# AlPha: A Mixed Integer Linear Programming Approach for Genome Haplotyping

Kerian Thuillier

# AlPha: A Mixed Integer Linear Programming Approach for Genome Haplotyping

Kerian Thuillier

19th August 2020

**Abstract**

**Background:** Nowadays, biological samples often contain DNA of several species or individuals. When sequenced, we loose the origin of each fragments of DNA and their position on the molecule. The problem of associating a DNA fragment to a version of the genome and to find its position is called the *haplotyping problem*.
**Results:** In this report, we present two new *Mixed Integer Linear Programming* (MILP) methods based on *multicommodity-flow* to solve this problem. Unlike previous approaches, the resolution avoid the use of heuristics by using global optimisation.
**Conclusion:** We do not currently known the quality of our results. However, proposed a comparison between the solving time complexity of our two MILP models. We also proposed an experimental protocol to test the solution quality.

**Keywords** – *metagenomic, genome assembly,* MILP*, multicommidity-flow*

# Contents

# Introduction

A sequencer is a machine able to read short fragments of DNA called *reads*. A read is a short sequence of characters representing nucleotides. As only short fragments of the genome are read by the sequencer, we loose the read position on the genome. Reads can also be paired, in this case we know that both reads come from the same molecule. Standard genome assembly consists in ordering all the reads to reconstruct the whole genome. However, such assembly reconstructs the genome by consensus. When an assembler has the choice between several reads to complete the assembled sequence, it adds the most-read read. Thus, standard genome assembly could not reconstruct more than one genome from a set of reads.

A sample contains often more than one individual or comes from a polyploid organism, an organism which has several versions of its genome. For example, human beings are diploids, they have two versions of the human genome: one version inherited from the father and the other inherited from the mother. The different versions of one genome are called haplotypes. During the sequencing process, all the haplotypes are sequenced together. Reconstructing all the haplotypes from a set of reads is called *haplotype-aware genome assembly*. Recently proposed algorithmic solutions solve this problem [3, 4]. However, these solutions are error-prone and dedicated to small bacterial genomes.

In this report, we propose two new approaches for haplotype-aware genome assembly based on *Mixed Integer Linear Programming* (MILP). The first one relies on *fact graph* (see section 3.1), while the second one relies on *alleles graph* (see section 4.1). These graphes come from the output of the *DiscoSNP* [12] algorithm. *DiscoSNP* is a *de novo*[1] variant detection tool. A variant is a single nucleotide difference between several genomes or haplotypes.

First, we will introduce both the *genome assembly problem* and the *haplotyping problem* (Section 1). Then, we will present our first approach relying on fact graph (Section 3) and its limits. Afterward, we will introduce our second approach based on allele graph (Section 4). The advantage and the limits of this approach are discussed in this section. Finally, we will present our result validation protocol (Section 5) and develop our work through a discussion (Section 6).

# 1 Haplotype-aware genome assembly

## 1.1 Genome assembly Problem

Modern sequencers allow reading a genome by slicing it in fragments. Those fragments are called *reads*. A read is a sequence of about 200 characters: A, C, G or T. Note that sequencer are error-prone, a read could contain errors. A sequencer output is a list of billions of reads. The output often does not contain any information about the read positions. Some modern sequencers can provide more information as the distance on the genome between some reads. An example of sequencer output is provided Figure 1.

*Genome assembly problem* is a scheduling problem. Each read is ordered to reconstruct the genome. Read position is determined by its neighbours. A read is positioned

[1] It does not use any reference genome to perform the assembly.

```
>read1
ACGACTTACGTCAGTCAGTCAGCAGCGAGTCATGACCGTGCGCAGTCAGCATCGATCAGC
>read2
CAGCGCGGACGCGCAGCATCTAGCGAGTAGGCGGCTTGTACGTGGGCGATACGATCGTGT
...
>read100,000,000
CGGCAGGCAGCGGCAGCGAGCAGCAGGAGAGGCGCGAGCGACGGACGTCGTATTTGTGGT
```

Figure 1: Example of sequencer output. It contains 100 millions of reads. The lines starting with '>' are comments provided by the sequencers, here it only contains the identifier of the read (the line below). The read is the sequence of 'A', 'C', 'G' and 'T'.

between two reads with which it overlaps. An overlap is a read's subsequence which is prefix for a read and suffix for another read. Thus, the *Genome assembly problem* could formally be defined as below.

---

```
GENOME ASSEMBLY PROBLEM
INPUT:
  R a set of reads
OUTPUT:
  S the longuest sequence of reads such that:
    1. each element of S is in R;
    2. each read of R is at most once in S;
    3. two successives reads of S are overlapping.
  The number of unused element of R is minimised.
```

---

The *genome assembly problem* is considered as an NP-complet problem due to the method used to solve it: hamiltonian path or longuest path [11].

Genome assemblers such as *SPAdes* [5] are commonly used to reconstruct genomes. However, these algorithms are not error-proof and have some restrictions. First, genomes contain a lot of repeats (nucleotide sequences repeated several times in the genome). The handling of repeats is an unresolved problem and an active research topic. Secondly, standard assemblers can only reconstruct one genome from a set of reads provided by one individual.

## 1.2 Haplotyping Problem

Humans are diploid, each chromosome is present twice. Those chromosomes are almost identical but have slight variations (about 1% of divergence). For humans, one version of these chromosomes was inherited from the mother while the second was inherited from the father. Chromosomes inherited from the mother form a haplotype as those inherited from the father. In the same way, a cup of sea water contains billions of micro-organisms. Each organism is a haplotype. When these elements are sequenced, reads from all haplotypes are merged. Thus, standard genome assembly strategies do not work. For example, if the human genome is sequenced and assembled with standard strategies, only one haplotype

will be produced. This haplotype will be a matchup of both real haplotypes. Even more absurd, if the water cup is sequenced and assembled, the reconstructed genome will be a chimera.

That's where the *haplotyping problem* takes place. Let $R$ be a set of reads obtained by sequencing $k$ haplotypes $\{h_1 ... h_k\}$. The number of haplotypes $k$ is unknown. The *haplotyping problem* consists in finding $k$, reallocating each element of $R$ to its associated haplotype and assembling them. In the end, each read $r \in R$ has a position in a haplotype or is ignored. Two versions of this problem exists: first one is the version described above, while second one has $k$ fixed or minimised.

---

```
HAPLOTYPING PROBLEM
INPUT:
  R a set of reads
  (k if k is fixed)
OUTPUT:
  k sequences of reads {S_1, ..., S_k} such that:
    1. for each sequence S_i, each element of S_i is in R;
    2. a read is at most in one sequence S_i;
    3. for each S_i, two successives reads of S_i are overlapping.
  The number of unused element of R is minimised.
```

---

The *haplotyping problem* is a well-known NP-complete problem. The *haplotyping problem* is the same as the *genome assembly problem* if there is only one haplotype. As this last problem is NP-complet, we deduce that *haplotyping problem* is too.

In recent papers, haplotype-aware assemblers have been proposed [3, 4]. However, these approaches are error-prone and can only process small bacterial genomes.

# 2 DiscoSNP: Variant Phasing

As different haplotypes provided by a single species are almost identical, it is more convenient to simplify the problem. Instead of working with reads, we could work with variants – *Single Nucleotide Polymorphism* (SNP). A SNP is a variation of only one nucleotide between two haplotypes. Working with SNP instead of reads is only removing information that is known to be shared between haplotypes. It allows to remove long non discriminative genomic regions.

## 2.1 DiscoSNP

*DiscoSNP* [12] is a reference-free variant phasing algorithm. From a set of reads, *DiscoSNP* computes the variants and phases them. The different versions of a variant are called *alleles*. For instance, the pair sequences ACCA**A**GTTT and ACCA**T**GTTT determine the variant A versus T (central position), while A is one allele of this variant and T the

other allele. Two alleles are phased if and only if there is a read overlapping both alleles. Phased alleles come from the same haplotype. A group of phased alleles is called a *fact*. Facts can be *paired*. Two facts are paired if they are overlapped by two paired reads (*i.e.* information provided by sequencers meaning that both reads come from the same haplotype). Figure 2 illustrates the phasing principle.

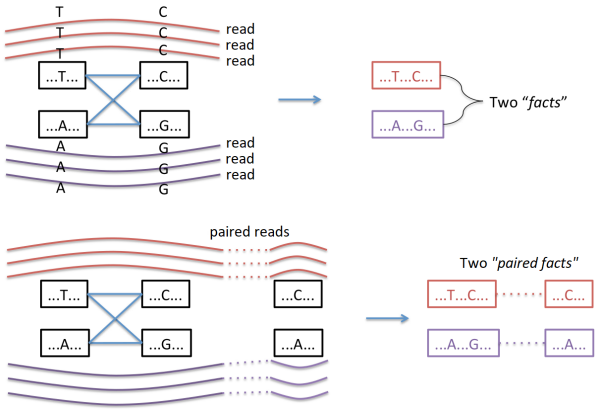



Figure 2: Illustration of the phasing problem: facts and paired facts construction. In the top figure, 'T'-'A' and 'C'-'G' are two variants such that 'T' and 'A' are the two alleles of the first variant, while C and G are the two alleles of the second variant. Red reads phase 'T' and 'C' into a fact. Purple reads phase 'A' and 'G' into another fact. In the bottom figure, there is a new variant 'C'-'A' (with 'C' and 'A' being its alleles). The dot edges between the reads represent paired reads. Red reads phase 'T' and 'C', and their paired reads phase 'C'. The facts formed by these paired reads are paired.

