ITAT 2013 Proceedings, CEUR Workshop Proceedings Vol. 1003, pp. 44–51
http://ceur-ws.org/Vol-1003, Series ISSN 1613-0073, © 2013 M. Kravec, M. Bobák, B. Brejová, T. Vinař

Variants of Genes from the Next Generation Sequencing Data

Martin Kravec, Martin Bobák, Broňa Brejová, and Tomáš Vinař

Faculty of Mathematics, Physics, and Informatics, Comenius University, Mlynská dolina, 842 48 Bratislava, Slovakia

Abstract: A typical next generation sequencing platform produces DNA sequence data of a very fragmented nature, consisting of many short overlapping reads that need to be assembled into a longer DNA sequence by an assembly program. There are many families of genes that evolve by gene duplication. In the genomes, such genes have several copies that are very similar to each other, and therefore they pose a difficult challenge for sequence assembly programs. In this paper, we present a method for recovering variants of genes from the NGS data without need for assembly of the whole DNA sequence. We show that our problem is NP-hard, but also demonstrate that in practice it can be solved by integer linear programming.

1 Introduction

The next generation sequencing (NGS) has brought much cheaper and much faster technologies for obtaining DNA sequences of various organisms. In the process of sequencing, the technology generates many short reads (subsequences of the original DNA sequence), which are often sequenced as pairs with known ordering, orientation, and approximate gap length in the original sequence (see Fig.1). For example, typical reads from Illumina HiSeq sequencing technology are approx. 100 bp long and the length of the gap between paired reads is approx. 60 bp.

The result of genome sequencing is thus a giant puzzle of overlapping pieces that need to be assembled into the original sequence. The most popular methods for assembling short reads are based on creating so called deBruijn graphs [Pevzner et al., 2001], where the problem of sequence assembly is formally transformed into a problem of following a path in a graph. Several practical assembly programs have been developed based on this idea (see e.g., [Zerbino and Birney, 2008]). Since some regions are not necessarily covered by sequencing reads (sequencing gaps), and some regions are too similar to each other in order to be distinguished by short reads (repetitive sequences), the resulting assembly is often fragmented into contigs, that cannot be further assembled without resolving ambiguities or closing the gaps by additional sequencing. To overcome some of these problems, information from paired reads can be used, however, a systematic use of the paired reads has been incorporated into deBruijn graph framework only recently (see e.g., [Medvedev et al., 2011, Bankevich et al., 2012, Pham et al., 2013]).

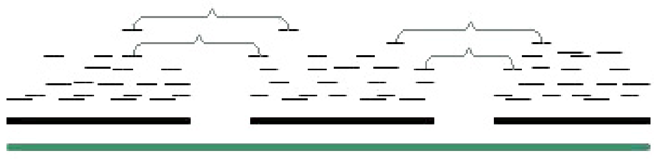

Figure 1: Illustration of genome sequencing data. The result of the sequencing are many short overlapping reads (thin lines, top) from the original sequence (green line, bottom) that can be assembled into longer contigs (thick black lines). Some regions cannot be assembled due to low coverage, resulting in sequencing gaps. Some problems can be solved by considering paired read information (brackets), which discloses order, orientation, and the distance between paired reads.

Assuming that we already have an assembled sequence template, another problem is to align all sequencing reads from a given experiment to the template. Due to overwhelming amount of data, fast text search methods are used (e.g., FM index [Ferragina and Manzini, 2001]) to find perfect or almost perfect matches of reads to the template sequence (see e.g. [Li et al., 2008a, Li et al., 2008b, Lin et al., 2008, Langmead et al., 2009]).

Software for all of these tasks has now become part of a standard bioinformatics toolbox for processing and analysis of the NGS data. However, a single sequencing run can produce up to 50 GB of data [Minoche et al., 2011], and consequently time and memory requirements of these tools impose limitations on researchers working with NGS data.

This has motivated research into use of NGS data without assembling or aligning reads, focusing on sequence patterns on a local scale rather than globally. Targeted assemblers use short regions as seeds to assemble short regions of interest to the researcher [Warren and Holt, 2011, Peterlongo and Chikhi, 2012]. [Philippe et al., 2011] have developed a data structure for querying large read collections in main memory. [Zhai et al., 2012] have developed a framework for approximating the distribution of word occurrences in NGS data without assembly or alignment. GapFiller [Nadalin et al., 2012] utilizes paired reads to locally fill the gaps in the sequence assembly.

In this paper, we will also take this local approach and examine a problem of discovering variants of a given gene from the NGS data. Our problem is motivated by large gene families which arise by repeatedly copying a gene throughout the evolution. An extant genome may contain several copies of a gene at varying levels of similarity, since each copy will mutate independently. We call such gene copies *variants of a gene*.

