FEDERAL UNIVERSITY OF FRONTEIRA SUL
CAMPUS CHAPECÓ
COMPUTER SCIENCE UNDERGRADUATE COURSE

LAURA MESQUITA BRUEL

A GRAPH-BASED INTEGER LINEAR PROGRAMMING MODEL FOR THE UNBALANCED MINIMUM COMMON STRING PARTITION PROBLEM

CHAPECÓ
2026

Bruel, Laura Mesquita

A GRAPH-BASED INTEGER LINEAR PROGRAMMING MODEL FOR THE UNBALANCED MINIMUM COMMON STRING PARTITION PROBLEM / Laura Mesquita Bruel – 2026.

13 f.

Advisor: Prof. Dr. Andrei de Almeida Sampaio Braga

Final Undergraduate Work (Undergraduate) – Federal University of Fronteira Sul, Computer Science Undergraduate Course, Chapecó, SC, 2026.

1. Mininum Common String Partition Problem 2. String Partition 3. Integer Linear Programming 4. Combinatorial Optimization I. Braga, Dr. Andrei de Almeida Sampaio, orient. II. Federal University of Fronteira Sul. III. Título.

LAURA MESQUITA BRUEL

A GRAPH-BASED INTEGER LINEAR PROGRAMMING MODEL FOR THE UNBALANCED MINIMUM COMMON STRING PARTITION PROBLEM

Final Undergraduate Work submitted to the Federal University of Fronteira Sul in partial fullfillment of the requirements for the degree of Bachelor in Computer Science.

Advisor: Prof. Dr. Andrei de Almeida Sampaio Braga

This Final Undergraduate Work was evaluated and approved by the examiners on: 25/06/2026.

EXAMINERS

Prof. Dr. Andrei de Almeida Sampaio Braga - UFFS

Prof. Me. Alesom Zorzi - Unochapecó

Prof. Dr. Samuel da Silva Feitosa - UFFS

A Graph-Based Integer Linear Programming Model for the Unbalanced Minimum Common String Partition Problem

Laura Mesquita Bruel
Federal University of Fronteira Sul
SC-484, Km 02 - Fronteira Sul, Chapecó - SC, 89815-899
laura.bruel@estudante.uffs.edu.br

Andrei de Almeida Sampaio Braga
Federal University of Fronteira Sul
SC-484, Km 02 - Fronteira Sul, Chapecó - SC, 89815-899
andrei.braga@uffs.edu.br

ABSTRACT

The Minimum Common String Partition Problem (MCSP) consists of finding a minimum common partition between two related strings. This problem has applications in Computational Biology, wherein a string can represent a genome. The MCSP and its variants, which consider relevant information from genomes, are NP-hard and have already been addressed using heuristics and integer linear programming (ILP). There are ILP models that have been successfully applied to the MCSP and adaptations of these models to solve its variants have shown great results. Motivated by these results, in this paper, a Graph-Based ILP model to the MCSP was adapted to solve the Unbalanced Minimum Common String Partition Problem (UMCSP). The experimental results show that the adapted model can be similar or outperform existing ILP models to the UMCSP.

KEYWORDS. Minimum Common String Partition Problem. Integer Linear Programming. Combinatorial Optimization.

1. Introduction

String comparison problems are known for their application in Computational Biology, since genomes can be represented by strings [Goldstein et al., 2005]. In that study field, Genome Rearrangement Distance Problems can be described as, given two genomes, to find the least amount of operations that can transform the first genome into the second. This type of problem allows to estimate the evolutionary distance between two individuals [Swenson et al., 2008]. Adding more characteristics to these genomes results in more biologically relevant information and leads to more accurate estimates of evolutionary distances between genomes [Romeiro et al., 2024]. Therefore, different rearrangement problems can be defined based on the characteristics of the input genomes and the operations under consideration.

The Minimum Common String Partition Problem (MCSP) and its variants are NP-hard [Goldstein et al., 2005] and frequently discussed rearrangement problems throughout literature. They have been approached several times using heuristics because of their low computational cost, although such algorithms do not guarantee an optimal solution. However, it is possible to address these problems using integer linear programming (ILP), which guarantees an optimal solution, despite its high computational cost.