## 2.2 Outputs

*DiscoSNP* returns three files that contains the fact information:

- one text file containing all the raw facts and raw paired facts with their coverage, *i.e.* the number of reads mapping them. Each line of this file is a fact or a paired facts. A fact is described as below:

  ```
  1h_0;2l_-30; => 5
  ```

  Here, $1h$ is the allele $h$ of the variant 1 while $2l$ is the allele $l$ of the variant 2. The value after the underscore represent the distance of the allele to the previous one: the allele $2l$ start 30 nucleotide before the end of the allele $1h$. This fact has been seen by 5 reads.
  A paired fact is described as below:

  ```
  1h_0;2l_-30; 5h_0;6h_10;7l_-30; => 2
  ```

  This line represents two paired facts: $1h\_0; 2l\_ - 30;$ and $5h\_0; 6h\_10; 7l\_ - 30;$. Due to sequencing constraints, the second fact must be reversed to get the real paired facts. The real paired facts is thus $1h\_0; 2l\_ - 30;$ and $-7l\_0; -6h\_ - 30; -5h\_10$ (the reverse of $5h\_0; 6h\_10; 7l\_ - 30;$) which has been seen by 2 reads.

- two *fasta* files[2] containing alleles informations. These files contain the nucleotide sequences associated with each allele.

### 2.3 Post-process

Before being processed, *DiscoSNP* output is simplified. There are two phased of simplification. First one consists in filtering the uncoherent alleles. Second one consists in spliting the facts into *working zone*. A *working zone* is a cycle-free connected components where erroneous data were removed.

## 3 AlPha-Fact: MILP approach for genome haplotyping

Our first model is based on the interaction between the facts. These interactions are modelled as a *fact graph*. This model is implemented according to the Mixed Integer Linear Programming (MILP) paradigm. This paradigm relies upon declarative programming; problems are written in the form of linear objective function under linear constraints [7]. Used variables could be either integers or real values.

### 3.1 Fact Graph

From *DiscoSNP* output, we can build a *fact graph*. A *fact graph* is a graph modelling interactions between facts. There are four interactions between facts:

**Overlaps** There is an overlaps between two facts $f_1$ and $f_2$ iff the last phased alleles of $f_1$ are the same as the first phased alleles of $f_2$. For example, let $f_1 = 1h; 2l; 3h;$, $f_2 = 2l; 3h; 4l;$ and $f_3 = 2l; 3l; 4h$ then there is an ovelaps of size 2 between $f_1$ and $f_2$, and no overlaps between $f_1$ and $f_3$.

**Successive** Two facts $f_1$ and $f_2$ are successives iff the last variant phased by $f_1$ are known as preceding the first variant phased by $f_2$. For example, let $f_1 = 1h; 2l; 3h;$, $f_2 = 4h; 5l;$ and $f_3 = 3l; 4l;$ then there is no overlaps between those three facts. However, we know from $f_3$ that variant 3 comes before variant 4. Thus, we conclude that $f_1$ and $f_2$ are successives.

**Incompatible** Two facts $f_1$ and $f_2$ are incompatibles iff they phased different alleles of the same variant. For example, let $f_1 = 1h; 2l;$ and $f_2 = 1l; 2l;$ then $f_1$ and $f_2$ are incompatibles: $f_1$ phases $1h$ while $f_2$ phases $1l$ which are two alleles of variant 1.

**Paired** Two facts $f_1$ and $f_2$ are paired iff they have been seen as paired in the *DiscoSNP* output.

Let formally define a fact graph $G = (V, E, Ext, A, weight)$ where:

**V:** set of facts representing the graph vertices.

[2]The *fasta* format is used to stock nucleotide sequences

**E:** set of edges, there are four different types of edges corresponding to the four interactions between facts. First one is the overlaps edge $(f_1, f_2, overlaps)$ representing the overlap of two facts $f_1$ and $f_2$. Second one is the successive edge $(f_1, f_2, successive)$ modelling the successive relation between the facts $f_1$ and $f_2$. Third one is the incompatible edge $(f_1, f_2, incompatible)$. Last one is the linked edge $(f_1, f_2, links)$ meaning that $f_1$ and $f_2$ are paired. All these edges are directed except incompatible edges.

**Ext:** $Ext$ is a subset of vertices called extremities. Any $v$ in $Ext$ is such that either its set of predecessors, or its set of successors is an empty set (according to the overlaps and successive edges).

**A:** the set of alleles phased with each fact such that $A[v]$ is the set of alleles associated with the fact $v \in V$.

**weight:** weight associated with each overlaps and links edge. For an overlaps edge, $weight[u, v, overlaps]$ represents the overlap size, *i.e.* the number of overlaped variants. For a links edge, $weight[u, v, links]$ represents the abundance of the edge, *i.e.* the number of paired reads associated to the edge.

We will denote by $ab[al]$ the abundance of an allele $al \in AL$ where $AL$ is the set of all the alleles. The allele abundance $ab[al]$ is the number of reads containing $al$.

Note that some facts can be striclty included in others. For example, we can have both following facts in the graph: $f_1 = 1h; 2l; 3h; 4l;$ and $f_2 = 2l; 3h;$. These facts could not be removed since they can overlaps with other facts. Removing them could delete real path. Let $f_3 = 1l; 2l;$ and $f_4 = 3h; 4l; 5l;$. There are overlaps between $(f_3, f_2)$, $(f_2, f_4)$ and $(f_3, f_4)$. If $f_2$ is removed, the relation between $f_3$ and $f_4$ is definitively lost.

The figure 7 is an example of 4 haplotypes of 5 variants. Each variant has two alleles. A full example of fact graph over the haplotypes of figure 7 is available in figures 8 and 9.

## 3.2 MILP model

### 3.2.1 Multicommodity-flow based approach

Single genome assembly MILP models are based on the *longest path problem*. The longest path problem is a well-known NP-hard problem. This strategy consists in spreading a flow on an elementary path such that this flow is maximal [2]. According to this, we choose to base our approach on the *multicommodity-flow* principle [1]. *Multicommodity-flow* lets spread $k$ flows in a graph, where $k$ is the number of commodities and is fixed by the user. The $k$ flows are optimised simultaneously. Let $k$ be the number of haplotypes belonging to a sample and $G = (V, E, Ext, A, weight)$ be a fact graph. Let's denote by $\Phi^i_{al}$ a real variable representing the quantity of the haplotype $i$ passing through the allele $al$ (i.e. how many times the haplotype $i$ has been detected traversing the allele $al$). *Multicommodity-flow* principle defines that the sum of each commodity's flow is lower-bounded by the abundances of alleles [eq. 1].

$$\forall al \in AL, \quad ab[al] \leq \sum_{i=1}^{k} \Phi^i_{al} \tag{1}$$

To simplify the next constraints, we will give them from a commodity point of view.

### 3.2.2 Elementary path with abundance

In our context, a commodity can be seen as an elementary path representing a haplotype. The value of the commodity is the haplotype abundance. The haplotype abundance is the number of times this haplotype has been sequenced.

**Elementary path**

An elementary path is a path that does not pass through the same vertex twice. To model such path, we use GAT: a technique described in [9]. Let's define three binary variables $i_v$, $s_v$ and $t_v$ denoting the state of the node $v$ in the path, respectively intermediary node, source node and terminal node. GAT constraints are the following. A vertex could only have one state [eq. 2].

$$\forall v \in V, \quad s_v + i_v + t_v \leq 1 \tag{2}$$

There is only one starting vertex [eq. 3] and only one terminal vertex on a path [eq. 4].

$$\sum_{v \in Ext} s_v = 1 \tag{3}$$

$$\sum_{v \in Ext} t_v = 1 \tag{4}$$

A fact $v \in V$ could be a source or a target only if it belongs to $Ext$ [eq. 5].

$$\sum_{v \in V \setminus Ext} (t_v + s_v) = 0 \tag{5}$$

Now that vertex states are defined, we must add constraints to link them in an elementary path. Let's denote by $x_{(u,v)}$ a binary variable representing the activation of an edge $(u, v, overlaps) \in E$. An edge $(u, v, overlaps)$ is activated if it is used in the elementary path, *i.e.* $u$ is a source or intermediary vertex and $v$ is a terminal or intermediary vertex. Finding an elementary path can be perceived as spreading a flow worth 1. Thus, a law of conservation is necessary: a vertex has a single input iff it is terminal or intermediary [eq. 6], and a single output iff it is source or intermediary [eq. 7].

$$\forall v \in V, \quad \sum_{(u,v,overlaps) \in E} x_{(u,v)} = i_v + t_v \tag{6}$$

$$\forall v \in V, \quad \sum_{(v,w,overlaps) \in E} x_{(v,w)} = s_v + i_v \tag{7}$$

All these constraints define a path. To define an elementary path, we must remove loops. To remove loops, we use the *Miller-Tucker-Zemlin* (MTZ) constraint [8]. MTZ constraint consists in labelling each used vertex with a real number. Let's call $mtz_v$ this value where $v \in V$. The idea is that each vertex's value is strictly greater than its predecessor's value [eq. 8].