Variants of a gene, especially those that are very similar to each other, pose a large problem to NGS assembly tools. The individual variations are often mistaken for sequencing errors, or the copies may appear indistinguishable through the short reads, effectively creating an ambiguity in the assembly. Consequently, variants of a gene are often collapsed into a single sequence segment that does not fit very well into the rest of the assembly. The input to our method is an unassembled NGS data set and a gene template. We filter the data set for reads that are similar to the template and use a new algorithm to organize these reads into several variants that explain the variability in the data.

The organization of the paper is as follows. In Section 2, we examine the problem more closely and formulate it as a simple graph problem. In Section 3, we show that our formulation is NP-hard. In Section 4, we propose an integer linear programming (ILP) solution and in Section 5, we provide an algorithm that reduces the size of the ILP. Finally, we demonstrate that our approach can be applied to real data and that the ILP solver (CPLEX) can be used to effectively find the variants of a gene.

2 Problem of Finding Gene Variants

In this section, we formulate the problem of finding gene variants as a graph problem. The input to the problem is a sequence of a reference gene (for example, one gene from a large gene family), and the sequencing reads that align to this reference sequence. While at most positions, the bases in the sequencing reads will likely agree with the reference sequence, there will be some positions where some of the reads disagree with the reference sequence and with each other. We will call these positions *variable positions* and different bases occurring at a variable position will be called *alternatives*.

Definition 1 (Graph of variants)**.** *The* graph of variants *is a directed acyclic graph with several groups of vertices, each group corresponding to one variable position. Each alternative at a variable position is represented by a single vertex in the graph. There is an edge between two vertices if and only if there exists a sequencing read that contains both alternatives corresponding to these two vertices. All edges are directed from left to right according to the position of the corresponding alternative.*

Figure 2 shows a simple example of the graph of variants. The individual reads differ from the reference because they are sequenced from different copies of that particular genes. The alternatives represent mutations and connections between these individual mutations in the reads will allow us to disentangle original gene copies.

Our approach also works with pair-end read pairs. In the variant graph construction, we approach such reads in the

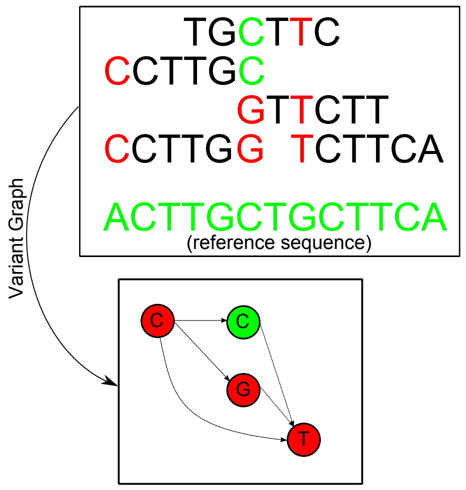

Figure 2: Example of a graph of variants

same way as non-paired reads (i.e., we create an edge for each pair of alternatives covered by a pair of reads). We will assume that the gap between the paired reads is not longer than the standard read length. We will also assume that the sequencing coverage is large; consequently, all possible connections up to some distance will be present in the data. (These assumptions indeed hold in typical sequencing data sets.) This means that if three vertices u, v, and w represent alternatives within the same gene variant such that (u,w) is an edge and (v,w) is an edge, then (u,v) must also be an edge in the variant graph.

An individual gene variant can be represented as a path through the graph of variants that passes through corresponding alternatives at variable positions. Now consider all reads originating from a given gene variant. These reads induce a set of edges in the graph. Only some of these edges are included in the path for the variant, others connect more distant positions along the path. For these edges, we need a notion of explained edges.

Definition 2 (Explaining edges)**.** *Path p* explains *edge (u,v) of variant graph G if both u and v lie on path p.*

In theory, every edge in the variant graph originates from some variant of the gene and thus it should be explained by the path for this variant. Using Occam's razor, we seek to explain all edges using a small number of paths. This leads to the following formulation of the problem of finding gene variants in the genome sequencing data.

Problem 1 (Finding Gene Variants)**.** *Given a graph of variants G, find the smallest set of paths that explain all edges in G.*

Note that the data may give us only incomplete information about variants of genes. If, for example, there is a long region within the gene without any variable positions, there will be no connections between alternatives on the left and the right side of this region. The graph of variants will be split into connected components and there will be no path going from left to right. Yet, the above problem is well defined even in such cases, and the resulting set of paths gives us at least some information on partial variants of genes. However, the interpretation of such results is more difficult. In the rest of the paper, we will assume that the graph of variants is connected.

