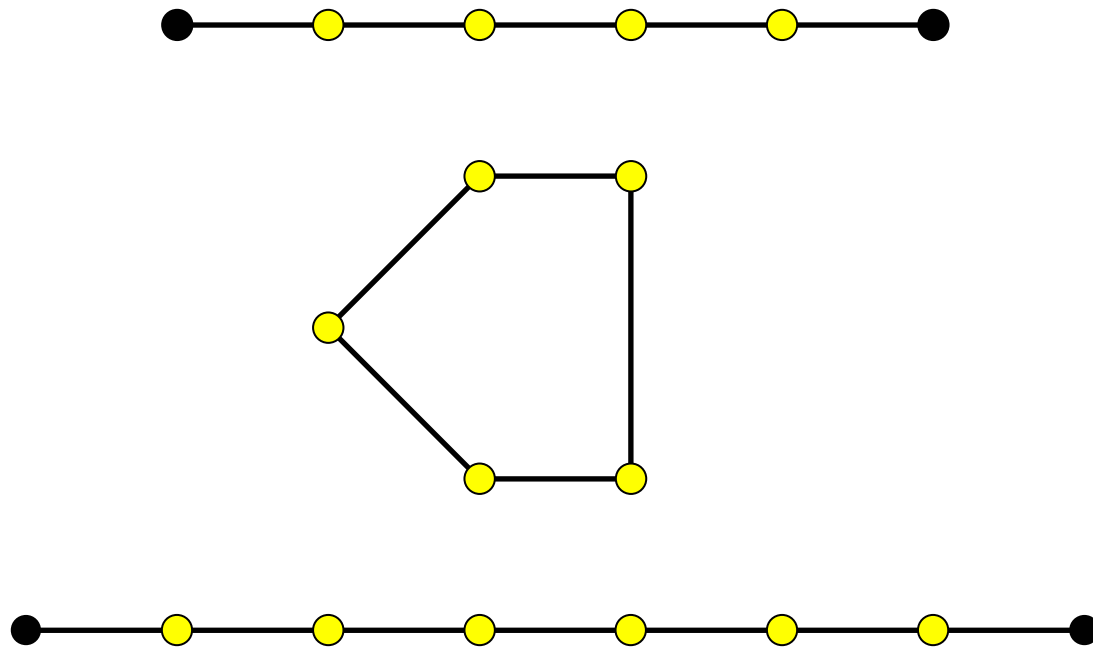
forbidden!

2	2
0	2
2	0

The 22 here must not be resolved 00/11.