$$\forall (u, v, overlaps) \in E, \quad mtz_v - mtz_u \geq x_{(u,v)} - |V| \times (1 - x_{(u,v)}) \tag{8}$$

**Path abundance**

Each haplotype has an abundance. This abundance can be computed by spreading a flow on the elementary path representing the haplotype. The model must be completed with new constraints. Let $f_{(u,v)}$ be the flow passing through the edge $(u, v, overlaps) \in E$. Let's denote by $FF$ an upper bound for the flow. A flow passes through an edge $(u, v, overlaps) \in E$ iff the edge is activated ($x_{(u,v)} = 1$) [eq. 9].

$$\forall (u, v, overlaps) \in E, \quad x_{(u,v)} \leq f_{(u,v)} \leq x_{(u,v)} \times FF \tag{9}$$

According to the conservation law, for internal vertices the quantity of the input flow equals the quantity of the exit flow [eq. 10–11]. No constraint is applied for source and terminal vertices, they are not subject to the conservation law (no inputs or outputs).

$$\forall v \in V, \quad \sum_{(u,v,overlaps)\in E} f_{(u,v)} - FF \times (1 - i_v) \leq \sum_{(v,w,overlaps)\in E} f_{(v,w)} \tag{10}$$

$$\forall v \in V, \quad \sum_{(v,w,overlaps)\in E} f_{(v,w)} \leq \sum_{(u,v,overlaps)\in E} f_{(u,v)} + FF \times (1 - i_v) \tag{11}$$

Thus, we can formulate the abundance of the path. Let $f$ be a real variable representing the abundance of the path. The path abundance is equal to the outgoing flow of the path's source [eq. 12].

$$\forall v \in V, \quad \sum_{(v,w,overlaps)\in E} f_{(v,w)} - FF \times (1 - s_v) \leq f \leq \sum_{(v,w,overlaps)\in E} f_{(v,w)} + FF \times (1 - s_v) \tag{12}$$

### 3.2.3 Haplotype specifications

Sequencers and *DiscoSNP* can provide us complementary information about the haplotypes. This information is useful and needs to be used to prevent the creation of chimeras.

**Incompatible edges**

Firstly, an incompatible edge $(u, v, incompatible) \in E$ represents an incompatible combination of facts: $u$ and $v$ do not belong to the same haplotype [eq. 13].

$$\forall (u, v, incompatible) \in E, \quad (s_u + i_u + t_u) + (s_v + i_v + t_v) \leq 1 \tag{13}$$

**Links edges**

Secondly, a link edge $(u, v, links) \in E$ is a data provided by sequencers. Alleles phased by the paired facts have been seen on the same molecule. That means that $u$ and $v$ belong to the same haplotype. As data provided by sequencers is error-prone, we could not force these links to be satisfied. Let's denote by $l_{(u,v)}$ a binary variable representing the links validation. A link edge is satisfied iff $u$ and $v$ belong to the path [eq. 14] and $u$ is an ancestor of $v$ [eq. 15].

$$\forall (u, v, links) \in E, \quad 2 \times l_{(u,v)} \leq (s_u + i_u + t_u) + (s_v + i_v + t_v) \tag{14}$$

$$\forall (u, v, links) \in E, \quad l_{(u,v)} - (1 - l_{(u,v)}) \times |V| \leq mtz_v - mtz_u \tag{15}$$

### 3.2.4 Variants

Each fact phases a set of alleles. The use of alleles' data allows to specify the haplotype. Let $a^i_{al}$ be a variable such that $a^i_{al} = 1$ iff the haplotype $i$ passes through the allele $al$, otherwise $a^i_{al} = 0$ [eq. 16–17].

$$\forall i \in [1;k], \forall v \in V, \forall al \in A[v], \quad (s^i_v + i^i_v + t^i_v) \leq a^i_{al} \tag{16}$$

$$\forall i \in [1;k], \forall al \in AL, \quad a^i_{al} \leq \sum_{\substack{v \in V \\ al \in A[v]}} (s^i_v + i^i_v + t^i_v) \tag{17}$$

Moreover, all the alleles must be used by at least one haplotype [eq. 18].

$$\forall al \in Al, \quad 1 \leq \sum_{i \in [1;k]} a^i_{al} \tag{18}$$

Let define $\Phi^i_{al}$ formulated in section 3.2.1. It is a real variable representing the flow passing through the allele $al \in AL$ in the haplotype $i \in [1;k]$. The flow passing through an allele $al$ is equal to the flow of the path $i$ if this path passes through $al$ [eq. 19–20].

$$\forall i \in [1;k], \forall al \in AL, \quad f^i - FF \times (1 - a^i_{al}) \leq \Phi^i_{al} \leq f^i + FF \times (1 - a^i_{al}) \tag{19}$$

$$\forall i \in [1;k], \forall al \in AL, \quad -FF \times a^i_{al} \leq \Phi^i_{al} \leq FF \times a^i_{al} \tag{20}$$

Note that $f^i$ is the abundance of the $i^{th}$ path. This real variable is the extended version of $f$ (see section 3.2.2) for the different commodities.

### 3.2.5 Objective Function

DNA sequencers are error-prone: we are almost sure of what we have seen, but it is possible to have missed some information. Thus, all the haplotype could not be of the same length. The length of the haplotypes should be as close as possible to each other. Let us introduce the real variable $\delta^l$ modelling this difference. This variable is defined according to the following constraints [eq. 21].

$$\forall i_1 \in [1;k], i_2 \in [1;k], \quad \sum_{al \in AL} (a^{i_1}_{al} - a^{i_2}_{al}) \leq \delta^l \tag{21}$$

The allele abundance is the number of times that we are sure to have seen the allele. It is possible to have missed some allele occurrences during the sequencing process. Thus, the abundance of haplotypes should be as close as possible to the abundance of alleles. Let $\delta^v_{al}$ be the difference between the allele's abundance and the flow passing through the allele on each haplotype, where $al \in AL$. This variable is defined by the constraints below [eq. 22].

$$\forall al \in AL, \quad (\sum_{i \in [1;k]} \Phi^i_{al}) - ab[al] \leq \delta^v_{al} \tag{22}$$

Note that according to equation 1, $\delta^v_{al} \geq 0$.

The objective function of our model is a multi-objective one. It must minimise the difference between alleles abundance and paths flow, and the difference between the length

of each haplotype. It must also maximise the number of validated links edges, since it is a powerful information given by the sequencer (see section 3.2.3). To model this function, we must introduce three real variables fixed by the user: $\alpha_1$, $\alpha_2$, $\alpha_3$ with $\alpha_i \geq 0$. These variables are coefficients used to weight the three objectives. Our objective function $H_k$ for a given $k$ is as below [eq. 23].

$$\begin{aligned} \max \quad H_k = \; & \alpha_1 \times \sum_{\substack{1 \leq i \leq k \\ (u,v,links) \in E}} l^i_{(u,v)} \times weight_{(u,v)} \\ & - \alpha_2 \times \sum_{al \in AL} \delta^v_{al} \\ & - \alpha_3 \times \delta^l \end{aligned} \tag{23}$$

The solution of the fact graph presented in figure 8 is available in figure 9. The solution satisfied all the constraints presented in this report and the three links edges. It is the solution find by the model.

## 3.3 Results

The model was tested on a simulated dataset. This dataset was obtained by simulating mutations on SARS-COV-2 genome (Wuhan strain). This data set is composed of 3 761 facts, 13 847 overlaps and successives, and 36 385 links (see figure 14).

Despite the lack of results showing the assembly quality, the model shows a scalability problem. Small instances of about 100s facts take more than 3 hours to be solved, while we have not been able to process bigger instances in less than 6 hours. For example, we have not been able to process the connected component shown in figure 15.

# 4 AlPha-Allele: MILP approach for genome haplotyping

To solve the scalability problem of previous approach, we proposed an other model relying on allele graph instead of fact graph. The idea behind this model is that allele graphs are smaller and more regular than fact graphs.

The model implementation is still made according to the MILP paradigm.

## 4.1 Allele Graph

From *DiscoSNP* output, we can build an *allele graph*. An *allele graph* is a graph modelling the alleles: each vertex models an allele and each edge models the allele order.

Let formally define an allele graph $G = (V, A, E, F, P)$ where:

**V:** set of variants, it represents the graph columns.

**A:** set of alleles $(v, a) \in A$, such that $v \in V$ is a variant and $a \in \{h, l, -\}$ is one allele of variant $v$. The abundance of the allele $(v, a)$ will be denoted by $ab((v, a))$.

**E:** set of edges, $((v_1, a_1), (v_2, a_2)) \in E$ with $(v_1, a_1)$ and $(v_2, a_2)$ two alleles. There is a link between two alleles iff there is at least one fact containing both alleles.

**F:** set of facts, a fact $f$ is an ordered sequence of alleles. It models the raw facts returned by *DiscoSNP*. It is a set of tuple, such that $(n, (v, al)) \in F$ where $n$ is the fact id and $(v, al) \in A$ an allele.

**P:** set of paired facts, $(f_1, f_2) \in P$ such that $f_1$ and $f_2$ are two facts. It is a set of couple of facts id.

Figure 10 is an example of allele graph. Each vertex is an allele, each graph column is a variant. This graph is constructed with the same input (figure 7) as the fact graph in figure 8. Figure 11 show the original haplotypes on this allele graph.