3 Finding Gene Variants is NP-hard

Unfortunately, the problem of finding gene variants defined in the previous section is NP-hard. We will prove this by reducing 3-SAT problem to the problem of finding gene variants. Consider a formula in with clauses $c_1,\dots,c_n$, where each clause c_i consists of three literals $\ell_{i,1},\ell_{i,2},\ell_{i,3}$, and each literal is either a variable from a set of variables $Z=\{x_1,\dots,x_z\}$, or its negation.

We will construct a graph representing this formula as an instance of finding gene variants problem. An example of this construction is shown in Fig.3.

For each clause c_i and variable x_j, we will have a gadget of five vertices $v_{i,j,1},\dots,v_{i,j,5}$. These vertices are connected by edges forming "a cross gadget" as follows: $(v_{i,j,1},v_{i,j,3}),(v_{i,j,2},v_{i,j,3}),(v_{i,j,3},v_{i,j,4}),(v_{i,j,3},v_{i,j,5})$. There will be an additional level of these cross gadgets that do not correspond to any clause (denote them $v_{0,j,*}$), and there will be additional vertices u_1 and u_2 in this zeroth level, and vertices $w_1,\dots,w_n$ separating the levels for individual clauses. Finally, there will be one designated source vertex s and a sink vertex t.

Starting vertex s will be connected to the vertices in the leftmost level by edges $(s,v_{0,j,1})$ and $(s,v_{0,j,2})$ for $j=1,\dots,z$, and edges (s,u_1), (s,u_2). Connections between individual levels go through the separating vertices w_i, i.e. we have edges $(v_{i-1,j,4},w_i),(v_{i-1,j,5},w_i),(w_i,v_{i,j,1}),(w_i,v_{i,j,2})$, and at the zeroth level also $(u_1,w_1),(u_2,w_1)$. Finally, gadgets at the last level connect to the sink vertex: $(v_{n,j,4},t),(v_{n,j,5},t)$. We will call all these edges *blue edges*.

Green edges will be used to enforce consistency between the gadgets (see below). These will connect vertices as follows: $(v_{0,j,1},v_{i,j,1}),(v_{0,j,2},v_{i,j,2}),(v_{0,j,4},v_{i,j,4}),(v_{0,j,5},v_{i,j,5})$. Finally, red edges will be used to encode the formula in the graph. If clause c_i contains literal x_j, there will be edge $(v_{i,j,1},v_{i,j,5})$. If clause c_i contains literal $\neg x_j$, we will have edge $(v_{i,j,1},v_{i,j,4})$.

Intuitively, paths in the solution will encode the satisfying assignment (if the formula can be satisfied). If the variable x_j is true in the satisfying assignment, then we will have two paths passing through each gadget concerning x_j: $v_{i,j,1}\mapsto v_{i,j,3}\mapsto v_{i,j,5}$ and $v_{i,j,2}\mapsto v_{i,j,3}\mapsto v_{i,j,4}$. We call this a *crossing path* configuration. On the other hand, if the variable x_j is false, we have two paths: $v_{i,j,1}\mapsto v_{i,j,3}\mapsto v_{i,j,4}$ and $v_{i,j,2}\mapsto v_{i,j,3}\mapsto v_{i,j,5}$. We call this an *avoiding path* configuration. Below we will use this intuition to prove that the formula is satisfiable if and only if the smallest number of paths explaining all edges in the corresponding graph is $2z+2$.

Lemma 1. *If the 3CNF formula is satisfiable, there exist $2z+2$ paths explaining all edges in the corresponding graph.*

Proof. Consider a satisfying assignment. For each variable x_j, we will have a pair of paths, that will pass through all cross gadgets corresponding to that variable in crossing (in case that x_j is true) or avoiding configuration (in case that x_j is false). These paths will explain all blue edges except for those involving vertices u_1 and u_2, as well as all green edges, since a particular path will consistently pass each cross gadget corresponding to x_j in the same way. Moreover, these $2z$ paths will also explain at least one of the three red edges at each level, since each clause must have at least one satisfied literal. The remaining edges (those that pass through u_1 and u_2, and at most two red edges at each level corresponding to unsatisfied literals) are easily explained using two additional paths. Therefore $2z+2$ paths suffice to explain all edges in the graph. □

Lemma 2. *If the 3CNF formula is not satisfiable, explaining all edges in the corresponding graph requires at least $2z+3$ paths.*

Proof. Assume that we explain all edges by at most $2z+2$ paths. In order to explain the edges at zeroth level, we need to have two paths that explain edges of the cross gadget corresponding to each variable x_j at level 0; the two additional paths must be used to explain blue edges connected to vertices u_1 and u_2 and consequently they cannot be used to explain any green edges in the graph. Thus, we only have two paths that are usable for explaining green edges connecting to all the other cross gadgets corresponding to variable x_j. From this it follows that the two paths must pass through all cross gadgets corresponding to x_j in crossing or avoiding formation, and consequently they can be interpreted as an assignment of the truth values to the variables. Since this assignment cannot satisfy the 3CNF formula, there must be at least one level where all three red edges remain unexplained. However, we only have two paths (those passing through u_1 and u_2) that could be used to explain those three red edges. This leads to a contradiction and proves, that in such a case we require at least $2z+3$ paths. □