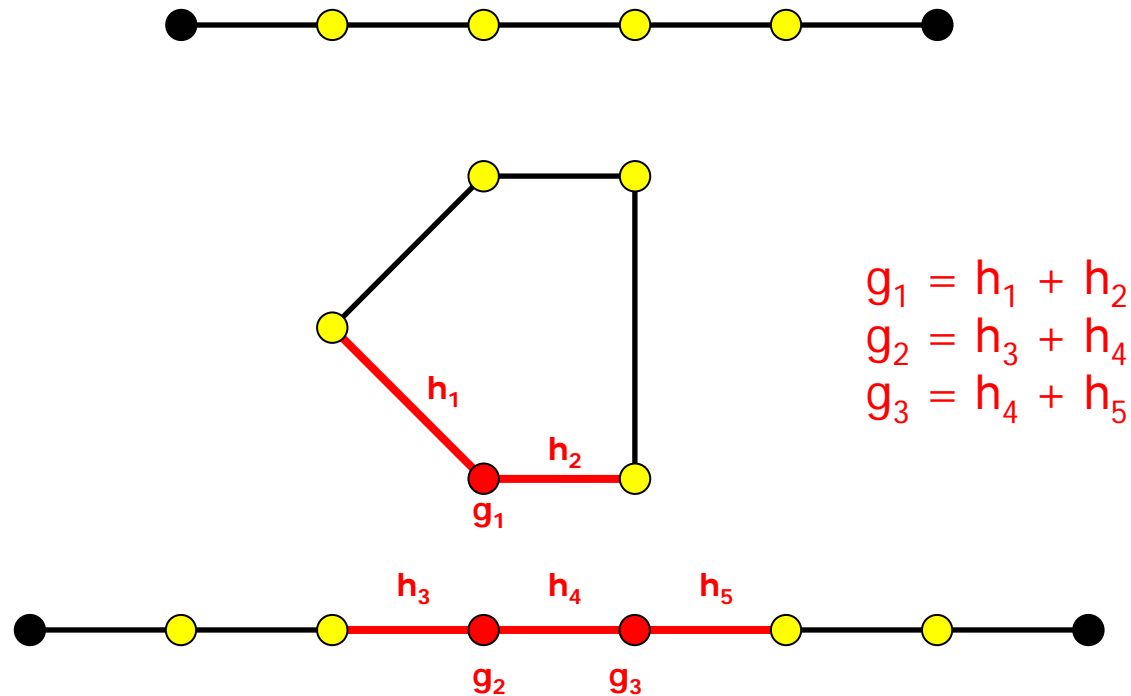


Example of restricted compatibility graph



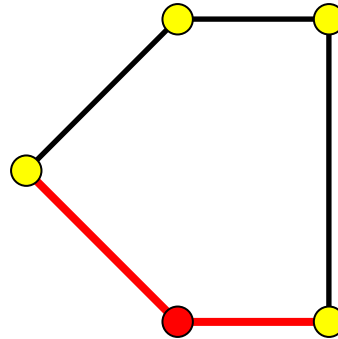
Vertices that in the beginning are degree-2 are coloured yellow.
These can never be resolved using the all-0 haplotype!

Step 1: Where degree-2 vertices can be resolved as the sum of their two adjacent edge haplotypes, do that.

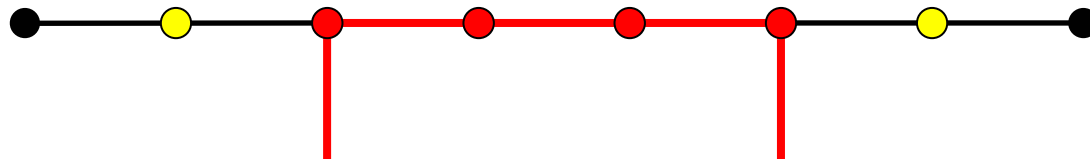
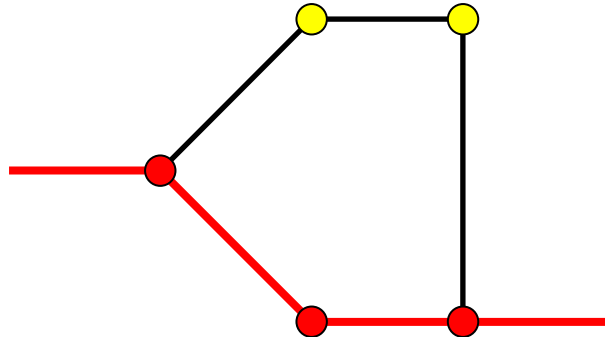


A red edge denotes that the corresponding edge haplotype has been put in the solution. Resolved genotypes are also shown in red.

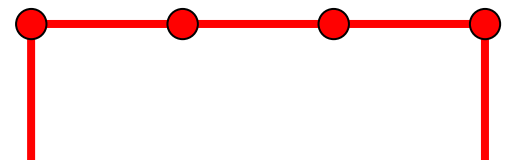
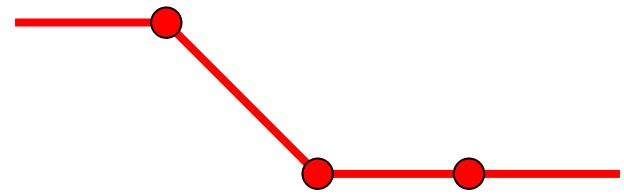
Step 2: Resolve genotypes that are adjacent to edge haplotypes added in the previous step (i.e. in this case where yellow vertices are next to red edges)