### 4.1.1 Properties

Unlike fact graphs, allele graphs have few interesting properties:

1. allele graphs are *directed acyclic graph*. It is impossible to pass through the same allele twice;
2. allele graphs are *n-partite graphs*. It is possible to split the vertex in $n = |V|$ parts, such that two vertices of the same parts are not linked. Thus, no elementary path can pass through the same variant twice.
3. allele graphs are almost regular: almost each variant has at most one successor and at most one predecessor.

### 4.1.2 Linearisation

As previously explained, allele graphs are almost regular. There are few irregularities in the graphs. There are three types of irregularities which are illustrated in figure 3:

**Bubbles:** a variant has at least two variant successors. It create two concurrent branches in the graph (figure 3-a).

**Shortcuts:** a variant has at least two variant successors, one of these successors is a descendant of the others variant successors (figure 3-b).

**Deadends:** one allele of a variant does not have any successors while the others alleles have some (figure 3-c).

Let denote by *branches* two sequences of variants such that they have the same predecessor and successor, and that there are no edge between element of the two branches.

In order to remove these unwanted structures, we developed a linearisation method. It consists in ordering the variants such that:

- Let $v_i$ be a variant and $v_{i+1}$ one of its successors, thus $\mathrm{id}(v_i) < \mathrm{id}(v_{i+1})$.
- Let $b_1$ and $b_2$ be two concurrent branches, $u$ the variant preceding the two branches and $v$ the variant succeding the two branches. Thus, we must have $\forall w \in b_1, \mathrm{id}(u) < \mathrm{id}(w) < \mathrm{id}(v)$ and $\forall w \in b_2, \mathrm{id}(u) < \mathrm{id}(w) < \mathrm{id}(v)$.

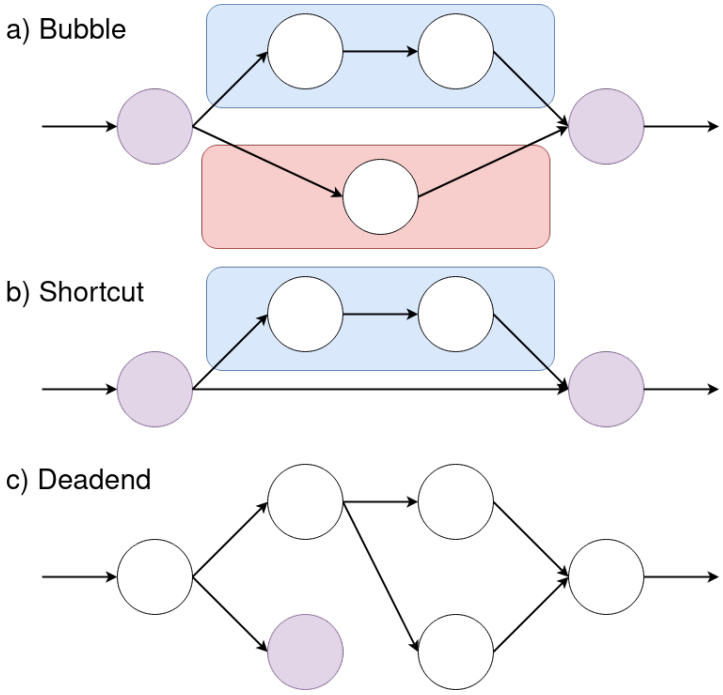



Figure 3: **a)** Bubble example. Purple nodes are the bubble extremities. The left one has two variant successors which create two branches (blue one and red one). These branches merge into the right purple node.
**b)** Shortcut example. Left purple node has two variant successors. One of these successors is the right purple node which is also a successor of the blue branch.
**c)** Deadend example. Purple node does not have any successors while others nodes of the column have some.

- Two distinct variants $u$, $v$ could not have the same id, $\mathrm{id}(u) \neq \mathrm{id}(v)$.

It is impossible to use classical ordering algorithm. Such algorithms could not properly order the variants of concurrent branches. To order these branches, let introduce the notion of *dummy alleles*. A *dummy allele* is a fake allele used to model skipped variant. To order the variant, we apply the following algorithm:

1. Finding the smallest concurrent branches $b_i$ and $b_j$. It can be considered as finding the smallest cycle basis of the graph.

2. Ordering the two branches. We choose to order the branches according to the name of the first element of each branch. Let denote by $b_1$ and $b_2$ both branches such that $b_1$ is the fist one.

3. Retrieving the $u$ the branches predecessor and $v$ the branches successor.

4. Removing all the edges between $u$ and the first element of $b_2$, and the last element of $b_1$ and $v$.

5. Adding a dummy allele to each variant of $b_1$ and $b_2$.

6. Adding edge between the dummy alleles of two consecutives variants of each branch. Adding edges between $u$ and the first dummy alleles of $b_1$, the last dummy allele of $b_1$ and all the first non-dummy alleles of $b_2$, all the last non-dummy alleles of $b_1$ and the first dummy allele of $b_2$, and the last dummy allele of $b_2$ and $v$.

An example of this algorithm applied to a bubble can be seen figure 4. This algorithm was applied on the allele graph shown in figure 10, the result is shown in figure 12. The graph with the original haplotype (*i.e.* the solution) is shown in figure 13.

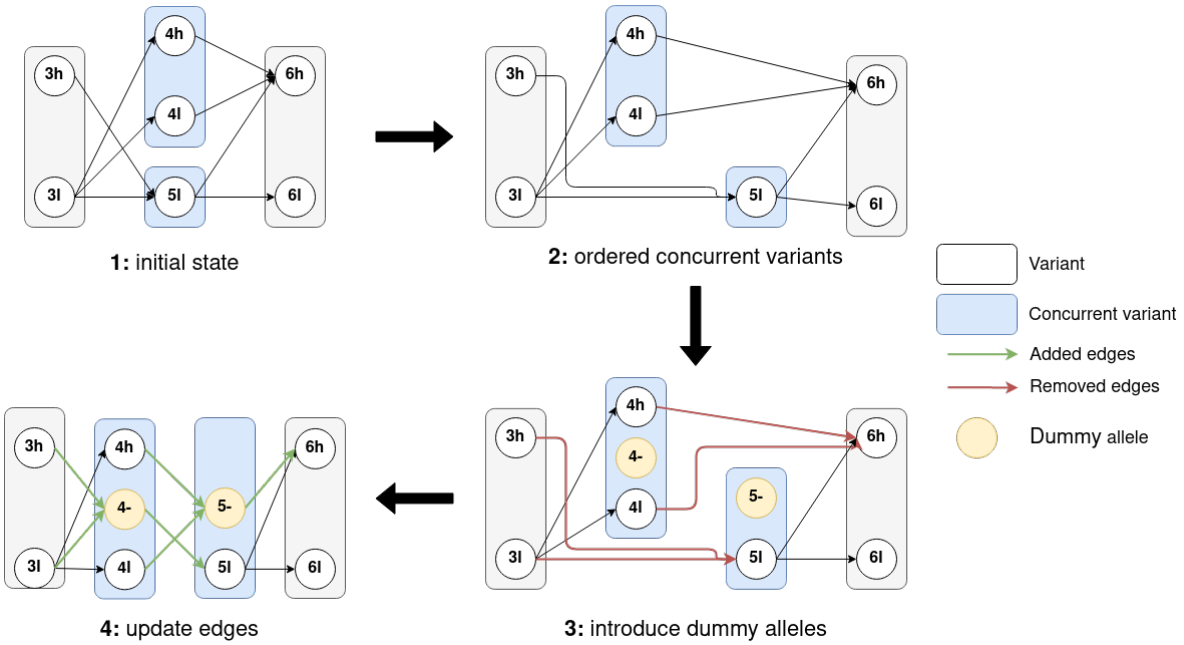



Figure 4: Example of linearisation on a bubble. The two blue blocks are the concurrent branches Each block is a variant and each circle is an allele. Yellow circles are dummy alleles.

## 4.2 MILP Model

Like MILP model explained in section 3.2, this new model is based on multicommidity-commodity flow.

Thereafter, we will denote by $G = (V, A, E, F, P)$ the linearised allele graph and by $k$ the number of haplotypes. The set of linear constraints is divided in 4 categories: haplotype constraints, flow constraints, fact constraints and the objective function.