There are ILP models that have been successfully applied to the MCSP [Blum et al., 2015; Blum and Raidl, 2015; Ferdous and Rahman, 2015]. Regarding MCSP variants, Blum [2024] was

the first one to adapt an ILP formulation for the MCSP in order to solve a variant of the problem. Romeiro et al. [2024] adapted the two well-established ILP formulations in the literature, proposed by Blum et al. [2015] and Blum and Raidl [2015], to solve four MCSP variants. In both papers, the adaptations were simple and provided good results. Given the relevance of the problem, strategies that guarantee an optimal solution are important in the field of Computational Biology. Nonetheless, approaches to the MCSP and its variants using ILP can still be further explored. There is room for research on other ILP models for the MCSP, adaptations of ILP models that solve the MCSP to address its variants, and ILP models designed specifically for MCSP variants.

There is a third ILP formulation for the MCSP that has not yet been adapted to solve any variant of the problem. In that regard, considering the results provided by previous adaptations, this paper adapts the ILP formulation proposed by Ferdous and Rahman [2015] in order to add new constraints to the model that allow it to approach the characteristics of a variant of the MCSP. Furthermore, given an experimental analysis, it investigates whether adapting a different ILP formulation for the MCSP to solve a variant of this problem could yield good results, particularly with regard to finding a guaranteed optimal solution to the problem.

1.1. Main Contributions

The present paper presents a new ILP model for the Unbalanced Minimum Common String Partition Problem (UMCSP), a variant of the MCSP. This ILP model uses Common Graphs, a characteristic that differs from other models described across literature. The proposed model was able to outperform an existing ILP model based on Common Blocks and, considering model optimization execution time and quality of the obtained solution, the model has a similar performance to an existing ILP model based on Common Substrings. Furthermore, the ideas used in this paper – namely, adapting an ILP model for the MCSP to a variant of this problem – can be explored to formulate ILP models for other variants of the MCSP.

The remainder of this paper is organized as follows. Section 2 holds a literature review for related work and Section 3 explains the theoretical background for this study. A Graph-Based ILP model for the UMCSP is presented in Section 4. Section 5 presents the experimental results and their analysis. Finally, Section 6 draws conclusions and directions for future research.

2. Related Work

The Unbalanced Minimum Common String Partition Problem (UMCSP) is NP-hard [Goldstein et al., 2005] and a combinatorial optimization problem, due to its nature of finding an optimal solution from a finite set of possibilities. These characteristics allow its solutions to be addressed using heuristics or integer linear programming (ILP). This section describes studies that explores these methods to solve UMCSP or a variant of the problem.

Ferdous and Rahman [2017] introduced a common substring graph (CSG) mapping design to the Minimum Common String Partition Problem (MCSP) to solve it using an Ant Colony Optimization technique called MAX-MIN Ant System (MMAS). They compared the results obtained by MMAS with those obtained by a greedy heuristic. The MMAS found better solutions in 93% of the tests run using random DNA sequence and was able to improve 10 out of 15 cases run with real gene sequence in 5% significance level. They also conducted experiments with and without applying dynamic heuristic to MMAS and found a positive difference in 90% of cases, which depicts the improvement using dynamic heuristic. Similarly, Rodrigues et al. [2023] also mapped MCSP to a CSG in order to reduce it to a permutation problem and apply optimization algorithms, in particular the Particle Swarm Optimization algorithm (PSO). They approached MCSP using PSO, a greedy heuristic and a combination heuristic. The results indicate that, depending on the parameters and instances, PSO may take longer to complete its execution, however, it is substantially faster than

the other algorithms for larger instances. It was also shown that the CSG representation was able to improve the results of the heuristics. Therefore, much of the quality presented on the solutions obtained by the algorithms can be attributed to the CSG representation.

Blum [2024] addressed the UMCSP using the combinatorial optimization algorithm Construct, Merge, Solve and Adapt (CMSA). On each iteration of CMSA, (1) solutions to the problem instances are generated probabilistically, (2) the solution components found are added to an incumbent subinstance, (3) exact solvers are used to solve the incumbent subinstance, and (4) mechanisms are used to remove useless solution components. Given the third iteration step, an ILP model that solves MCSP [Blum et al., 2015] was adapted to approach UMCSP. The authors compared the results obtained by CMSA, a greedy heuristic and the usage of CPLEX [Cplex, 2009]. CMSA has a better performance than CPLEX for larger instances and than the greedy heuristic for every instance; it also provided viable solutions to every instance.

Dissimilar to the papers above, the solution proposed in this work does not use heuristics. However, similar to Ferdous and Rahman [2017] and Rodrigues et al. [2023], this study uses a graph to attempt to solve UMCSP, and, like Blum [2024], it adapts an existing ILP model for MCSP with the goal of solving a variant of the problem.