Together, these two lemmas have shown that the edges of a graph constructed from the 3CNF formula can be explained by $2z+2$ paths if and only if the formula is satisfiable. In other words, we have shown by a reduction from 3-SAT:

Corollary 1 (NP-hardness)**.** *The problem of finding gene variants is NP-hard.*

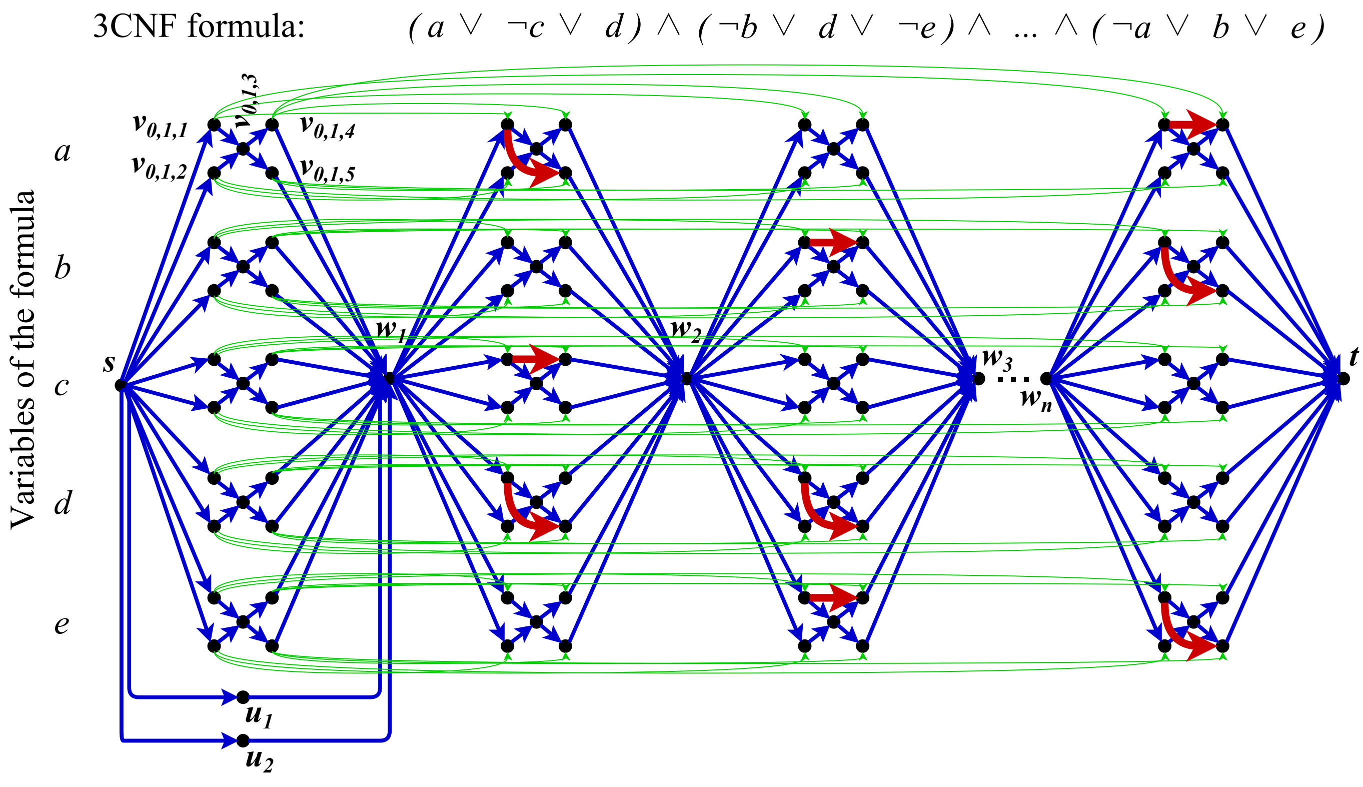
3CNF formula: $(a \vee \neg c \vee d) \wedge (\neg b \vee d \vee \neg e) \wedge \ldots \wedge (\neg a \vee b \vee e)$
Variables of the formula
a
b
c
d
e
s
t
$v_{0,1,1}$
$v_{0,1,2}$
$v_{0,1,3}$
$v_{0,1,4}$
$v_{0,1,5}$
w_1
w_2
w_3
w_n
u_1
u_2

Figure 3: Example of a construction of a graph of variants from the 3CNF formula. Blue edges (regular width) form and connect individual gadgets, while green edges (thin) enforce consistency between individual gadgets, and red edges (thick) encode the formula.