### 4.2.1 Haplotype Constraints

Firstly, let's define the constraints defining haplotypes. Let $a^i_{(v,al)}$ be a binary variable equaled to 1 iff the allele $al$ of variant $v$ belongs to the $i^{\text{th}}$ path. As the allele graph is linear, each haplotype must pass through each variant exactly once [eq. 24].

$$\forall i \in [1; k],\ \forall v \in V,\quad \sum_{(v,al)\in A} a^i_{(v,al)} = 1 \tag{24}$$

Let $last(V)$ be the last variant of the graph, the only variant without any successor, and $next(v)$ for $v \in V$ the successor of variant $v$. Two activated alleles of successive variants are linked by an edge in the allele graph [eq. 25].

$$\forall i \in [1;k],\ \forall v \in V \setminus \{\ last(V)\},\ \forall (v,al) \in A,\quad a^i_{(v,al)} \leq \sum_{\substack{w=next(v) \\ ((v,al),(w,bl))\in E}} a^i_{(w,bl)} \tag{25}$$

### 4.2.2 Flow Constraints

Let's now define the abundance of an haplotype, *i.e.* the flow passing through the commodities. Let $f^i_{(v,al)}$ be a real variable modelling the flow of the allele $al$ of variant $v$ on the $i^{\text{th}}$ commodity. Let $FF$ be an upperbound of the flow. The flow on inactivated alleles is null [eq. 26].

$$\forall i \in [1;k],\ \forall (v,al) \in A,\quad a^i_{(v,al)} \leq f^i_{(v,al)} \leq FF \times a^i_{(v,al)} \tag{26}$$

The flow on a commodity is constant between two successive variants [eq. 27].

$$\forall i \in [1;k],\ \forall v \in V \setminus \{last(V)\},\quad \sum_{(v,al)\in A} f^i_{(v,al)} = \sum_{\substack{w=next(v) \\ (w,bl)}} f^i_{(w,bl)} \tag{27}$$

### 4.2.3 Fact Constraints

A fact, *i.e.* a tuple of alleles, is satisfied by an haplotype iff all the alleles of the fact belongs to the haplotype. Let $q^i_n$ be a binary variable modelling the satisfaction of fact $n$. Fact satisfaction can be modelled by the following linear equation [eq. 28].

$$\forall i \in [1;k],\ \forall (n,(v,al)) \in F,\quad q^i_n \leq a^i_{(v,al)} \tag{28}$$

Thus, let's denote by $q_n$ a binary variable equalled to 1 iff the fact $n$ is globally satisfied. A fact is said *globally satisfied* iff at least one haplotype satisfied it [eq. 29].

$$\forall (n,\_) \in F,\quad q_n \leq \sum_{i\in[1;k]} q^i_n \tag{29}$$

A paired fact $(n_1, n_2) \in P$ is satisfied by an haplotype $i$ iff $n_1$ and $n_2$ are satisfied by $i$. Let $p^i_{(n,m)}$ be a binary variable modelling the satisfaction of the paired fact $(n,m)$. Paired facts satisfaction can be defined as below [eq. 30].

$$\begin{aligned} \forall i \in [1;k],\ \forall (n,m) \in P,\ & p^i_{(n,m)} \leq q^i_n \\ & p^i_{(n,m)} \leq q^i_m \end{aligned} \tag{30}$$

Thus, let's denote by $q_{(n,m)}$ a binary variable equalled to 1 iff the paired fact $(n,m)$ is globally satisfied. A paired fact is said *globally satisfied* iff at least one haplotype satisfied it [eq. 31].

$$\forall (n,m) \in P,\quad p_{(n,m)} \leq \sum_{i\in[1;k]} p^i_{(n,m)} \tag{31}$$

### 4.2.4 Objective Function

Multi-objective function are harder to solve and to set up. To stay with a mono-objective function, we choose to force the number of satisfied facts and paired facts. Let $\alpha \in [0, 1]$ be the percentage of facts to satisfied and $\beta \in [0, 1]$ the percentage of paired facts to satisfied. Following equations force a certain percentage of facts [eq. 32] and paired facts [eq. 33] satisfaction.

$$\alpha \times |F| \leq \alpha \times \sum_{n \in F} q_n^i \tag{32}$$

$$\beta \times |P| \leq \beta \times \sum_{(n,m) \in P} p_{(n,m)}^i \tag{33}$$

Thus, the only objective is to minimise the gap between the sum of haplotype abundances on each allele and the real allele abundances. Let $A_r$ be the set of real alleles, *i.e.* $A_r = A \setminus \{(v, al) \in A \mid (v, a) \text{ is a dummy allele}\}$. Let define $\epsilon$ a real value modelling the biggest gap. This variable can defined thanks to following equation [eq. 34].

$$\forall (v, al) \in A_r, \quad -\epsilon \leq \sum_{i \in [1;k]} f_{(v,al)}^i - ab((v, al)) \leq \epsilon \tag{34}$$

The objective function can be defined as:

$$\text{minimise} \quad \epsilon$$

## 4.3 Results

This model was tested on the same dataset as previously (see section 3.3). Despite the lack of results showing the assembly quality, we see an improvement of computation time.

Let's see the impact of linearisation on computation time. The computation time for a normal graph and a linearised graph is shown figure 5. We can see that the linearisation process drastically improve the computation time. The number of variants has an impact on the computation time as we can see it in figure 6. This figure compares computation times of linearised and normal graphs according to the number of variants. It shows that normal graph with less than 50 variants can be processed in less than 10 800 seconds. In contrast, linearised graph with 200 variants can be processed in less than 10 800 seconds.

# 5 Results validation

At this stage, we do not have any idea on the model assembly quality. The model is still under development but can already prefectly solve toy examples like the one shown in figure 8. Real instances are more complex than the toy example shown in figure 8. An example of a full Covid-19 instance is shown figure 14 with a zoom on one of its connected components in figure 15. This instance is about 500 times bigger than our toy example. The problem is solved separately on each connected component. Thus, we reduce the problem to instances about 20 times bigger than the toy example.

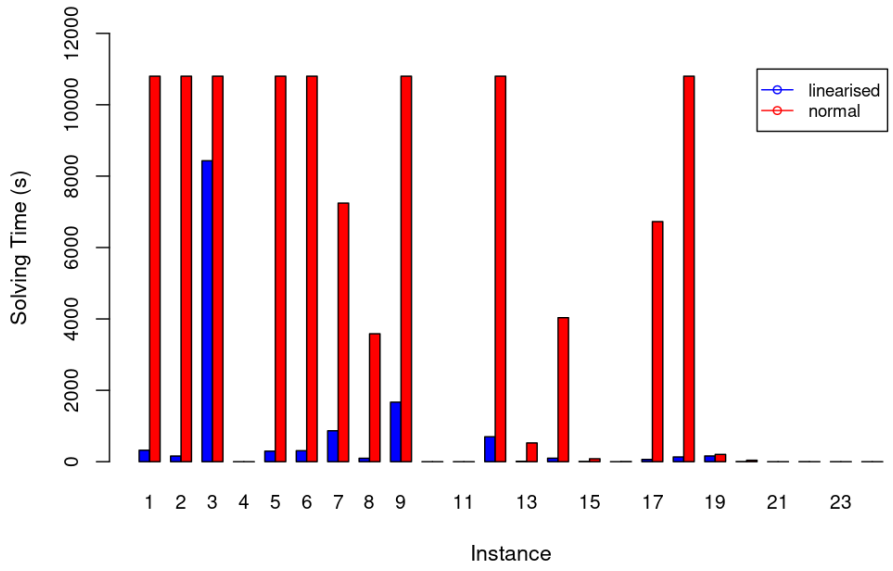



Figure 5: Solving times in seconds for each instance. Blue stack is the computation time of the linearised instance. Red stack is the computation time of the normal instance.

To validate the model results on real data sets, we have established an experimental protocol. The objective of this project is to use the reconstructed haplotypes in population genomic. In particular, model results will be used to compute the *nucleotide diversity*. Nucleotide diversity is a metric used to measure the degree of polymorphism within a population (*i.e.* the population diversity). Thus, we can validate our results by comparing the nucleotide diversity of our solution with the real nucleotide diversity. Note that this kind of validation could be used only when the original haplotypes are known. We formulate the following results validation protocol.

1. We take the genome of $\alpha$ species. Let $[S_1 \dots S_\alpha]$ be these genomes. From each genome $S_i$, we compute $\beta$ haplotypes $[h_{i,1} \dots h_{i,\beta}]$ by mutating the original genome according to a mutation rate $\nu$.

2. Let $H = \{h_{i,j} \,|\, 1 \leq i \leq \alpha,\ 1 \leq j \leq \beta\}$ be the set of all the haplotypes generated for each species. We sample with replacement $n$ haplotypes from $H$. This last step is repeated $\gamma$ times. Let $[I_1 \dots I_\gamma]$ be these samples.

3. The original nucleotide diversity value is computed for each sample. Let $\pi_j^i$ be the real nucleotide diversity for the $i^{\text{th}}$ sample and the $j^{\text{th}}$ species.

4. From each sample $I_i$, we simulate illumina reads with *SimSeq* [6]. Abundances are generated for each haplotype during the simulation process.

5. Simulated reads are filtered by species thanks to *MetavaR* [10]. *MetavaR* is a *de novo* algorithm used to cluster reads by species. For each reads, we have a probability of belonging to a cluster. Each cluster is then sent to *DiscoSNP* [12]

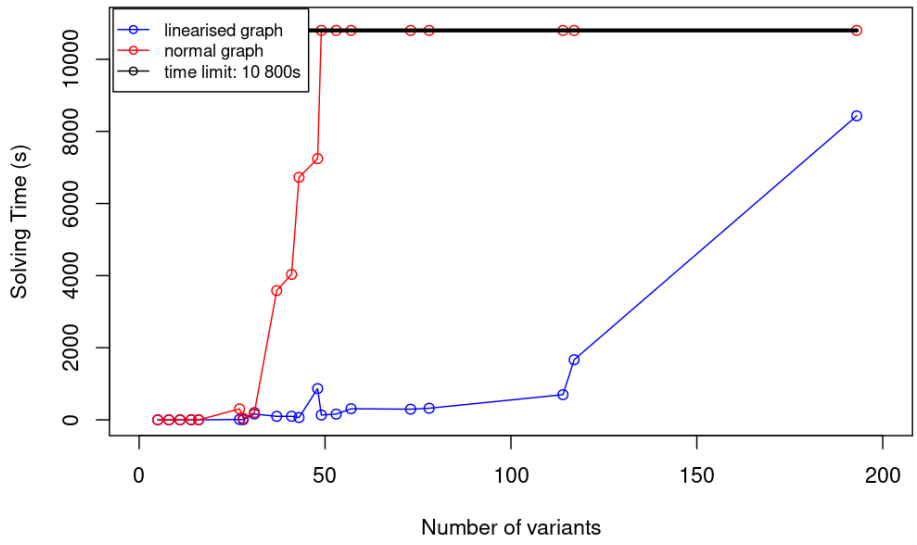



Figure 6: Solving time in seconds according to the number of variants. Blue line is the solving time for linearised graph. Red line is the solving time for the normal graph. Black line is the time limits, *i.e.* the process is kill after 10 800 seconds. The time limit is reached for normal graph with at least 50 variants, while it is not reached for linearised graph of 200 variants.

to create fact graphs. Notice that *DiscoSNP* filters reads to reduce the number of erroneous facts.