Considering ILP approaches to MCSP, Blum and Raidl [2015] presented an ILP model based on Common Substrings (CS) to solve the problem and proved that the model's linear relaxations are equally strong as the ones showed on the ILP model based on Common Blocks (CB). The authors compared the two models, and the results favor CS over CB, since the gaps obtained by CS are smaller and it finds the first viable solution, on average, using 0,7% of the time required to execute CB. Furthermore, Ferdous and Rahman [2015] developed an ILP formulation to MCSP based on a CSG mapping of the problem described in their previous work [Ferdous and Rahman, 2017]. The formulation is precise, effective, and provides excellent results. The results showed that their ILP formulation outperforms a greedy heuristic at every stage of the process and finds better viable solutions faster.

Romeiro et al. [2024] analyzed the adaptation of two ILP models – CS and CB – to solve four variants of MCSP. The adaptations made to the models added new restrictions to support unique characteristics of each variant, and different percentages of conservative operations were used to create the tests. The results showed that for both models, tests with 80% and 100% conservative operations run faster than those with 60%, suggesting that exclusive blocks, a concept introduced in the adaptation of the models to solve UMCSP, are computationally more expensive than common blocks. It was also possible to conclude that the CS model is better than the CB in all scenarios, running faster and achieving smaller gaps. Furthermore, the adapted models solved the tested instances with high quality considering the chosen execution time, and, in some cases, were able to find the optimal solution.

The present paper proposes an ILP formulation, as is done in the work above. This study uses the formulation proposed by Ferdous and Rahman [2015], and, similarly to Romeiro et al. [2024], which adapted the CS and CB-based models, adapts the CSG-based formulation so that it can solve a variant of MCSP. Moreover, the results of this work are based on how well the adapted formulation applied to the variant performs relative to the original formulation applied to MCSP; however, the CSG-based ILP formulation is applied to the UMCSP variant of MCSP, which adds complex characteristics to the problem, such as instances that are not guaranteed to be related.

3. Theoretical Background

This section describes notations and definitions required to understand the integer linear programming formulation used in this paper to solve the Unbalanced Minimum Common String Partition Problem.

3.1. Minimum and Unbalanced Minimum Common String Partition Problems

A *partition* of a string S_1 is a sequence $P = (p_1, p_2, \ldots, p_m)$ whose concatenation is equal to S_1. The substrings p_i are called *blocks* of P. Given a partition P of a string S_1 and a partition Q of a string S_2, $\pi = \langle P, Q \rangle$ is a *common partition* of S_1 and S_2, if Q is a permutation of P. The size of a common partition $\pi = \langle P, Q \rangle$ is $|\langle P, Q \rangle|$ [Rodrigues et al., 2023]. The number of times a character r appears in a string S_1 denotes the *occurrence* of r in S_1 [Romeiro et al., 2024]. Two strings are called *related* if each character has the same occurrence in both strings. Therefore, the Minimum Common String Partition Problem (MCSP) can be described as follows: given two related strings S_1 and S_2, find the common partition of S_1 and S_2 with the least amount of blocks [Goldstein et al., 2005].

The Unbalanced Minimum Common String Partition Problem (UMCSP) is a generalization of MCSP in which there is no guarantee that the input strings are related. Hence, the input strings can have different lenghts. Given a partition P of a string S_1, a partition Q of a string S_2, $P' \subseteq P$, and $Q' \subseteq Q$, with Q' being a permutation of P'. Let X represent the set of characters in S_1 that do not exist in P' and Y represent the set of characters in S_2 that do not exist in Q', with $X \cap Y = \emptyset$. X and Y denote the set of *abundant characters* in S_1 and S_2, respectively. $\pi = \langle P, Q, P', Q' \rangle$ is a *common partition* of S_1 and S_2 [Romeiro et al., 2024]. Therefore, the UMCSP can be described as follows: given two strings S_1 and S_2, find the common partition of S_1 and S_2 in which $|P' + X + Y|$ is minimum.

3.2. Graphs

A *graph* is a mathematical structure used to model objects [Diestel, 2010]. A graph G is a pair (V, E) consisting of a set of *vertices* V and a set of *edges* E. The set E is a subset of $V \times V$ (unordered pairs of V), since each edge is represented by a set $\{v_i, v_j\}$ of vertices of G. If $\{v_i, v_j\}$ is an edge, v_i and v_j are *neighbors* [Feofiloff et al., 2011]. According to Bollobas [1998], given a graph $G = (V, E)$, the *neighborhood* of a vertex $v \in V$ is the set of all the neighboring vertices of v. Moreover, a *path* of length n in G is a sequence of $n + 1$ distinct vertices such that for every pair of consecutive vertices v_k, v_{k+1}, $\{v_k, v_{k+1}\} \in E$.