4 Integer Linear Programming Formulation

In the previous section, we have shown that Problem 1 is NP-hard. Here, we propose that the problem can be effectively solved by an integer linear programming (ILP) solver.

For a given graph of variants $G(V,E)$, denote $V_s \subseteq V$ set of vertices without input edges (*source vertices*) and $V_t \subseteq V$ set of vertices without output edges (*sink vertices*). We will assume that the minimum number of paths explaining all edges E is smaller than some number k; this can be set for example as the number of edges in G, but in practice much smaller values of k will suffice.

We will have three types of 0-1 variables, each set of variables replicated for each possible path. For path number i ($1 \leq i \leq k$), variable $x_{i,e} = 1$ if edge e is included in the path; variable $y_{i,e} = 1$ if edge e is explained by the path; variable $z_{i,v} = 1$ if vertex v is included in the path.

Our goal is to minimize the number of paths, which can be, for example, computed as the number of source vertices used in the solution $\sum_{i=1}^{k} \sum_{v \in V_s} z_{i,v}$. This sum is minimized subject to the following conditions:

$$\forall i : \forall v \in V - V_s : z_{i,v} = \sum_{(u,v) \in E} x_{i,(u,v)} \tag{1}$$

$$\forall i : \forall v \in V - V_t : z_{i,v} = \sum_{(v,w) \in E} x_{i,(v,w)} \tag{2}$$

$$\forall i : \sum_{v \in V_s} z_{i,v} \leq 1 \tag{3}$$

$$\forall i : \sum_{v \in V_t} z_{i,v} \leq 1 \tag{4}$$

$$\forall i : \forall (u,v) \in E : y_{i,(u,v)} \leq \frac{1}{2}(z_{i,u} + z_{i,v}) \tag{5}$$

$$\forall (u,v) \in E : \sum_{i=1}^{k} y_{i,(u,v)} \geq 1 \tag{6}$$

$$\forall i : \sum_{v \in V} id_v z_{i,v} \geq \sum_{v \in V} id_v z_{i+1,v} \tag{7}$$

$$\forall i : \forall (u,v) \in E : x_{i,(u,v)} \in \{0,1\} \tag{8}$$

$$\forall i : \forall (u,v) \in E : y_{i,(u,v)} \in \{0,1\} \tag{9}$$

$$\forall i : \forall v \in V : z_{i,v} \in \{0,1\} \tag{10}$$

where id_v is a unique identifier of vertex v. Conditions (1)-(4) ensure consistency between assignments of X and Z variables and at the same time, they enforce that the object represented by the i-th group of variables is indeed a path. Condition (5) determines explained edges and condition (6) assures that each edge is explained by at least one path. Finally, condition (7) helps to break symmetries leading to many equivalent solutions (problems having many symmetric solutions may be difficult to solve).

This ILP has $O((m+n)k)$ variables and conditions. Variables $x_{i,e}$ fully determine the solution; the other variables are auxiliary and can be uniquely determined from $x_{i,e}$. In the next section, we show how to reduce the number of determining variables which will increase the efficiency of the ILP in practice.

5 Reduced Graph of Variants

In this section, we introduce an algorithm that will reduce the number of edges in the graph of variants. In particular, we will only be interested in the subset of edges that are needed for the optimal solution.

Definition 3 (Reduced graph of variants)**.** *Let* $\{p_1, p_2, \dots, p_k\}$ *be the optimal set of paths explaining all edges in the graph of variants.* Reduced graph of variants *is a union of all edges of paths* $p_1, \dots, p_k$*. In case of multiple optimal solutions we choose the smallest such set of edges.*

Although computing the optimal set of paths is hard, union of edges on these paths can be found by a simple algorithm. A *detour* for an edge (v_i, v_j) is a path of length at least two connecting v_i and v_j. If an edge e does not have a detour, it can be explained only by paths contaning e, and thus it must be included in the reduced graph. Conversely, if e has a detour, any path in the optimal set can use edges in the detour instead of e. Any optimal set of paths can be thus transformed to contain only edges without a detour and these edges form the reduced graph.

To find the reduced graph, we number the vertices of the variant graph $v_1, \dots v_n$ from left to right. Let M_i be the set of vertices, from which vertex v_i is reachable. Algorithm 1 computes sets M_i for all vertices v_i and simultaneously selects edges for the reduced graph of variants.