6. Finally, we use our model to solve each instance. The nucleotide diversity $\tilde{\pi}_j^i$ is computed for each species and compared to the original one $\pi_j^i$.

Another solution would be to compare the reconstructed haplotypes to the original ones. However, it is more complicated to have perfectly reconstructed haplotypes. We do not know yet if the reconstructed haplotypes have variant permutations (i.e. the haplotypes are correct within a few variant permutations). These permutations should not impact nucleotide diversity. Thus, we can first study our results with respect to nucleotide diversity before focusing on the quality of sequence reconstruction.

# 6 Discussion

In this report, we propose a *de novo* approach for haplotype-aware genome assembly. In contrast to similar approaches [3, 4], our model aims to reconstruct all haplotypes thanks to global optimisation. We do not use heuristics to reconstruct haplotypes from the input data. Currently, the number $k$ of haplotypes to reconstruct is fixed by users. We aim to automate this according to the following heuristic: $k$ is the minimum number of haplotypes needed to solve the problem. If $k$ is too small, the model will be infeasible. Thus, finding the minimum $k$ could be done by first trying to solve the model with $k = 1$.

We then increase $k$ until the model is feasible. Finding that a model is infeasible is really quick and is usually determined during the presolve process. The presolve process is a process during which the problem is simplified before being sent to the solver.

The approach introduced in this report is based on the multi-commodity flow strategy. Almost each model's variable is multiplied $k$ times (where $k$ is the number of commodities). Thus, the computation time will increase as a function of $k$. In order to reduce the impact of $k$ on the model, we have thought about another (not implemented) approach. This approach relies on extracting a set of paths from the linearised allele graph. Let's see how to compute this set of paths:
Remember that each fact and paired fact has a weight.

1. Extract one of the paths that satisfies the most facts and paired facts.
2. Reduce the weight of each satisfied fact and paired fact by one.
3. Remove all the facts and paired facts with a null weight.
4. Restart at step 1 until all facts have been removed.

Thus, each extracted paths is a candidate for our assembled haplotypes. In order to filter these paths and select the assembled haplotypes, we proposed to solve a simple linear problem: finding coefficients for each path such that they minimise the difference between haplotypes weights and alleles weights.

# Conclusion

In this report, we introduce two MILP approaches for the *Haplotyping problem*. From what we know, we present the first mathematical formulations of the *Haplotyping problem*. Unlike already existing approaches [3, 4], ours is not based on heuristics and solve the problem with global optimisation. To realise it, we start by presenting the biological context: genome assembly problem (Section 1.1) and haplotyping problem (Section 1.2). Then, we present both our MILP model based on *multicommodity-flow* [1] (Sections 3 and 4). This model aims to reconstruct highly reliable haplotypes fragments that will be used in population genomic. Finally, we finish with the presentation of our results validation protocol (Section 5) and a discussion (Section 6).

We currently only have results on toy examples and we do not have any on real data. However, we provide computation times for several real instances. This data shows that the computation time is better for the alleles approach with linearised graph. Result quality has not been studied. However, we proposed a full experimental protocol that will be used to validate the model results.

We introduce a promising new MILP approach for haplotyping and the first MILP formulation of this problem. However, further tests need to be done and the experimental protocol must be implemented. Moreover, this stage the number of haplotypes to reconstruct is fixed by users. We aim to automate this task. In the discussion (Section 6), we also proposed another (not implemented yet) approach allowing to divide by $k$ the number of model's variables (where $k$ is the number of haplotypes to reconstruct).

# Acknowledgement

I would like to thank Pierre Peterlongo and Rumen Andonov for giving me this project and for helping me to realise it. I would like to also thank Amin Madoui who helped me to understand the biological base needed for this work. Finally, I thank the whole GenScale team for giving me the opportunity to present it to ROADEF2020.


# References


[1] R. Ahuja, T. Magnanti, and J. Orlin. *Network flows: theory, algorithms, and applications*. Prentice Hall, 1993.

[2] R. Andonov, H. Djidjev, S. François, and D. Lavenier. Complete Assembly of Circular and Chloroplast Genomes Based on Global Optimization. *Journal of Bioinformatics and Computational Biology*, pages 1–28, 2019.

[3] J. A. Baaijens and A. Schönhuth. Overlap graph-based generation of haplotigs for diploids and polyploids. *Bioinformatics*, 35(21):4281–4289, 2019.

[4] J. A. Baaijens, L. Stougie, and A. Schönhuth. Strain-aware assembly of genomes from mixed samples using variation graphs. *bioRxiv*, 2019.

[5] A. Bankevich, S. Nurk, D. Antipov, A. Gurevich, M. Dvorkin, A. Kulikov, V. Lesin, S. Nikolenko, S. Pham, A. Prjibelski, A. Pyshkin, A. Sirotkin, N. Vyahhi, G. Tesler, M. Alekseyev, and P. Pevzner. Spades: A new genome assembly algorithm and its applications to single-cell sequencing. *Journal of computational biology : a journal of computational molecular cell biology*, 19:455–77, 04 2012.

[6] S. Benidt and D. Nettleton. SimSeq: a nonparametric approach to simulation of RNA-sequence datasets. *Bioinformatics*, 31(13):2131–2140, 02 2015.

[7] Bradley, Hax and Magnanti. *Applied Mathematical Programming*, chapter 9- Integer Programming. Addison-Wesley, 1977.

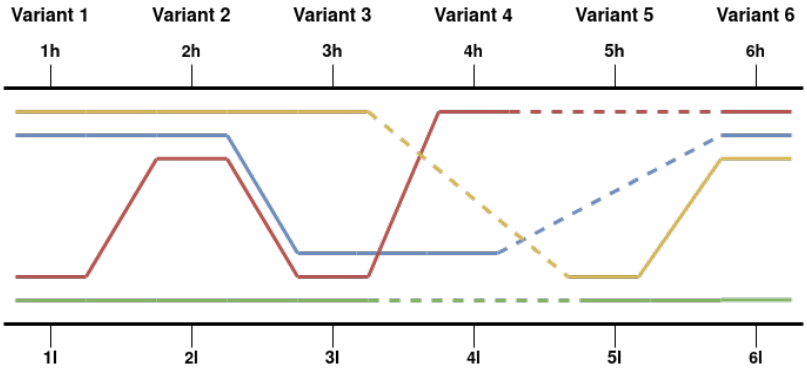
[8] C. E. Miller, A. W. Tucker and R. A. Zemlin. Integer programming formulation of the traveling salesman problems. *Journal of the ACM*, 7 Issue 4:326–329, 1960.

[9] S. François, R. Andonov, D. Lavenier, and H. Djidjev. Global optimization for scaffolding and completing genome assemblies. *Electronic Notes in Discrete Mathematics*, 64:185–194, 02 2018.

[10] R. Laso-Jadart, C. Ambroise, P. Peterlongo, and M.-A. Madoui. metaVaR: introducing metavariant species models for reference-free metagenomic-based population genomics. working paper or preprint, Feb. 2020.