If the edges of a graph $G = (V, E)$ are ordered pairs of vertices, that is, $(v_i, v_j) \neq (v_j, v_i)$, G is *directed graph*. An ordered pair (v_i, v_j) is a directed edge from v_i to v_j [Bollobas, 1998]. The concepts defined for simple graphs can easily be translated into directed graphs. Thus, given a directed graph $G = (V, E)$, the *outgoing neighborhood* of a vertex $v \in V$, denoted by $N^+_{(v)}$, is the set of all vertices with directed edges leaving v, that is, all the outgoing neighbors of v. The *incoming neighborhood* of a vertex $v \in V$, denoted by $N^-_{(v)}$, is the set of all vertices with directed edges entering v, that is, all the incoming neighbors of v. Furthermore, a path of length n in G is a sequence of distinct vertices such that for every pair of consecutive vertices $v_k, v_{k+1}, (v_k, v_{k+1}) \in E$.

3.3. Integer Linear Programming Model for the MCSP

Integer Linear Programming (ILP) is a type of optimization model with the objective to maximize or minimize a linear function where all the variables must only take integer values. ILP can be described as a mathematical model with the following components: decision variables, objective function and constraints. *Decision variables* represent the decisions to be made. The *objective function* express the function to be maximized or minimized. *Constraints* limit the values that can be taken by the decision variables. A *feasible solution* to an ILP model is an assignment of values to the variables such that all constraints are satisfied.

Ferdous and Rahman [2015] proposed an ILP formulation for the MCSP based on their graph mapping design of the problem [Ferdous and Rahman, 2017]. Their formulation uses the following concepts. A *block* $B = ([id, i, j]), 0 \leq i \leq j < |S|$ of a string S is a data structure

with three attributes: id is an identifier of S, i is the starting position of B in S and j is the ending position of B in S. The substring delineated by $B = ([id, i, j])$ is called $substring([id, i, j])$. A block B *matches* a block B', regardless of the value attributed to id, if both blocks represent the same substring. Given a block B and a block list L_b, $macthList(B, L_b)$ defines the list of blocks of L_b that matches B. Given two input strings X and Y, a *common substring graph* (CSG) of X is defined by the directed graph $G^X_{sc} = (V, E)$. Each vertex represents a position of X, that is, for each $v \in V, v \in \{0, 1, \ldots, |X| - 1\}$. Two vertices $v_i \leq v_j$ are connected by an edge $(v_i, v_j) \in E$ if the substring induced by the block $[id(X), v_i, v_j]$ matches some substring of Y. Therefore, every edge in E corresponds to a block that satisfies the above condition. If $X = ebacded$ and $Y = bdcdebedd$, Figure 1 represents the CSG of X.

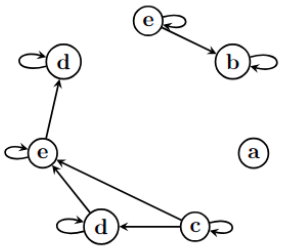

Figure 1: CSG of X

The formulation suggested by Ferdous and Rahman [2015] was used as the basis to the formulation proposed on this paper.

4. Integer Linear Programming Model

Ferdous and Rahman [2015] proposed an ILP formulation for the Minimum Common String Partition Problem (MCSP) based on a graph mapping design presented by Ferdous and Rahman [2017]. Their ILP formulation requires the understanding of some notations and definitions that are also used on the adaptation showed in this paper to solve the Unbalanced Minimum Common String Partition Problem (UMCSP).

Considering the notations given on Section 3, in the approach of UMCSP, there is no guarantee the input strings are related, that means, have the same character occurrence on both strings. Therefore, it is possible that exclusive parts appear on an input string, which introduces the concept of *exclusive blocks* to the formulation [Romeiro et al., 2024].

Let A determinate the set of abundant characters of an input string X and $G_{e1} = (V, E)$ determinate the exclusive blocks graph of X. V is the vertex set where each $v \in V$ represents a character on X and E is the edge set. The edge (v_i, v_j) exists only if for every vertex $v_k, i \leq k \leq j, v_k \in A$. It occurs because every exclusive block stands for an exclusive part of an string S, which, by definition, are made only of abundant characters in S [Romeiro et al., 2024].