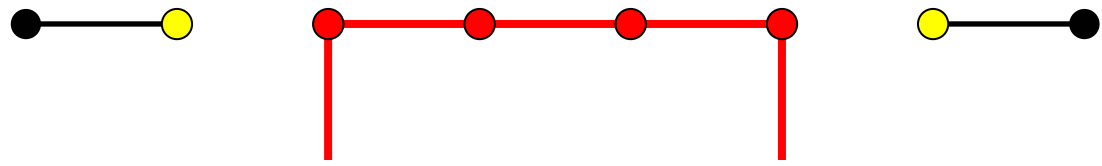
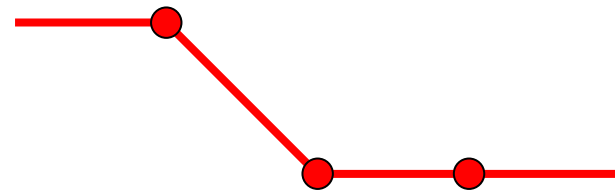
Step 2: Resolve genotypes that are adjacent to edge haplotypes added in the previous step (i.e. in this case where yellow vertices are next to red edges), done



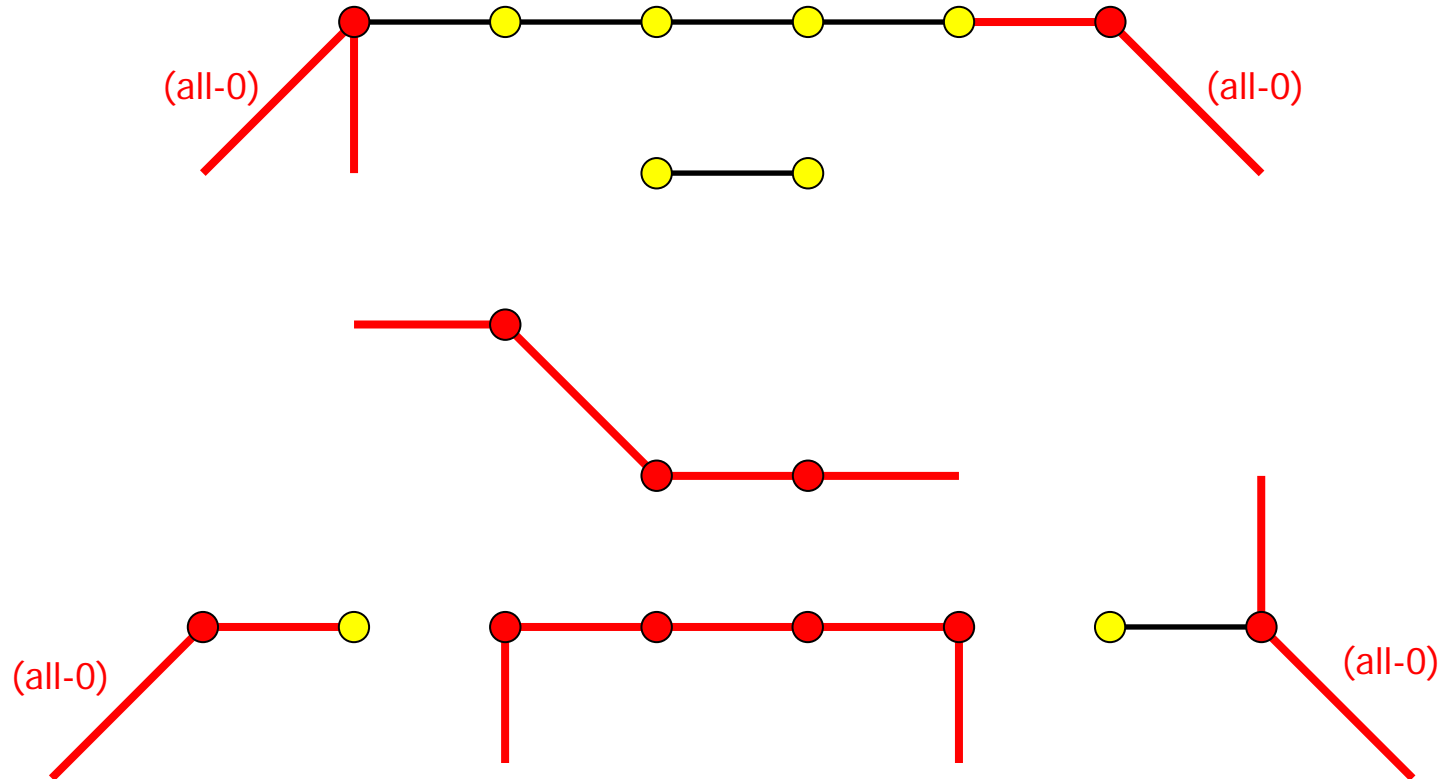
The new red edges, representing the newly added haplotypes, will be private haplotypes and thus cannot be shared by any other genotypes (symbolised here by the edge leaving the vertex at an angle.)