[11] P. Medvedev, K. Georgiou, G. Myers, and M. Brudno. Computability of models for sequence assembly. In R. Giancarlo and S. Hannenhalli, editors, *Algorithms in Bioinformatics*, pages 289–301, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[12] P. Peterlongo, C. Riou, E. Drezen, and C. Lemaitre. Discosnp++: de novo detection of small variants from raw unassembled read set(s). *bioRxiv*, 2017.




| Haplotype | Path | Abundances |
|---|---|---|
| H1 | 1h; 2h; 3l ; 4l ; 6h | 15 |
| H2 | 1l ; 2l ; 3l ; 5l ; 6l | 25 |
| H3 | 1l ; 2h; 3l ; 4h; 6h | 10 |
| H4 | 1h; 2h; 3h; 5l ; 6h | 30 |

| Allele | Abundance | Allele | Abundance |
|---|---|---|---|
| 1h | 35 | 1l | 35 |
| 2h | 55 | 2l | 25 |
| 3h | 30 | 3l | 50 |
| 4h | 10 | 4l | 15 |
| 5h | 0 | 5l | 55 |
| 6h | 70 | 6l | 25 |

Figure 7: Example of 4 haplotypes of 6 variants.
The figure on the left is a visual representation of the 4 haplotypes. Each colored line represents a haplotype. Straigh lines mean that the haplotype passes through the variant, dotted lines mean that the haplotype does not pass through the variant. For example, the haplotype $H1$ (blue line) passes through the following alleles: 1h;2h;3l;4l;6h. It does not pass through the variant 5.
In this example, haplotypes pass either through the variant 4 or through the variant 5.
On the top right is a summary table of the haplotypes and their abundances.
On the bottom right is a summary of the allele abundances. The abundance of an allele is the sum of the abundance of the haplotypes passing through it. The abundance of a variant is the sum of its allele abundances.

| Fact | Alleles |
|---|---|
| 1 | 1h; 2h |
| 2 | 2h; 3l |
| 3 | 2h; 3l; 4l |
| 4 | 4l; 6h |
| 5 | 1l; 2l; 3l |
| 6 | 2l; 3l |
| 7 | 3l; 5l; 6l |
| 8 | 1l; 2h |
| 9 | 3l; 4h |
| 10 | 4h; 6h |
| 11 | 1h; 2h; 3h |
| 12 | 5l; 6h |
| 13 | 3h; 5l |
| 14 | 3l; 5l |

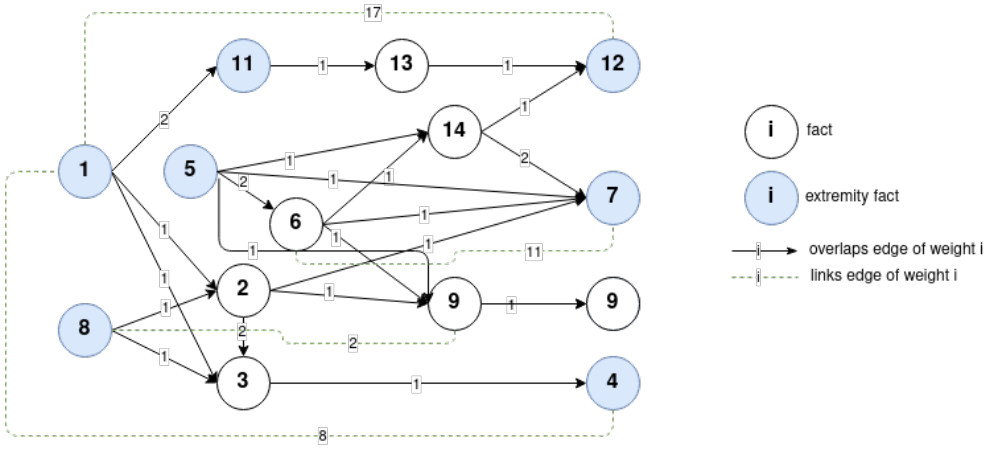



Figure 8: Example of fact graph obtains from the haplotypes shown in figure 7.
On the left is the set of alleles associated with each fact.
On the right is the fact graph. The nodes are the facts and blue nodes are the extremities. Black arrows are the overlaps edges and green dotted lines are the links edges. The value over each edge is its weight. The incompatibles edges are not shown on the figure due to visibility purposes. There are no successive in this example.

| Fact | Alleles |
|---|---|
| 1 | 1h; 2h |
| 2 | 2h; 3l |
| 3 | 2h; 3l; 4l |
| 4 | 4l; 6h |
| 5 | 1l; 2l; 3l |
| 6 | 2l; 3l |
| 7 | 3l; 5l; 6l |
| 8 | 1l; 2h |
| 9 | 3l; 4h |
| 10 | 4h; 6h |
| 11 | 1h; 2h; 3h |
| 12 | 5l; 6h |
| 13 | 3h; 5l |
| 14 | 3l; 5l |

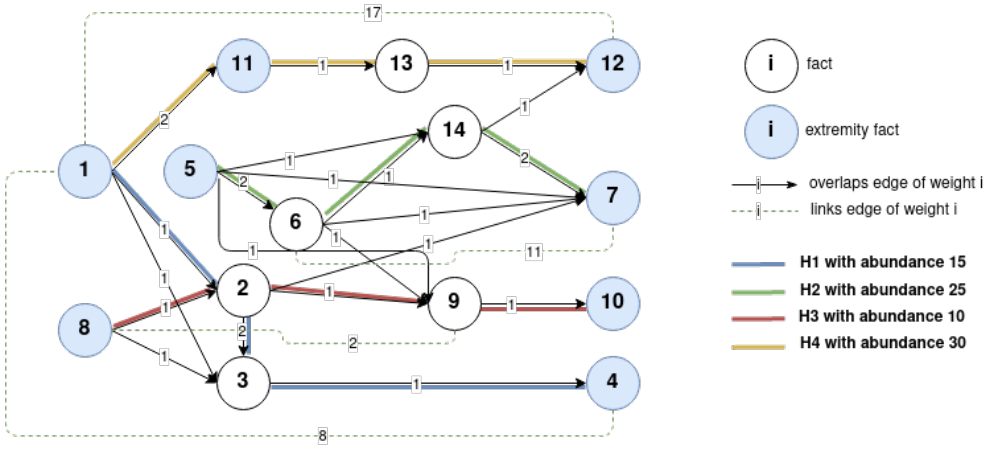



Figure 9: Fact graph presented figure 8 with the reconstructed haplotypes of figure 7. The four links edges are satisfied: $(1, 12)$ is satisfied by $H4$, $(1, 4)$ by $H1$, $(6, 7)$ by $H2$ and $(8, 9)$ by $H3$.

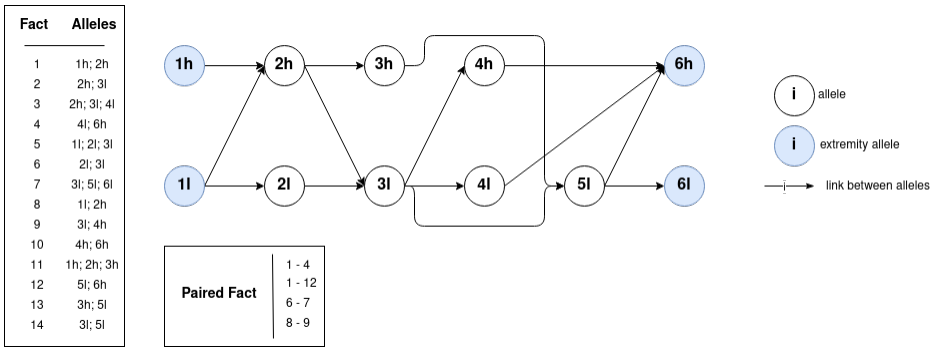



Figure 10: Example of allele graph obtains from the haplotypes shown in figure 7.
On the left is the set of alleles associated with each fact.
On the right is the allele graph. Each node is an allele (blues nodes are extremities alleles). All the nodes of each column belong to the same variant. Black arrows represent the allele succession according to the facts. Notice that a path could either passes by the variant 4 or by the variant 5.
On the bottom-left is the set of paired facts.

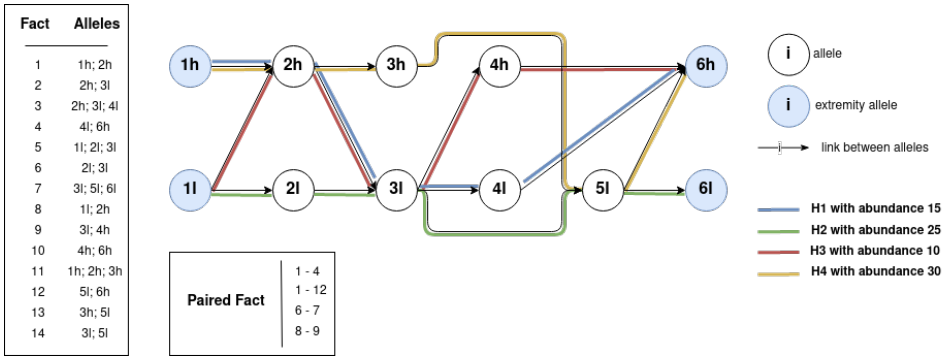



Figure 11: Allele graph presented figure 10 with the reconstructed haplotypes of figure 7. The four links edges are satisfied: $(1, 12)$ is satisfied by $H4$, $(1, 4)$ by $H1$, $(6, 7)$ by $H2$ and $(8, 9)$ by $H3$.

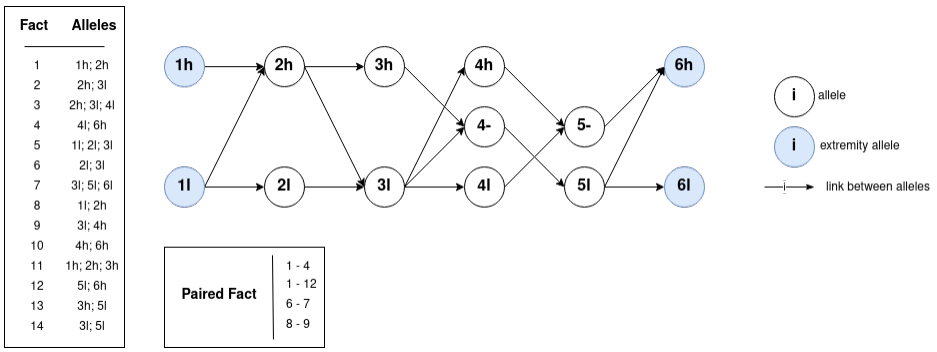



Figure 12: Example of simplified allele graph obtains from allele graph shown in figure 10. On the left is the set of alleles associated with each fact.
On the right is the allele graph. Each node is an allele (blues nodes are extremities alleles). All the nodes of each column belong to the same variant. Black arrows represent the allele succession according to the facts. Unlike the allele graph shown figure 10, each path passes through each variant. The alleles $4-$ of variant 4 and $5-$ of variant 5 are ghost alleles. Ghost alleles are used to allow a path to skip unwanted variants while still passing through each variant.
On the bottom-left is the set of paired facts.