Since, considering the UMCSP, the input strings are over the same alphabet, there may be mapped common blocks that are also made of abundant characters. For example, if $X = ebacded$ and $Y = bdcdebedd$, $G_x = (V, E)$ determines X's CSG and $G_y = (V, E)$ determines Y's. X's abundant characters are $A_x = \{a\}$ and Y's abundant characters are $A_y = \{b, d, d\}$. The block $b = [X, 2, 2]$ stands for the substring a, that has no matching block on Y. So, if B_y is the list of all mapped blocks on Y, $matchList([X, 2, 2], B_y) = \{\}$ and there is no edge (v_2, v_2) on G_x. The block $b = [X, 4, 4]$, however, stands for the substring d, which has matching blocks on Y. So, $matchList([X, 4, 4], B_y) = \{[Y, 1, 1], [Y, 3, 3], [Y, 7, 7], [Y, 8, 8]\}$ and every block on $matchList([X, 4, 4], B_y)$ is an edge on their string's CSG. d is an abundant character on Y and

may be an exclusive block in an optimal solution, even if there is a corresponding common block in X.

The sets V and E of an *exclusive block graph* (EBG) of an input string X are denoted by V_{e1} and E_{e1}. If $X = ebacded$ and $Y = bdcdebedd$, the EBG of X has just one edge (v_2, v_2). That is because a is the only abundant character in X. Also, the binary variables $x_{t_e1}, t_{e1} \in E_{e1}$ and $y_{t_e2}, t_{e2} \in E_{e2}$, indicate whether an exclusive block has been chosen.

Given that there are four input graphs, it is necessary to ensure that each character belongs to only one common or exclusive block. This implies adapting the constraints in the original formulation [Ferdous and Rahman, 2015] that guarantee that a string S must be factored into non-overlapping blocks. In addition, for every edge sum constraints on the original formulation, the edges of each EBG need to be considered.

With the above settings, the ILP formulation for the UMCSP was developed as follows:

$$min \sum_{t_1 \in E_1} x_{t_1} + \sum_{t_e1 \in E_e1} x_{t_e1} + \sum_{t_e2 \in E_e2} y_{t_e2} \tag{1}$$

$$s.t. \sum_{t_1 \in E_1} x_{t_1} = \sum_{t_2 \in E_2} y_{t_2} \tag{2}$$

$$\sum_{t_1 \in E_1 \mid t_1=(v_i,v_j),\, i \le k \le j} x_{t1} + \sum_{t_{e1} \in E_{e1} \mid t_{e1}=(v_i,v_j),\, i \le k \le j} x_{t_{e1}} = 1 \qquad \forall v_k \in \{0, \ldots, n_1 - 1\} \tag{3}$$

$$\sum_{t_2 \in E_2 \mid t_2=(v_i,v_j),\, i \le k \le j} y_{t_2} + \sum_{t_{e2} \in E_{e2} \mid t_{e2}=(v_i,v_j),\, i \le k \le j} y_{t_{e2}} = 1 \qquad \forall v_k \in \{0, \ldots, n_2 - 1\} \tag{4}$$

$$\sum_{t_1 \in \delta_1(0)^+} x_{t1} + \sum_{t_{e1} \in \delta_{e1}(0)^+} x_{t_{e1}} = 1 \tag{5}$$

$$\sum_{t_1 \in \delta_1(v)^-} x_{t1} + \sum_{t_{e1} \in \delta_{e1}(v)^-} x_{t_{e1}} = \sum_{t_1 \in \delta_1(v+1)^+} x_{t1} + \sum_{t_{e1} \in \delta_{e1}(v+1)^+} x_{t_{e1}} \qquad \forall v \in \{0, \ldots, n_1 - 1\} \tag{6}$$

$$\sum_{t_2 \in \delta_2(0)^+} y_{t2} + \sum_{t_{e2} \in \delta_{e2}(0)^+} y_{t_{e2}} = 1 \tag{7}$$

$$\sum_{t_2 \in \delta_2(v)^-} y_{t2} + \sum_{t_{e2} \in \delta_{e2}(v)^-} y_{t_{e2}} = \sum_{t2 \in \delta_2(v+1)^+} y_{t2} + \sum_{t_{e2} \in \delta_{e2}(v+1)^+} y_{t_{e2}} \qquad \forall v \in \{0, \ldots, n_2 - 1\} \tag{8}$$

$$\sum_{b_1 \in matchList(E_1, t_1)} x_{b1} = \sum_{b_2 \in matchList(E_2, t_1)} y_{b2} \qquad \forall t_1 \in E_1 \tag{9}$$

$$x_{t_1}, y_{t_2}, x_{t_{e1}}, y_{t_{e2}} \in \{0, 1\} \tag{10}$$

The objective function (1) is to minimize the partition size calculated. The restriction (2) ensures that the two partitions in two the CSGs must be of the same size (i.e., have the same number of blocks). Constraints (3) imply that every character in X belongs to one and only one common block or exclusive block and constraints (4) hold the same for Y. Constraints (5) and (6) together for string X, and similarly constraints (7) and (8) for Y, ensure that a unit flow leaves the source

