HAL
archives-ouvertes.fr

A Graph Constraints Formulation for Contigs Scaffolding

Rodolphe Giroudeau, Annie Château, Clément Dallard, Eric Bourreau

▶ **To cite this version:**

Rodolphe Giroudeau, Annie Château, Clément Dallard, Eric Bourreau. A Graph Constraints Formulation for Contigs Scaffolding. WCB: Workshop on Constraint-Based Methods for Bioinformatics, Sep 2016, Toulouse, France. 12th International Workshop on Constraint-Based Methods for Bioinformatics, pp.136-149, 2016. <lirmm-01360463>

HAL Id: lirmm-01360463
https://hal-lirmm.ccsd.cnrs.fr/lirmm-01360463

Submitted on 5 Sep 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Graph Constraints Formulation for Contigs Scaffolding

Éric Bourreau[1], Annie Chateau[1,2], Clément Dallard[1], Rodolphe Giroudeau[1]

[1] LIRMM - CNRS UMR 5506 - Montpellier, France
[2] IBC - Montpellier, France
{eric.bourreau,annie.chateau,clement.dallard,rodolphe.giroudeau}@lirmm.fr

Abstract. This paper presents a constraint-based approach for genome scaffolding, which is one important step in genome whole sequence production. We model it as an optimization problem on a graph built from a paired-end reads mapping on contigs. We describe our constraint model using a graph variable representation with classical graph constraints. We tested our approach together with several search strategies, on a benchmark of various genomes.

1 Introduction

Last decade was marked by the race to the production of new genomic sequences. Since new technologies of sequencing, known as *High Throughput Sequencing* (*HTS*) or *New Generation Sequencing* (*NGS*), are available, price of genome sequencing has consequently dropped. Technological advances in sequencing, but also in computer science, allow to conduct studies involving tens of thousands of genomes of the same species or related species. The projects "1000 genomes" bloom, and necessit a phenomenal processing power. However *HTS* is mostly based on a technology which splinters the genomic sequence, resulting in a large amount of paired-end reads even for quite small genomes. Most sequencing projects are experiencing bottlenecks in the production of complete sequences from the sequencing libraries, and produce genomes often in draft form. Hence, assembling and scaffolding steps have to be as optimized as possible to obtain a satisfying solution in reasonable time. One of the crucial steps involved in this process is genome *scaffolding*. Once sequencing and assembly of `DNA` molecules have been performed, we end up with a set of genomic sequences of various lengths, called *contigs*, representing pieces of genome. The main goal of scaffolding process is to find an order and an orientation on these contigs such that resulting collections of oriented contigs map as good as possible to the reference genome. Such collections are named *scaffolds* and would ideally represent the genome chromosomes, which could be either linear or circular.

The scaffolding process has been modeled as various combinatorial problems which are unfortunately computationally hard [1,2]. This observation naturally encourages to try different ways to solve the problem, from heuristic, approximation or exact resolution point of view. Most of current scaffolding solvers use heuristic methods to solve this problem. These solvers, unfortunately, do not

propose any confidence on the optimality of the solution they return. Some of them, like *Bambus* [3], *SSPACE* [4] and *SSAKE* [5], directly solve the graph problem using greedy algorithms. Their first obvious interest is their time complexity, since the corresponding algorithms are strictly polynomial. However, the solution may be very faulty since the graph is sometimes simplified and because it only guarantees a local maximum. Other solvers like *Bambus2* [6] uses structures recognition and $\mathcal{NP}$-hard problem's approximations to generate scaffolds. *Opera* [7] uses a graph contraction method before solving the scaffolding on the smaller contracted graph. However, the contraction step may alter the original graph such that the optimal solution on the contracted graph is not optimal on the original one. *SCARPA* [8] combines *Fixed Parameter Tractable* algorithm (to remove odd cycles on the original graph) and mixed integer programming (to join contigs in scaffolds). Once again the yielded solution is not necessarily the optimal solution, because of the odd cycle deletion. *GRASS* [9] and *MIP Scaffolder* [10] use mixed integer programs to solve the scaffolding problem, but always on a simplified graph and then can not be considered as exact methods either.
A previous work using an incremental strategy and Integer Linear Programming (ILP) was proposed in [11]. After several attempts to model the scaffolding problem with CSP, the authors finally chose a simple ILP formulation instead, in order to achieve scalability. However, this formulation was not totally satisfying, since it could not prevent from small circular chromosomes in the solution. Thus, it has to be cured with an iterative treatment to forbid those cycles. As one can observe, there is no solver offering exact resolution for the scaffolding problem, possibly resulting in different solutions from different solvers working on a same graph. In the present work, we choose the CSP approach, to attack this problem. Using a recent library, dedicated to graphs, namely Choco-graph[3] [12], we formulate the contig scaffolding problem in a simple manner, given in Section 3. We discuss how search strategies could have an effect on the efficiency of the method, and run some experiments on a dataset of small instances, in Section 4.