For each vertex v_i, we compute set M_i by exploring incoming edges (v_j, v_i) in the descending order by v_j. If v_j is not already in M_i, then we add all vertices from M_j to M_i. In addition, there is no detour for (v_j, v_i), because such a detour would have to enter v_i by and edge (v_k, v_i) with $k > j$. Since set M_k for such k contains v_j, v_j would be already in M_i when we encounter edge (v_j, v_i), which is a contradiction. Therefore we select edge (v_j, v_i) for the reduced graph. On the other hand, if v_j is already in M_i, it can be easily seen that a detour must exist for (v_j, v_i).

The running time of this algorithm is $O(n^3)$ due set manipulation on line 8. Instead of this simple algorithm, we could also use faster transitive closure computation based on fast matrix multiplication, but this algorithm is sufficient for our purposes. Example of a reduced graph of variants can be seen in Figure 4.

Given a reduced graph of variants $G_r(V_r, E_r)$, we can modify the ILP program by including variables $x_{i,(u,v)}$ only

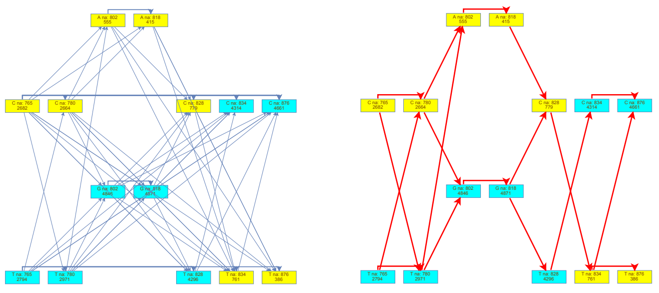

Figure 4: Example graph of variants (left) and reduced graph of variants (right).

for edges $(u,v) \in E_r$, because the remaining edges will not be used by the optimal set of paths. We will change the conditions as follows:

$$\text{Eq. (1)} \quad \forall i : \forall v \in V - V_s : z_{i,v} = \sum_{(u,v) \in E_r} x_{i,(u,v)} \tag{11}$$

$$\text{Eq. (2)} \quad \forall i : \forall v \in V - V_t : z_{i,v} = \sum_{(v,w) \in E_r} x_{i,(v,w)} \tag{12}$$

$$\text{Eq. (8)} \quad \forall i : \forall (u,v) \in E_r : x_{i,(u,v)} \in \{0,1\} \tag{13}$$

We also add one more condition stating that each edge from E_r must be used on some path:

$$\forall (u,v) \in E_r : \sum_{i=1}^{k} x_{i,(u,v)} \geq 1 \tag{14}$$

6 Experiments and Results

We tested our approach on Illumina sequencing data from yeast species *Magnusiomyces magnusii* (unpublished). For out tests, we have selected FDH (formate dehydrogenase) gene that exhibits variation in the number of copies in related species (for example, two copies in *S. cerevisiae*, three copies in *C. albicans*, and ten copies in *Y. lipolytica*). The sequence assembly by assembly program Velvet [Zerbino and Birney, 2008] contained several fragments of FDH-like genes (likely originating from different gene copies), but all of them were apparently incomplete. We have manually assembled some of these fragments and created a putative 1150 bp long template of the FDH gene. By a rather complicated process using blastx [Altschul et al., 1997], blat [Kent, 2002], and Velvet [Zerbino and Birney, 2008], we have selected 74 694 read pairs similar to the FDH template. The average length of a read is 100.8 bp, and the expected length of the gap between the paired reads is 60.

The individual reads were then aligned to the FDH template by blat. Based on read alignments, we have identified all variable positions and alternatives. To avoid sequencing errors, we have discarded all alternatives that accounted for less than 10% of reads at each variable position. The ILP was solved by CPLEX optimization library, the maximum number of paths was set to 30.

Algorithm 1 Algorithm for reducing the variant graph

```
 1: Input: Variant graph G(V,E)                    ▷ vertices of V are in topological order
 2: selectedEdges = { }
 3: for $i = 1 \ldots n$ do
 4:     $M_i = \{v_i\}$
 5:     for $j = i-1 \ldots 1$ ; $(v_j, v_i) \in E$ do      ▷ iterate through incoming edges in descending order
 6:         if $v_j \notin M_i$ then
 7:             $M_i = M_i \cup M_j$
 8:             selectedEdges = selectedEdges $\cup \{(v_j, v_i)\}$
 9:         end if
10:     end for
11: end for
12: return selectedEdges
```

Figure 5 shows the result of our experiment. The number of edges in the graph of variants can be significantly reduced, thus speeding up the ILP. The algorithm has discovered six variants of FDH gene in *M. magnusii* genome. Our experiments shows that even though, in general, the problem is NP-hard, it is possible to apply the ILP on the real data.