(vertex 0) and enters the sink (vertex $n-1$); that is, the strings are partitioned into non-overlapping blocks. Furthermore, after partitioning the input strings, there are four sets of blocks: two are sets of exclusive blocks and two are sets of common blocks. It must be ensured that there is a one-to-one matching between the sets of common blocks, since each block of a partition represents a substring of the original string. Therefore, to ensure one-to-one matching, for each selected block in the set of edges E_1, there must be one, and only one, corresponding block selected in the second set of edges E_2. The same must hold for the blocks selected from E_2 with respect to the blocks selected from E_1. It is not possible to ensure one-to-one matching between exclusive blocks. Constraints (9) ensure there is an one-to-one matching between the two sets of common blocks by securing that for each selected block, the number of selected blocks in E_1 equals the number of selected blocks in E_2.

5. Results

This section presents the results obtained by the comparison of the Graph-Based ILP model based on Common Graphs (CG) and two competitive formulations in the literature, an ILP model based on Common Blocks (CB) and another based on Common Strings (CS). The characteristics of the tests instances were described, followed by a presentation of the computational environment in which the experiments were conducted. Subsequently, the performance of the three models is analyzed in terms of quality of the obtained solution and computational time.

5.1. Test Instances

The tests were run using three different instance sets. The first set is made of 40 pairs of strings of the same size, created with the iteration of two parameters: length $n \in \{10, 30, 60, 100\}$ and alphabet size $|\Sigma| \in \{5, 10, 15, 20\}$. The second set is made of 40 pairs of strings of different lengths, created with the iteration of three parameters: length of the first string n_x, length of the second string n_y, $n_x, n_y \in \{10, 15, 30, 40, 60, 80, 100, 120\}$ and alphabet size $|\Sigma| \in \{5, 10, 15, 20\}$. In both sets, the alphabet size increases according to the length of the strings. The third set is made of 40 instances used by Romeiro et al. [2024], wherein length $n = 1000$ and alphabet size $|\Sigma| = 1000$. The instance sets were created using the instance generator proposed by Romeiro et al. [2024]. This generator requires a parameter called fraction of conservative operations c. Conservative operations alter the structure of chromosomes without changing the set of genes [Cleber V., 2007]. For all three sets, $c \in \{0.6, 0.8\}$; therefore, the strings are unbalanced because the occurrence of each character $r \in \Sigma$ is not the same in both strings of the pair. This holds even for pairs of the first and the third sets, which are made of strings of equal lengths. Furthermore, the pairs of strings of the second set are also unbalanced because the sizes of the two strings are different.

5.2. Computational Environment

The ILP model was implemented using Python [Python Core Team, 2019] 3.11.2, for its ease of implementation and its wide range of libraries. Library Numpy [Harris et al., 2020] version 2.4.4 was used to support development. The model was optimized with Gurobi [Gurobi Optimization, LLC, 2025] solver, since it has a free academic license and good performance. The ILP models CB and CS used for comparison were provided by Romeiro [2024] and implemented using Python and Gurobi. The tests were executed for the three models on a machine equipped with Intel(R) Xeon(R) E7-4850 processor at 2.00GHz, 125GB of RAM, running Ubuntu 18.04.6 (64-bit).

5.3. Results

Table 1 compares the results obtained by the ILP model based on Common Graphs (CG) and the results obtained by the ILP model based on Common Blocks (CB) [Romeiro et al., 2024]. Table 2 compares the results obtained by CG and the results obtained by the ILP model based on

Common Substrings (CS) [Romeiro et al., 2024]. The average model optimization time, the average total execution time and the average gap obtained by the models were compared, grouped by the input string length n; for instances comprising pairs of strings with different lengths, n is the size of the largest string.