2 Notation and description of the problem

In what follows, we consider $G = (V, E)$ an undirected graph with an even number $2n$ of vertices and without self loops. We suppose that there exists a perfect matching in G, denoted by M^*. Let $w : E \to \mathbb{N}$ be a weight function on the edges. In the bioinformatic context, edges in M^* represent contigs, and the other edges figure the ways to link the contigs together, their weight representing the support of each of these hypotheses (e.g. the number of pairs of reads matching on both contigs). Figure 1 shows an example of a scaffold graph.

[3] https://github.com/chocoteam/choco-graph

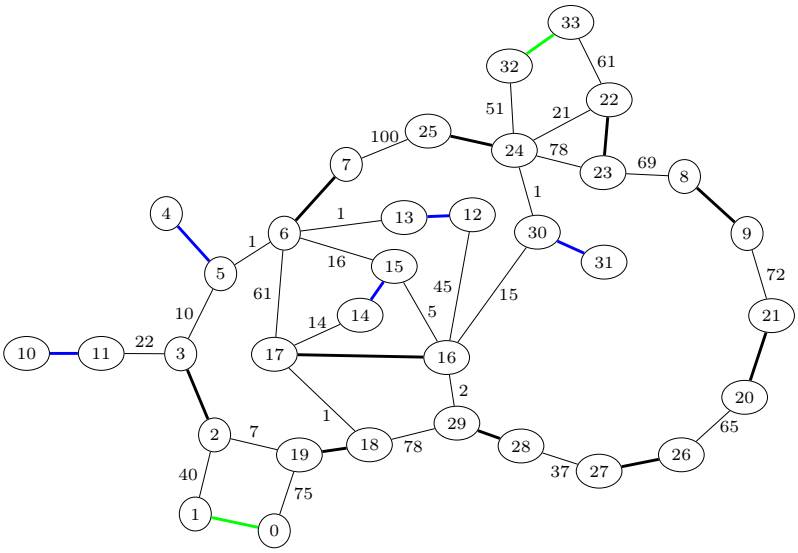

Fig. 1. A scaffold graph with 17 contigs (bold edges) and 26 (weighted) links between them, corresponding to the `ebola` data set (see Section 4). Contig-edges in green and blue correspond to contig of size small enough to be inserted between two other contigs, leading to possible misplacements in the final scaffolding. Green contigs are still well placed in the solution, but blue one are ambiguous.

In order to model the genomic structure by fixed numbers of linear (paths) and circular (cycles) chromosomes, the class of considered problems are parameterized by two integers, respectively denoted by σ_{p} and σ_{c}. These parameters correspond to what is *desired* as genomic structure, however it is not always possible to exactly obtain this structure. Thus we consider a relaxed version of the problem, called Scaffolding, together with the original one, denoted as Strict Scaffolding.

We may use the notation $\mathrm{Gr}(\cdot)$ to denote the induced graph of an edge set. For instance, let $G = (V, E)$ be a graph, then $\mathrm{Gr}(E) = G$.