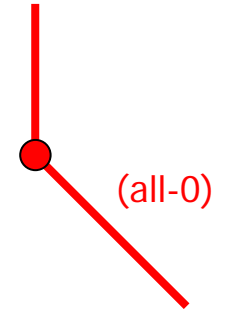
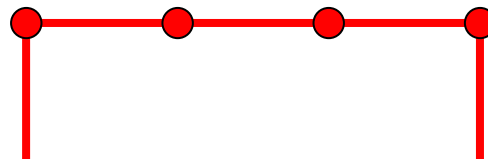
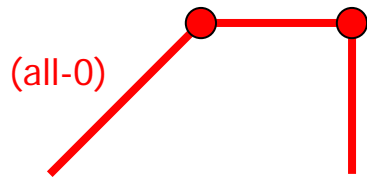
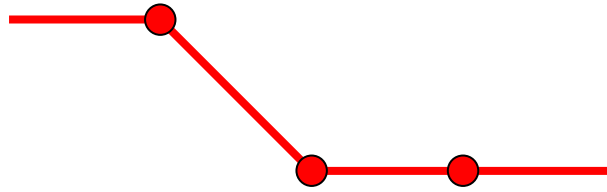
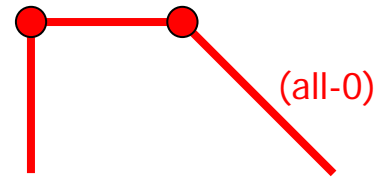
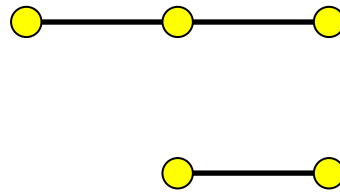
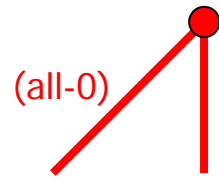


Step 3: Unless all components are 'bad', or no black genotypes left, add all-0 haplotype and use it to resolve remaining black genotypes



Step 3: Unless all components are 'bad', or no black genotypes left, add all-0 haplotype and use it to resolve remaining black genotypes...done!







Open problems...lots to do!

- Complexity of $\text{PH}(*, 2)$?
- Complexity of $\text{MPPH}(*, 2)$?
- Complexity of $\text{PH}(3, 2)$?
- Complexity of $\text{MPPH}(3, 2)$?
- Is $\text{MPPH}(j, k)$ always of the same complexity as $\text{PH}(j, k)$?
- In how far is PH reducible to MPPH, and vice-versa?
- Approximation algorithms for $\text{MPPH}(j, *)$ for fixed j
- Using the fact that k is fixed to improve the 2^{j-1} approximation algorithms for $\text{PH}(j, k)$
- How close are the solutions provided by the various PPH feasibility algorithms (i.e. “produce ANY solution”) to optimal MPPH solutions?
- How feasible is it to simply search through the (implicit) set of all PPH solutions?
- Using more biologically interesting restraints (e.g. galled networks, SNP block partitioning models)