Table 1: Execution times and gaps comparing CB and CG

n	Average Execution Time (seconds)				Average Gap (%)	
	Model		Total			
	CB	CG	CB	CG	CB	CG
10	0.0058	0.0084	0.0086	0.0156	0.0000	0.0000
30	0.0120	0.0103	0.0279	0.0669	0.0000	0.0000
60	0.0169	0.0106	0.0567	0.3009	0.0000	0.0000
100	0.0242	0.0303	0.1172	1.1807	0.0000	0.0000
15	0.0089	0.0132	0.0227	0.0343	0.0000	0.0000
40	0.0208	0.0198	0.0430	0.1023	0.0000	0.0000
80	0.0570	0.0594	0.1258	0.5203	0.0000	0.0000
120	0.0795	0.0792	0.2172	1.6103	0.0000	0.0000
1000	3504.1720	2728.5951	5023.1951	4804.4354	20.9896	0.3338

The results on Table 1 show that CG and CB have similar execution times, although CB runs faster for smaller instances (strings where $n < 1000$) and CG has better average execution time for larger instances (strings where $n = 1000$). This could be explained by the time each model uses to build the necessary structures. Considering the smaller instances, the average time CG spent building structures was 0.45 seconds and CB had an average time for building structures of 0.0492 seconds. However, both models had a similar average model optimization time: 0.0289 seconds for CG and 0.0281 seconds for CB. The two models were able to solve all the smaller instances. Given the set of larger instances, CG model was able to find the optimal solution for 30 instances and CB was able to solve 3 instances. For the 10 instances CG was not able to optimally solve, the biggest gap was 1.8957% and the average gap was 1.3351%. For the 37 instances CB was not able to optimally solve, the biggest gap was 51.2255% and the average gap was 20.9896%. Within the subset of instances both models were not able to optimally solve, CG gave a better solution value in 9 out of 10 cases and both models found the same solution value in 1 case. The biggest difference between the solution values obtained was 484, with CG finding the better solution, and the average difference was 234. Both models found the same lower bound for 8 out of 10 cases and CG provided a better bound for 2 cases. The biggest difference between the provided lower bounds was 49 and the average difference was 5. For that subset, the biggest gap found using CG was 1.8957% and the average gap was 1.3351%. The biggest gap found using CB was 50.36% and the average gap was 29.9064%.

The results on Table 2 shows that CS runs significantly faster than CG. For the smaller instances (strings where $n < 1000$), the average time CS used for building structures was 0.0489 seconds and the average model optimization time was 0.0166 seconds. Considering the larger in-

Table 2: Execution times and gaps comparing CS and CG

n	Average Execution Time (seconds)				Average Gap (%)	
	Model		Total			
	CS	CG	CS	CG	CS	CG
10	0.0054	0.0084	0.0095	0.0160	0.0000	0.0000
30	0.0090	0.0103	0.0297	0.0669	0.0000	0.0000
60	0.0093	0.0106	0.0531	0.3009	0.0000	0.0000
100	0.0102	0.0303	0.1014	1.1807	0.0000	0.0000
15	0.0090	0.0132	0.0235	0.0224	0.0000	0.0000
40	0.0145	0.0198	0.0402	0.1023	0.0000	0.0000
80	0.0405	0.0594	0.1083	0.5203	0.0000	0.0000
120	0.0352	0.0792	0.1587	1.6103	0.0000	0.0000
1000	2551.6719	2728.5951	3965.3118	4804.4354	0.3328	0.3338

stances (strings where $n = 1000$), the average time for structure creation was 1413.6398 seconds for CS and 2075.8403 seconds for CG; however, both had a similar average model optimization time: 2551.6719 seconds for CS and 2728.5951 seconds for CG. Both models found the optimal solution for all the smaller instances and CS could optimally solve 28 out of 40 larger instances. For the 12 instances CS was not able to optimally solve, the biggest gap was 2.7% and the average gap was 1.1095%. Within the subset of instances both models were not able to find the optimal solution, CS found a better solution value in 5 out of 10 cases, CG found a better solution value in 2 cases and they found the same solution value in 3 cases. The biggest difference between the solution values obtained was 3, with CS finding the better solution, and the average difference was 0.8. Both models found the same lower bound for 8 out of 10 cases, CS provided a better lower bound for 1 instance and CG gave a better lower bound for 1 case. The biggest difference between the provided lower bounds was 1 and the average difference was 0. For that subset, the biggest gap found using CG was 1.8957% and the average gap was 1.3351%. The biggest gap found using CS was 2.7% and the average gap was 1.1745%.

The results show that CG is able to consistently provide very good gaps for larger instances and has a satisfying average model optimization time. This means the formulation is efficient and performs better for larger instances. CG is able to outperform CB regarding average execution time and quality of the obtained solution for larger instances, but both models have a similar average execution time considering instances of all lengths. Moreover, CG has a similar performance to CS regarding quality of the obtained solution; however, CS has a significantly better average execution time than CG considering instances of all lengths. Thus, CG can be improved if the creation of structures is done more quickly, in order to improve the overall execution time.