In the following, we call *alternating cycle* (resp. *alternating path*) in G, relatively to a perfect matching M^* of G, a cycle (resp. a path) with edges alternatively belonging to M^* or not (resp. and where extremal edges belong to M^*). Notice that an alternating-cycle (resp. alternating-path) has necessarily an even number of vertices, at least four (resp. two). The class of Scaffolding problems are defined as follows:

Scaffolding (SCA)
Input: $G = (V, E)$, $\omega : E \to \mathbb{N}$, perfect matching M^* in G, $\sigma_{\mathrm{p}}, \sigma_{\mathrm{c}}, k \in \mathbb{N}$
Question: Is there an $S \subseteq E \backslash M^*$ such that $\mathrm{Gr}(S \cup M^*)$ is a collection of $\geq \sigma_{\mathrm{p}}$ alternating paths and $\leq \sigma_{\mathrm{c}}$ alternating cycles and $\omega(S) \geq k$?

The variant of the problem that asks for *exactly* σ_p paths and *exactly* σ_c cycles is called Strict Scaffolding (SSCA). When we want to precise particular values for σ_p and σ_c, we refer to the problem as $(\sigma_p, \sigma_c)-$Scaffolding. We refer to the optimization variants of Scaffolding that ask to minimize or maximize $\omega(S)$ as Min Scaffolding and Max Scaffolding, respectively. In what follows, we mainly focus on both problems Max Scaffolding and Max Strict Scaffolding, which correspond to the bioinformatic problem.

Problems Strict Scaffolding and Scaffolding received much attention in the framework of complexity and approximation [2,13]. In these articles, the authors showed the hardness of Scaffolding even in presence of restricted parameters or graph classes (cycle length, bipartite planar graph, tree, ...). Some lowers bounds according to complexity hypothesis ($\mathcal{P} \neq \mathcal{NP}$, $\mathcal{ETH}$) are proposed using the Gap-reduction, and reduction preserving lower bound in the field of exact exponential time algorithms. On positive side, some upper bounds with efficient polynomial-time approximation algorithms are designed. Theoretical aspects of Scaffolding are completed by a parameterized complexity approach in [14] and [13]. In such context, the authors proposed some negative results about the quest of a polynomial kernel, or a $\mathcal{FPT}$-algorithm.

3 Model and algorithms

In this section, we propose to solve, sometimes to optimality, the Strict Scaffolding problem with Constraint Programming. First, we remind some definitions, especially on not so classical graph variables. Then we describe variables and constraints used to model Strict Scaffolding. Finally we describe search strategies used to solve the optimization problem.

a - Constraint Programming Definitions

Definition 1. *A **domain** of a variable x, denoted $D(x)$, is a bounded set of values which can be affected to x. We note $\underline{x}$ (resp. $\overline{x}$) the lower bound (resp. upper bound) of domain $D(x)$.*

Definition 2. *A **constraint** $C_i \in \mathcal{C}$ on the subset of m variables $\mathcal{X}(C_i) = \{x_{i_1}, x_{i_2}, \ldots, x_{i_m}\}$ is a subset of $D(x_{i_1}) \times D(x_{i_2}) \times \ldots \times D(x_{i_m})$. It determines the m-tuples of values allowed to be assigned to variables $x_{i_1}, x_{i_2}, \ldots, x_{i_m}$.*

Definition 3. *A **CSP** is a triplet ($\mathcal{X}$,$\mathcal{D}$,$\mathcal{C}$), where $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ is a set of variables, $\mathcal{D} = \{D(x_1), D(x_2), \ldots, D(x_n)\}$ is a set of domains describing the different values which could be assigned to the variables in $\mathcal{X}$ and $\mathcal{C}$ is a set of constraints between the variables in $\mathcal{X}$.*

Definition 4. *A **solution** of a CSP is a set of assignments of values to variables, $\{(x_1, a_1), (x_2, a_2), \ldots, (x_n, a_n)\}$, with $\forall i \in [1, n], a_i \in D(x_i)$, satisfying all constraints in $\mathcal{C}$.*

Let us remark these general definitions related to CSP do not fix the possible types of variables yet. Classically, we manipulate integer values in integer domains for integer domain variables, but more recently, set variables were introduced and even graph variables [15,16], yielding easier modeling. We exploit here those improvements to expressivity.

Concerning a set variable x, there exists a compact representation of the domain $D(x)$, where we specify two sets of elements [17]: the set of elements that *must* belong to the set assigned to x (which we still call the lower bound $\underline{x}$) and the set of elements that *may* belong to this set (the upper bound $\overline{x}$). The domain itself can be represented as a lattice structure corresponding to the partial order defined by sets inclusion. When defined in Choco, a set variables are encoded with two classical bounds: the union of the all set of possible values called the *envelope* and the intersection of all set of possible values called the *kernel*.