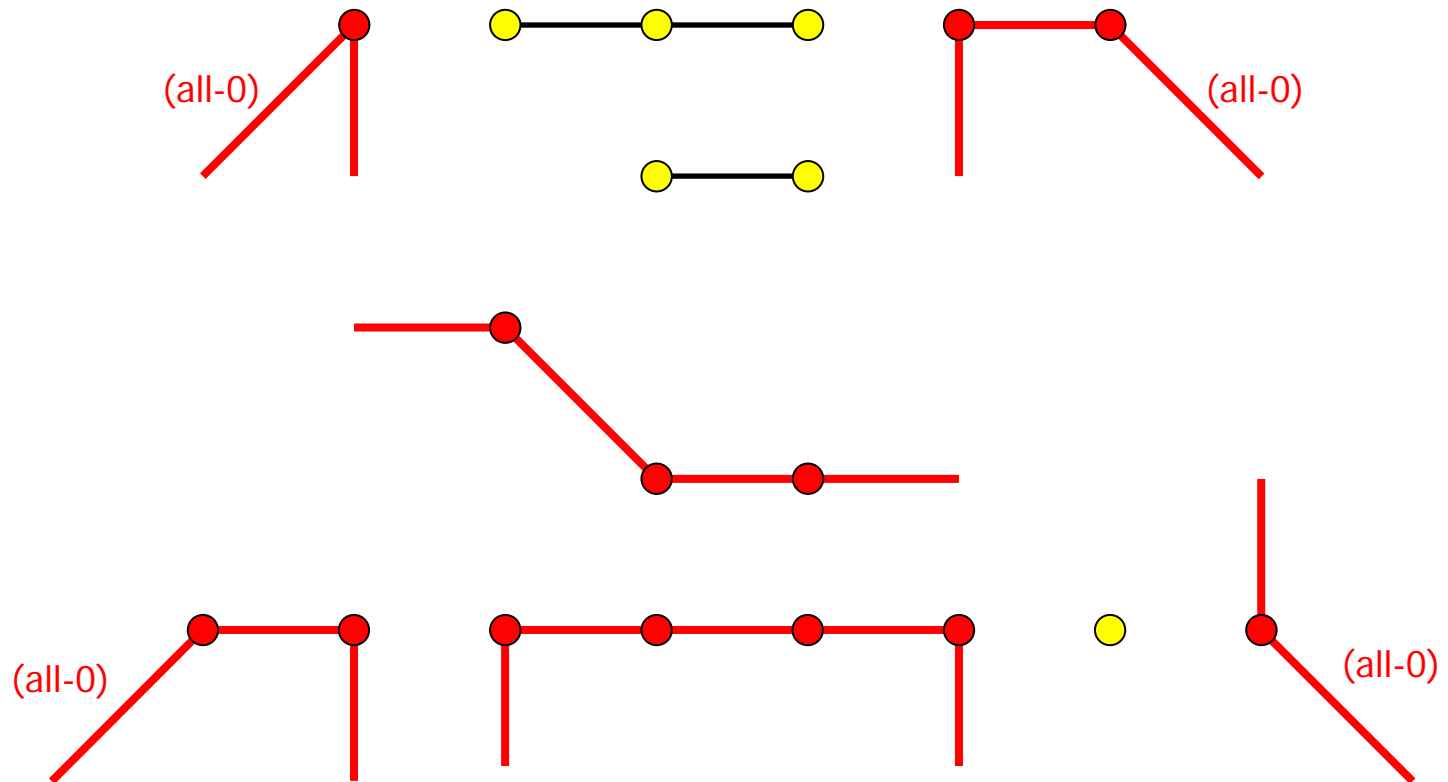


Open problems...lots to do!