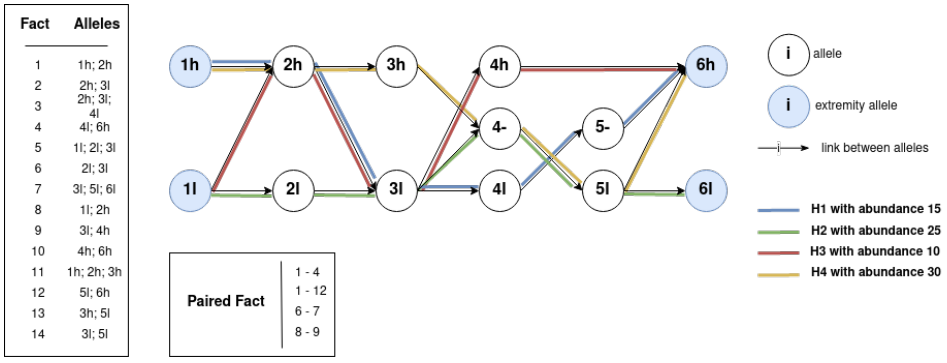



Figure 13: Simplified allele graph presented figure 12 with the reconstructed haplotypes of figure 7. The four links edges are satisfied: $(1, 12)$ is satisfied by $H4$, $(1, 4)$ by $H1$, $(6, 7)$ by $H2$ and $(8, 9)$ by $H3$. Unlike the solution shown figure 11, each haplotype passes through each variant.

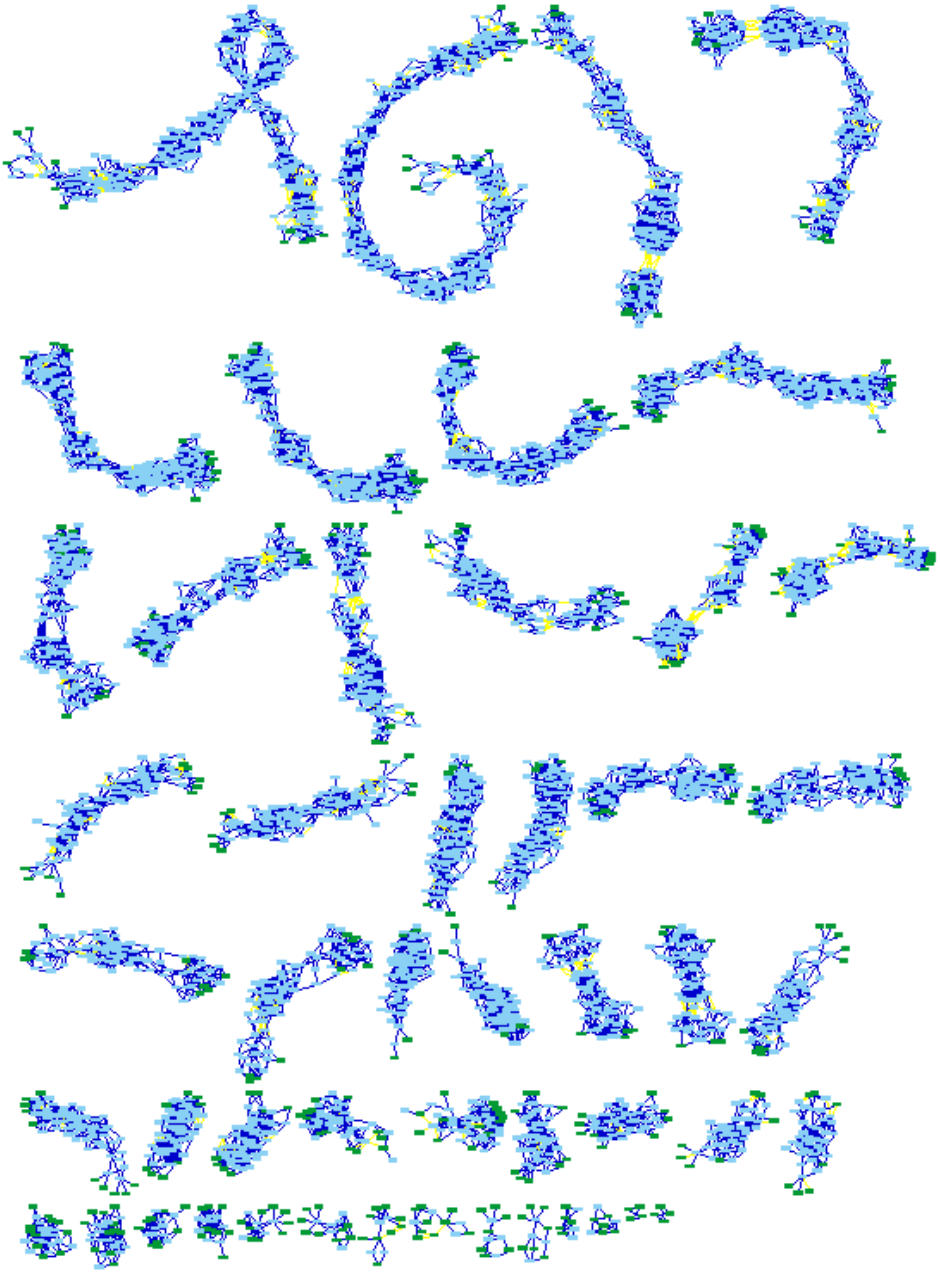


Figure 14: Example of a fact graph of a real data set: 10 haplotypes of SARS-COV-2. This fact graph contains 3 761 facts (including 435 extremities) divided into 25 connected components. The facts are the blue squares, while the green squares are the extremities. These facts phase 1 860 alleles. There are 13 257 overlaps edges (blue edges), 590 successive edges (yellow edges), 32 295 incompatible edges (not shown for visibility purposes) and 36 385 links edges (not shown for visibility purposes). This instance is about 500 times bigger than the toy example instance presented in figure 8.

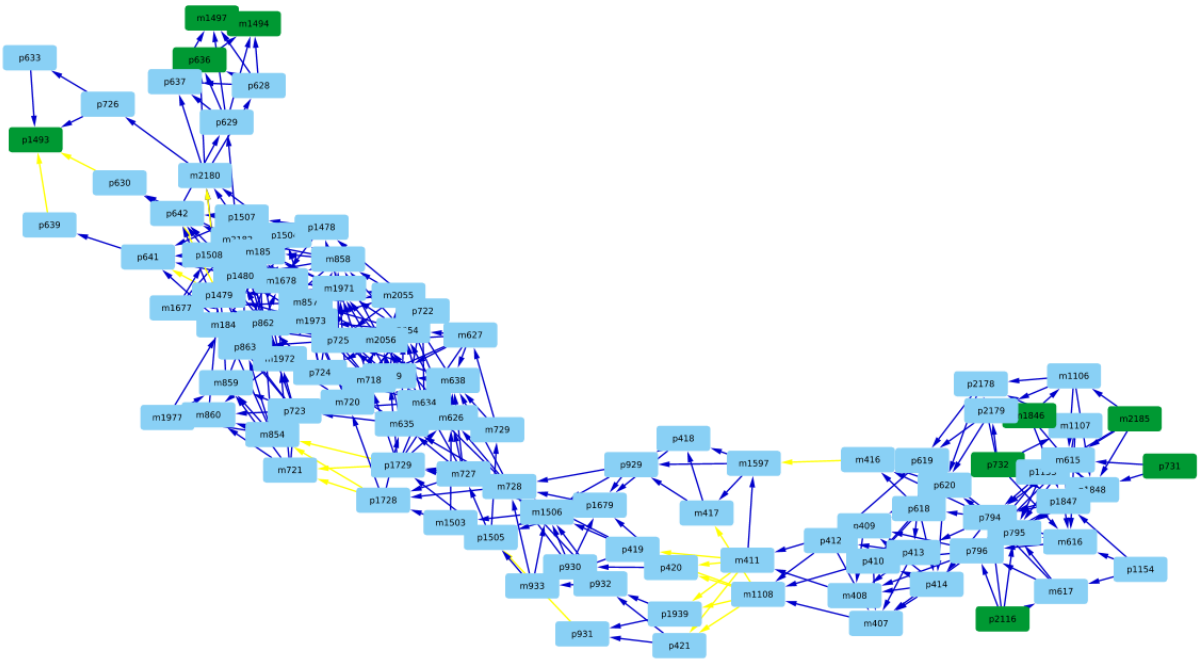



Figure 15: Zoom on one connected component of the SARS-COV-2 fact graph. This connected component is an average one. It contains 210 facts (including 18 extremities). The facts are the blue squares, and the extremities the green squares. There are 677 overlaps edges (blue edges), 39 successive edges (yellow edges), 1 447 incompatible edges (not shown for visibility purposes) and 1 521 links edges (not shown for visibility purposes). This instance is about 20 times bigger than the toy example presented in figure 8.