Generalizing this point of view, a graph can be seen as two sets V and E with an inherent constraint specifying that $E \subseteq V \times V$. The domain $D(G)$ of a graph variables G is specified by two graphs: a lower bound graph $\underline{G}$ and an upper bound graph $\overline{G}$, such that the domain is the set of all subgraphs of the upper bound which are supergraphs of the lower bound. For a better understanding of graph variables, we refer to Section "3.10. Graph based Constraints" of the Global Constraint review [18],

b - Constraint Model

Variables As previously described, we introduce an undirected graph variable $Gv(Vv, Ev)$ (Gv for Graph variable) whose value will represent the underlying solution. The variable $Gv(Vv, Ev)$ is derived from the graph $G(V, E)$ previously defined in Section 2, with allowed self loops on vertices. Although in the original formulation of Strict Scaffolding problem, there are no self loops allowed, here they will help us to differentiate cycles, which do not contain any, and paths with a final loop on each extremity. A self loop incident to a vertex u counts for one in its degree. All vertices are mandatory and simply belong to $\underline{\text{Vv}}$. Edges in M^* are also mandatory and belong to $\underline{\text{Ev}}$. We will look for a solution by adding, or not, remaining edges to the kernel.

For cost optimisation, we decide to manipulate only $|Vv|$ integer variables to handle the positive weights on a solution. Let's remark this is a more compact model than having $|Ev|$ variables to represent null of positive weights on edges. We denote by $wmax$ the maximum weight of an edge that could be met in the graph, and $E = (E_{ij})$ a boolean variable matrix channeling the graph Gv (*i.e.* if edge between i and j is in the kernel, $E_{ij} = 1$; if the edge is out of the envelope, $E_{ij} = 0$; else domain is $\{0, 1\}$) The integer variables are:

- a vector $Weights$ of size $|Vv|$. The variable $Weights[u]$ represents the sum of weights of edges incident to node u in the solution. Its domain $D(Weigths[u])$ is the set $\{0, \ldots, wmax + 1\}$. In order to count each weight only once in the solution, we derive from the weight function w defined in Section 2 an upper triangle weight matrix denoted by w^*.

– a variable $TotalWeight$ to sum up all the $Weights$. We can restrict initial domain of $TotalWeight$ to lower and upper bound computed by any heuristic described in previous sections. Here it belongs to $\{0, \ldots, |Vv| * wmax\}$.

We define the following constraint model, very simply defining the problem:

CSP Model : Scaffold Problem with Graph Var and Graph Constraints

$$connected\_components(Gv, \sigma_{\mathrm{p}} + \sigma_{\mathrm{c}}) \tag{1}$$

$$minDegree(Gv, 2) \tag{2}$$

$$maxDegree(Gv, 2) \tag{3}$$

$$nbLoops(Gv, 2\sigma_{\mathrm{p}}) \tag{4}$$

$$Weights[i] = \sum_{j \in Vv} (w_{ij} E_{ij}) \qquad \forall\, i \in Vv \tag{5}$$

$$TotalWeight = \sum_{i \in Vv} Weights[i] \tag{6}$$

In Equation 1, Gv is constrained to have a specific number of connected components (σ_{p}+σ_{c}). By default choco solver use fast filtering rules, first computing all connected components of G by performing one Depth First Search (using Tarjan algorithm [19]) where time complexity is $\mathcal{O}(|Vv| + |Ev|)$ and check their number according to the parameters. If necessary, better propagation can be performed by looking for articulation points in time complexity $\mathcal{O}(|Vv|.|Ev|)$, or even better by managing dominator [20]. This was not necessary in choco to get good performances.

In Equation 2 and Equation 3, we linearly maintain degree for each node u in Gv by checking size of upper bound and lower bound of variable-set E_u, designing edges incident to vertex u, and if necessary by applying complete filtering, removing (or forcing) associated edges. Moreover, since $M^* \subseteq Ev$ we necessarily construct alternating paths and cycles when adding consistent edges to the kernel.