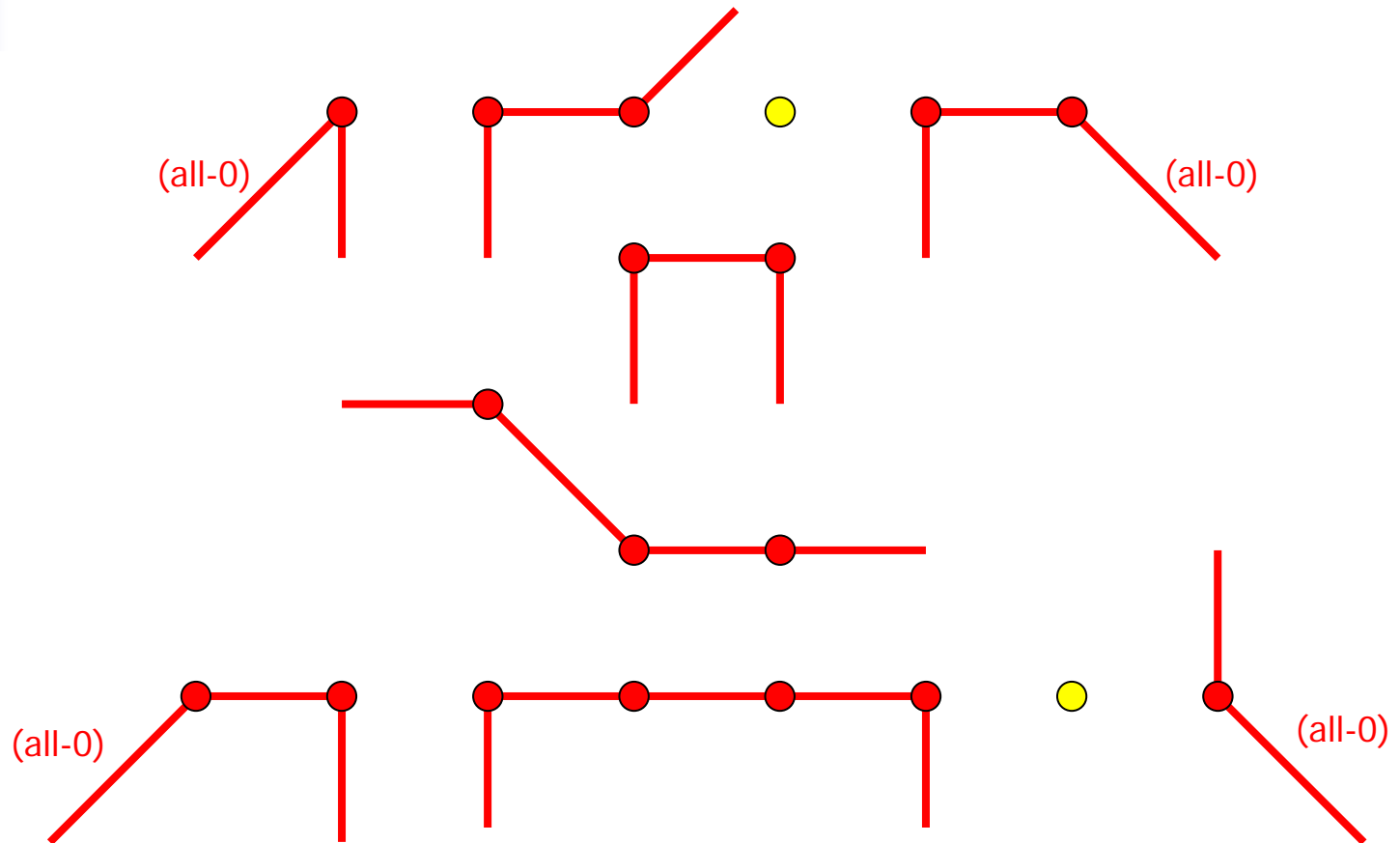
- Complexity of $\text{PH}(*, 2)$?
- Complexity of $\text{MPPH}(*, 2)$?
- Complexity of $\text{PH}(3, 2)$?
- Complexity of $\text{MPPH}(3, 2)$?
- Is $\text{MPPH}(j, k)$ always of the same complexity as $\text{PH}(j, k)$?
- In how far is PH reducible to MPPH, and vice-versa?
- Approximation algorithms for $\text{MPPH}(j, *)$ for fixed j
- Using the fact that k is fixed to improve the 2^{j-1} approximation algorithms for $\text{PH}(j, k)$
- How close are the solutions provided by the various PPH feasibility algorithms (i.e. “produce ANY solution”) to optimal MPPH solutions?
- How feasible is it to simply search through the (implicit) set of all PPH solutions?
- Using more biologically-interesting restraints (e.g. galled networks, SNP block partitioning models)

Thankyou for listening!

Steps 5: Take a maximum matching on the remaining yellow genotypes and add the corresponding edge haplotypes...



Step 5: Take a maximum matching on the remaining yellow genotypes and add the corresponding edge haplotypes...done.



Step 6: Finally, resolve each remaining isolated genotype using an arbitrary edge haplotype (plus one other haplotype.)