7 Conclusion and Future Work

In this paper, we have introduced a problem of recovering all variants of a given gene from unassembled next generation sequencing data. We have formulated the problem as a graph problem, shown that the problem is NP-hard, and proposed a practical solution with the integer linear program. We have shown that our solution can indeed be applied to real data, and we have detected six variants of the FDH gene in *M. magnusii* genome.

There are several problems open for future work. In this work, we have used pair-end reads, i.e. pairs of short reads that are separated by a short gap (in this work, we have assumed that the gap between the two paired reads is shorter than the read length). On the other hand, mate-pair sequencing produces paired reads that are at a much longer distance. How do we incorporate such reads into our framework?

Here, we have considered variants where only a single base has changed at a particular position. However, other variants, such as short insertions or deletions, or even larger scale changes, do occur in real data and they need to be accounted for.

One information that we ignored is the number of reads overlapping each position of the template sequence. This number is correlated with the number of copies of a particular segment of the sequence. Such correlations can be used to help to decide between alternative solutions (each variant should have the same coverage over its whole length).

Finally, sequences flanking the template sequence on each side will also differ between individual copies of a gene and may help to pinpoint the correct number of variants.

Acknowledgements. We thank prof. Jozef Nosek for access to the unpublished sequencing data and for suggesting FDH gene as an interesting target for study. This research was supported by VEGA grant 1/1085/12 and by Comenius University young researcher grant UK/425/2013.

References

[Altschul et al., 1997] Altschul, S. F., Madden, T. L., Schaffer, A. A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D. J. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res*, 25(17):3389–3392.

[Bankevich et al., 2012] Bankevich, A., Nurk, S., Antipov, D., Gurevich, A. A., Dvorkin, M., Kulikov, A. S., Lesin, V. M., Nikolenko, S. I., Pham, S., Prjibelski, A. D., Pyshkin, A. V., Sirotkin, A. V., Vyahhi, N., Tesler, G., Alekseyev, M. A., and Pevzner, P. A. (2012). SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *J Comput Biol*, 19(5):455–457.

[Ferragina and Manzini, 2001] Ferragina, P. and Manzini, G. (2001). An experimental study of an opportunistic index. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete algorithms (SODA)*, pages 269–278.

[Kent, 2002] Kent, W. J. (2002). BLAT–the BLAST-like alignment tool. *Genome Res*, 12(4):656–664.

[Langmead et al., 2009] Langmead, B., Trapnell, C., Pop, M., and Salzberg, S. L. (2009). Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol*, 10(3):R25.

[Li et al., 2008a] Li, H., Ruan, J., and Durbin, R. (2008a). Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome Res*, 18(11):1851–1858.

[Li et al., 2008b] Li, R., Li, Y., Kristiansen, K., and Wang, J. (2008b). SOAP: short oligonucleotide alignment program. *Bioinformatics*, 24(5):713–714.

[Lin et al., 2008] Lin, H., Zhang, Z., Zhang, M. Q., Ma, B., and Li, M. (2008). ZOOM! Zillions of oligos mapped. *Bioinformatics*, 24(21):2431–2437.