In Equation 4, number of nodes with self loop is linearly maintained to adjust the parameter with complete filtering when bounds are reached. As explained, a cycle will not have self loops but paths will have it at extremities (only one due to upper triangle matrix). Then, without the need to distinguish paths and cycles, this constraint guarantees the solution to have exactly σ_{p} paths. Since Equation 1 fixes the number of connected components, we also have exactly σ_{c} cycles.

The remaining Equation 5 and Equation 6 are simple scalar constraints.

c - Search Strategies

As search implementation, we use different variables families (branching on edges or directly branching on cost) and we focus on ordering the variables to be assigned. We test :

1. a static lexicographic strategy assignment on edges (E_{ij}), meaning that variables are simply not sorted;
2. a random strategy on edges (E_{ij}) with max value first, which help us to have a median behavior on edges branching;
3. a dynamic variable ordering heuristic, called *dom over wdeg*, applied to weight variables (default strategy in our solver). This strategy consists in ordering the involved variables by increasing size of domain, combined with a weighted sum on the constraints they belong to. This strategy is supposed to well behave for a majority of problems [21];
4. a maximum value strategy on cost: as we have to maximize the $TotalWeight$ variable, we use a standard max domain value strategy first on $Weights$ variables: by propagation, assigning first edges with biggest weights leads to connect edges with maximum numbers of pairs of reads mapping on both contigs;
5. a max-regret strategy: assigning first edge with biggest weight for variable with biggest difference between maximum and previous value in domain (aka regret, if not chosen). This last strategy yields usually good results on such "max $\sum$" optimization problem.

4 Experiments

Scaffold graphs used to run our experiments are coming from two sources. A first dataset, called *real instances*, has been build using the following pipeline:

1. We choose a reference genome, on a public database, for instance on the Nucleotide NCBI database[4]. Table 1 shows the selected genomes used to perform our experiments. They were chosen because of their various origins: a virus, an insect, a plant and a yeast; and their small size: two of them comes from organelles, a mitochondrion and a chloroplast, which have small genomes.
2. We simulate paired-end reads, using a tool like `wgsim` [22]. The chosen parameters are an insert size of 500bp and a read length L of 100bp.
3. We assemble those simulated reads using a *de novo* assembly tool, based on a de Bruijn graph efficient representation. This tool is `minia` [23], and was used with a size $k = 30$ for the k-mers stored in the de Bruijn graph.
4. We map the reads on the contigs, using `bwa` [24]. This mapping tool was chosen according to results obtained in [25], a survey on scaffolding tools.
5. We generate the scaffold graph from the mapping file. Statistics on number of vertices and edges in produced scaffold graphs can be viewed on Table 2.

[4] `http://www.ncbi.nlm.nih.gov/`

Table 1. Dataset of real instances.

Species (Alias)	Size (bp)	Type	Accession number
Zaire ebolavirus (`ebola`)	18959	Complete genome	NC_002549.1
Danaus plexippus (`monarch`)	15314	Mitochondrion	NC_021452.1
Oryza sativa Japonica (`rice`)	134525	Chloroplast	X15901.1
Saccharomyces cerevisiae (`sacchr3`)	316613	Chromosome 3	X59720.2
Saccharomyces cerevisiae (`sacchr12`)	1078177	Chromosome 12	NC_001144.5

Since it is quite complicated to find real instances through fully meeting needed parameters, especially size of the scaffold graphs, and to perform average analysis on classes with only one element, a second dataset of generated scaffold graphs was used to complete the size gap between our real instances: the `rice` scaffold graph counts 84 contigs, but `sacchr3` counts 296, and `sacchr12` counts 889. Instances were generated by the tool `Grimm` [26]. The parameters used to generate this dataset were chosen to be similar to the ones observed on real instances. However we are conscious that they do not exactly meet the reality. A set of thirty instances were generated by pair of parameters $(\#vertices, \#edges)$: these pairs come from real instances parameters, completed by $\{(200, 300), (300, 450), (400, 600), (500, 750)\}$.

We run experiments on a MacBook Pro with Intel i7 2.8Ghz processor and 4 Go RAM. First we evaluate each of the previously described strategies. Then, using the best strategy, we solve the real instances and discuss the convenience of such modeling. Finally, we compare obtained results to previous ILP approach.

5 Results

Testing search strategies.

Figure 2 shows the comparison between the different tested search strategies on a generated instance with 200 vertices. Scores shown on this figure is simply the total weight of the current solution. As expected, lexicographic and random strategies on edges do not perform well. This is due to the fact that scaffold graph have a very peculiar structure. They are sparse, and as one can see on Figure 1, the degree of vertices is quite small. For cost based strategies, surprisingly the standard max value performs very well contrary to max regret. We can explain this by the correlation between weight value (based on the number of bridging reads) and the probability that this will be a good link. Weight value are not randomly distributed but are extracted from partial information (reads) coming from a precise structure (the underlying connected structuration of the chromosome).

Let's remark that default dom/wdeg shows its classical robust good behavior without any knowledge !

In what follows, all experiments were performed using the max value strategy.

Results on real instances.

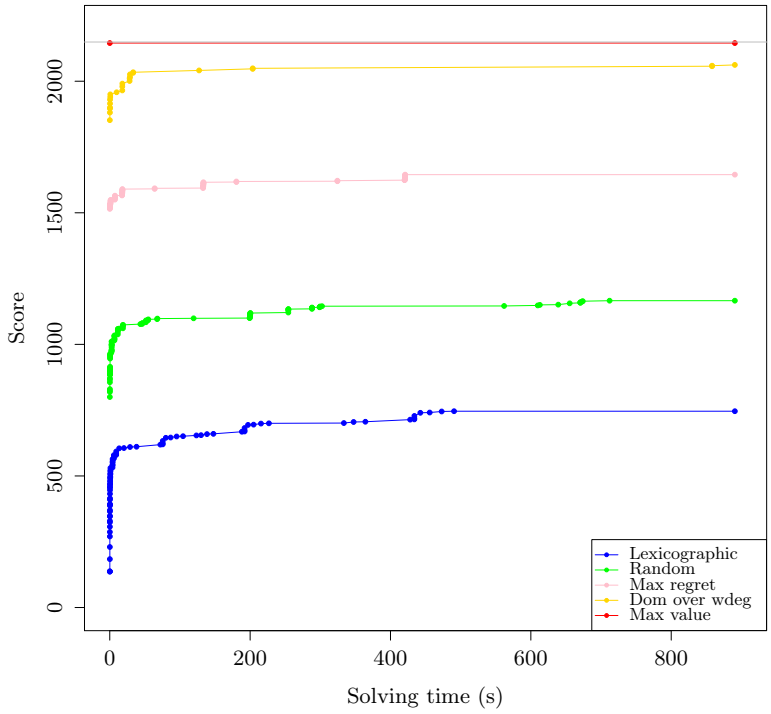

Fig. 2. Comparison of search strategies.

Table 2. Search for optimal solutions on real instances.

Instances	parameters				first sol			optimality		
name	nbContigs	nbEdges	σ_p	σ_c	value	searchNodes	time (s)	value	searchNodes	time (s)
monarch	14	19	5	0	520	16	0.013	520	16	0.013
ebola	17	26	3	1	793	20	0.026	793	20	0.026
ebola	17	26	4	0	707	42	0.040	707	56	0.123
rice	84	139	12	0	4377	106	0.091	4382	147k	177
sacch3	296	527	36	0	14845	406	0.48		> 1M	>3600
sacch12	889	1522	118	1	Out of Memory					