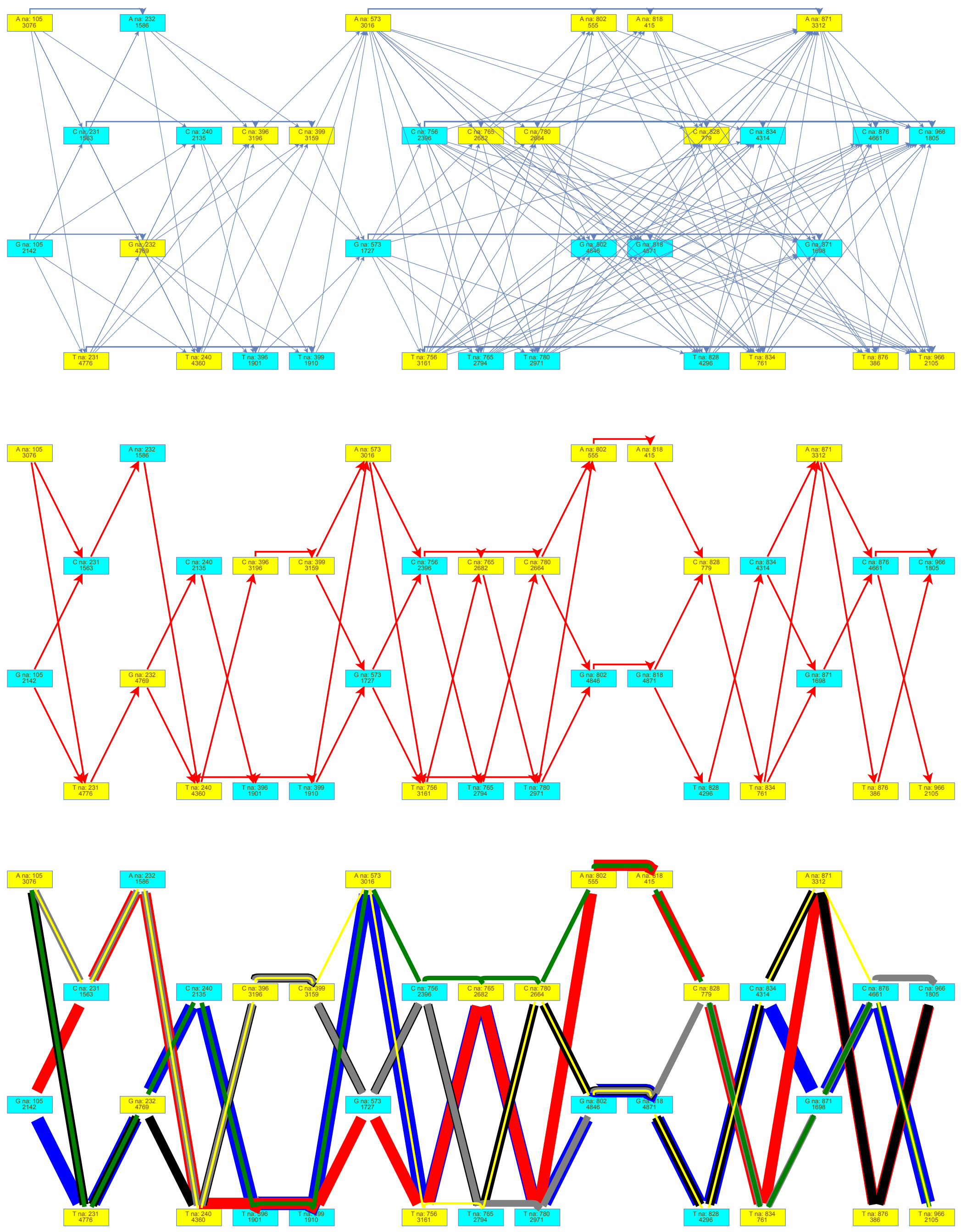

Figure 5: Variants of FDH gene in *M. magnusii* genome. (top) The full graph of variants; only alternatives accounting for more than 10% of aligned genes were included. (middle) Reduced graph of variants significantly reduces the number of edges. (bottom) Six variants discovered by the ILP; each variant is shown in a different color.

[Medvedev et al., 2011] Medvedev, P., Pham, S., Chaisson, M., Tesler, G., and Pevzner, P. (2011). Paired de bruijn graphs: a novel approach for incorporating mate pair information into genome assemblers. *J Comput Biol*, 18(11):1625–1634.

[Minoche et al., 2011] Minoche, A. E., Dohm, J. C., and Himmelbauer, H. (2011). Evaluation of genomic high-throughput sequencing data generated on Illumina HiSeq and genome analyzer systems. *Genome Biol*, 12(11):R112.

[Nadalin et al., 2012] Nadalin, F., Vezzi, F., and Policriti, A. (2012). GapFiller: a de novo assembly approach to fill the gap within paired reads. *BMC Bioinformatics*, 13 Suppl 14:S8.

[Peterlongo and Chikhi, 2012] Peterlongo, P. and Chikhi, R. (2012). Mapsembler, targeted and micro assembly of large NGS datasets on a desktop computer. *BMC Bioinformatics*, 13:48.

[Pevzner et al., 2001] Pevzner, P. A., Tang, H., and Waterman, M. S. (2001). An Eulerian path approach to DNA fragment assembly. *Proc Natl Acad Sci U S A*, 98(17):9748–9753.

[Pham et al., 2013] Pham, S. K., Antipov, D., Sirotkin, A., Tesler, G., Pevzner, P. A., and Alekseyev, M. A. (2013). Pathset graphs: a novel approach for comprehensive utilization of paired reads in genome assembly. *J Comput Biol*, 20(4):359–361.

[Philippe et al., 2011] Philippe, N., Salson, M., Lecroq, T., Leonard, M., Commes, T., and Rivals, E. (2011). Querying large read collections in main memory: a versatile data structure. *BMC Bioinformatics*, 12:242.

[Warren and Holt, 2011] Warren, R. L. and Holt, R. A. (2011). Targeted assembly of short sequence reads. *PLoS One*, 6(5):e19816.

[Zerbino and Birney, 2008] Zerbino, D. R. and Birney, E. (2008). Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome Res*, 18(5):821–829.

[Zhai et al., 2012] Zhai, Z., Reinert, G., Song, K., Waterman, M. S., Luan, Y., and Sun, F. (2012). Normal and compound poisson approximations for pattern occurrences in NGS reads. *J Comput Biol*, 19(6):839–844.