As one can observe in Table 2, we tested our program on several scaffold graphs produced from biological data. It is interesting to note that the first solution has already a high score and is found very quickly (less than one second with very few backtracks). Having a closer look to `ebola`, we noticed that scaffold graphs may contain what we call bad *contig jumps*, meaning there exists a contig a which should appear between two contigs b and c in the genome but such that the optimal solution does not contain $\{a, b, c\}$ in any scaffold. When the contig a is not included in the solution at its place in a path, we say that it is a *forgotten* contig. Necessarily, the length of a is small enough to be inserted in the gap between b and c, the latter being smaller than the insert size of the library. For instance, the ebola graph should ideally contain one unique linear scaffold representing the linear chromosome of ebola. Nevertheless, the optimal solution contains at least four scaffolds because it exists forgotten contigs which can be considered as scaffolds on their owns. On Figure 1, two small contigs appear without any bad consequence, namely 0-1 and 32-33. Indeed, the optimal path includes them. However, other cases are not so easy to solve: for instance the ambiguity between paths 31-30-16 and 13-12-16 leads to choose the latter one, only considering weights. But it is a hidden contig jump, and further examination of inter-contig distances should be included to disambiguate and include both contigs 12-13 and 30-31 in a same path of the solution. Same situation occurs for paths 4-5-3 and 10-11-4. Case of contig jump 14-15 is quite different: the sums of weights on side edges 6-15 and 14-17 is less that the weight of the "by-pass" edge 6-17. In such case, contig 14-15 is not included in the solution. In a nutshell, contigs 30-31, 4-5 and 14-15 are forgotten contigs, explaining the four scaffolds instead of only one expected. Here, we express the necessity to pre-process the graph to treat such contigs and we will consider it as a priority in our future works.

Comparison to previous ILP approach In [11], we proposed an ILP based incremental approach, which was able to heuristically handle large instances. The underlying idea is very simple and consists in optimizing the score, under constraints on degrees, and use an external treatment to forbid detected cycles. We ran this tool on our real dataset. The main difference with actual model is that we considered only cases with paths, and systematically forbade cycles. Thus, the actual model is more expressive, since it allows a given number of cycles.

Table 3. Comparison with previous ILP approach.

Instances	**first sol**		**optimality**		**ILP [11]**	
name	**value**	**time (s)**	**value**	**time (s)**	**value**	**time (s)**
monarch	520	0.013	520	0.013	507	0.00025
ebola	793	0.026	793	0.026	776	0.00028
rice	4377	0.091	4382	177	4320	0.00036
sacch3	14845	0.48		>3600	14616	0.0071

Table 3 shows comparison of solving time and scores between Choco-graph and ILP heuristic. What is noticeable is that, as expected, computation time stays very low when the size of instances increases, and that ILP does not provide an optimal score on these instances. More surprisingly, the score given by first solution is better than ILP score, meaning that efforts made on modelization are somehow rewarded.

6 Conclusion and future works

Classically, modeling real problems brings a gap between customers and modelers. Here, a first gap exist between biologist researchers and bioinformatic researchers to express biological problem into graph modeling problem. Constraint Programming provides a natural declarative way to express constraints on a given problem without worsen the modeling gap between combinatorial problem description and combinatorial solvers resolution, contrary to what happened with previous attempts using SMT, SAT or ILP models. Moreover, the Graph variable development are absolutely convenient to the modeling of combinatorial problems on graphs and we present here a typical example where its usefulness is demonstrated. Although the underlying problem is $\mathcal{NP}$-hard, and there is no hope to quickly solve very large instances in a reasonable time, we could improve solving time by introducing more constraints to help propagation efficiency. For instance, by considering contig lengths and expected lengths of chromosome, if it is known, we could set the minimum length of a cycle or a path. Or, considering that we could not guarantee that all linking information are provided by the original scaffold graph, we could consider only a maximum number of cycles, but allows the number of paths to be itself a variable. Finally, it would be interesting to embed the solver in an interactive tool which allows an expert user to solve and visualize on a given instance.

Acknowledgements

We want to thanks Valentin Pollet for an early version of the Choco graph code.

References

1. Daniel H. Huson, Knut Reinert, and Eugene W. Myers. The greedy path-merging algorithm for contig scaffolding. *Journal of the ACM (JACM)*, 49(5):603–615, 2002.
2. Annie Chateau and Rodolphe Giroudeau. A complexity and approximation framework for the maximization scaffolding problem. *Theor. Comput. Sci.*, 595:92–106, 2015.
3. Mihai Pop, Daniel S Kosack, and Steven L. Salzberg. Hierarchical scaffolding with bambus. *Genome research*, 14(1):149–159, 2004.
4. Marten Boetzer, Christiaan V. Henkel, Hans J. Jansen, Derek Butler, and Walter Pirovano. Scaffolding pre-assembled contigs using sspace. *Bioinformatics*, 27(4):578–579, 2011.

5. René L Warren, Granger G Sutton, Steven JM Jones, and Robert A Holt. Assembling millions of short dna sequences using ssake. *Bioinformatics*, 23(4):500–501, 2007.
6. Sergey Koren, Todd J. Treangen, and Mihai Pop. Bambus 2: scaffolding metagenomes. *Bioinformatics*, 27(21):2964–2971, 2011.
7. Song Gao, Wing-Kin Sung, and Niranjan Nagarajan. Opera: reconstructing optimal genomic scaffolds with high-throughput paired-end sequences. *Journal of Computational Biology*, 18(11):1681–1691, 2011.
8. Nilgun Donmez and Michael Brudno. Scarpa: scaffolding reads with practical algorithms. *Bioinformatics*, 29(4):428–434, 2013.
9. Alexey A Gritsenko, Jurgen F. Nijkamp, Marcel J. T. Reinders, and Dick de Ridder. Grass: a generic algorithm for scaffolding next-generation sequencing assemblies. *Bioinformatics*, 28(11):1429–1437, 2012.
10. Leena Salmela, Veli Mäkinen, Niko Välimäki, Johannes Ylinen, and Esko Ukkonen. Fast scaffolding with small independent mixed integer programs. *Bioinformatics*, 27(23):3259–3265, 2011.
11. Nicolas Briot, Annie Chateau, Rémi Coletta, Simon De Givry, Philippe Leleux, and Thomas Schiex. An Integer Linear Programming Approach for Genome Scaffolding. In *Workshop Constraints in Bioinformatics*, 2014.
12. Jean-Guillaume Fages, Narendra Jussien, and Xavier Lorca a nd Charles Prud'homme. Choco3: an open source java constraint programming library, 2013.
13. Mathias Weller, Annie Chateau, Clément Dallard, and Rodolphe Giroudeau. Scaffolding problems revisited: Complexity, approximation and fixed parameter tractable algorithms, and some specials cases. *submited to Algorithmica*, 2016.
14. Mathias Weller, Annie Chateau, and Rodolphe Giroudeau. Exact approaches for scaffolding. *BMC Bioinformatics*, 16(Suppl 14):S2, 2015.
15. Jean-Charles Régin. Modeling problems in constraint programming. *Tutorial CP*, 4, 2004.
16. Grégoire Dooms, Yves Deville, and Pierre Dupont. CP (graph): Introducing a graph computation domain in constraint programming. In *Principles and Practice of Constraint Programming-CP 2005*, pages 211–225. Springer, 2005.
17. Carmen Gervet. Interval propagation to reason about sets: definition and implementation of a practical language. *Constraints*, pages 191–244, 1997.
18. Jean-Charles Regin. Global constraints: A survey. In Pascal van Hentenryck and Michela Milano, editors, *Hybrid Optimization*, volume 45 of *Springer Optimization and Its Applications*, pages 63–134. Springer New York, 2011.
19. Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.
20. Luis Quesada, Peter Van Roy, Yves Deville, and Raphaël Collet. Using dominators for solving constrained path problems. In *Practical Aspects of Declarative Languages, 8th International Symposium, PADL 2006, Charleston, SC, USA, January 9-10, 2006, Proceedings*, pages 73–87, 2006.
21. Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. In *In Proceedings of ECAI'04*. Citeseer, 2004.
22. Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor T. Marth, Gonçalo R. Abecasis, and Richard Durbin. The sequence alignment/map format and samtools. *Bioinformatics*, 25(16):2078–2079, 2009.

23. Rayan Chikhi and Guillaume Rizk. Space-efficient and exact de bruijn graph representation based on a bloom filter. In *Algorithms in Bioinformatics - 12th International Workshop, WABI 2012, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*, pages 236–248, 2012.
24. Heng Li and Richard Durbin. Fast and accurate long-read alignment with burrows-wheeler transform. *Bioinformatics*, 26(5):589–595, 2010.
25. Martin Hunt, Chris Newbold, Matthew Berriman., and Thomas D. Otto. A comprehensive evaluation of assembly scaffolding tools. *Genome Biology*, 15(3):R42, 2014.
26. Adel Ferdjoukh, Eric Bourreau, Annie Chateau, and Clémentine Nebut. A Model-Driven Approach for the Generation of Relevant and Realistic Test Data. In *SEKE 2016*, page to appear, 2016.