6. Conclusion

The Minimum Common String Partition Problem (MCSP) and its variants are NP-hard string rearrangement distance problems known for their application in Computational Biology, wherein string comparison allows us to estimate the evolutionary distance between two individuals.

In this field, strings can be used to represent genomes. Adaptations of integer linear programming (ILP) models that solve the MCSP in order to approach a variant of the problem have already been proposed across the literature and shown positive results.

The present paper presents a Graph-Based ILP model that solves the Unbalanced Minimum Common String Partition Problem (UMCSP), a variant of the MCSP. The model is an adaptation of an existing ILP Model that solves the MCSP. In order to conduct an experimental analysis, tests were run using the new model and two competitive formulations in the literature, an ILP model based on Common Blocks (CB) and another based on Common Strings (CS). The results indicate that the Graph-Based ILP model, based on Common Graphs (CG), is effective and provides good results. The proposed formulation was able to outperform CB and has a similar output to CS. CG performs best when used for larger instances, considering it was able to find a optimal solution for most test cases and consistently finds a very good gap for tests that have not been solved optimally. However, the total execution time of CG could be improved. This can be made by optimizing the average structure construction time of the formulation, as it largely differs from the other models.

Therefore, research can be directed into investigating more efficient ways of creating the necessary structures needed for CG and comparing the proposed model using more diversified test instances. Contemplating the application of the UMCSP in Computational Biology, datasets from comparative genomics programs could be compiled for experimental analysis and comparison. Furthermore, the proposed models can be compared to heuristics, in order to obtain a more complete understanding of the different approaches to the UMCSP. Moreover, another direction can be described as exploring ideas similar to those presented in this study to formulate ILP models for other variants of the MCSP.

References

Blum, C. (2024). *Construct, merge, solve & adapt*. Springer, Cham, 2024 edition.

Blum, C., Lozano, J. A., and Davidson, P. (2015). Mathematical programming strategies for solving the minimum common string partition problem. *European Journal of Operational Research*, 242 (3):769–777.

Blum, C. and Raidl, G. R. (2015). Computational performance evaluation of two integer linear programming models for the minimum common string partition problem. *Optimization Letters*, 10(1):189–205.

Bollobas, B. (1998). *Modern Graph Theory*. Springer.

Cleber V., G. M. (2007). *Análise Algébrica de Problemas de Rearranjo em Genomas: Algoritmos e Complexidade*. PhD thesis, Universidade Estadual de Campinas.

Cplex, I. I. (2009). V12. 1: User's manual for cplex. *International Business Machines Corporation*, 46(53):157.

Diestel, R. (2010). *Graph Theory*. Springer.

Feofiloff, P., Kohayakawa, Y., and Wakabayashi, Y. (2011). *Teoria dos Grafos: Uma Introdução Sucinta.* Instituto de Matemática e Estatística da Universidade de São Paulo (IME-USP).

Ferdous, S. M. and Rahman, M. S. (2015). An integer programming formulation of the minimum common string partition problem. *PLOS ONE*, 10(7):e0130266.

Ferdous, S. M. and Rahman, M. S. (2017). Solving the minimum common string partition problem with the help of ants. *Mathematics in Computer Science*, 11(2):233–249.

Goldstein, Kolman, and Zheng (2005). Minimum common string partition problem: Hardness and approximations. *The Eletrical Journal of Combinatorics*, 12(R50):R50–R50.

Gurobi Optimization, LLC (2025). Gurobi Optimizer Reference Manual. URL `https://www.gurobi.com`. Accessed 11 november 2025.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

Python Core Team. *Python: A dynamic, open source programming language*. Python Software Foundation, 2019. URL `https://www.python.org/`. Accessed 11 november 2025.

Rodrigues, L. d. S., Siqueira, G., Alves, T. d. P., and Dias, Z. (2023). *Heurísticas para Partição Comum Mínima de Strings*. Bachelor's thesis, Universidade Estadual de Campinas.

Romeiro, Siqueira, G., and Dias, Z. (2024). *Formulações de PLI para Variantes do Problema de Partição de Strings*. Bachelor's thesis, Universidade Estadual de Campinas.

Romeiro, F. (2024). string-partition-experiments. `https://github.com/fmromeiro/string-partition-experiments`. Accessed 11 november 2025.

Swenson, K. M., Marron, M., Earnest-Deyoung, J. V., and Moret, B. M. E. (2008). Approximating the true evolutionary distance between two genomes. *J. Exp. Algorithmics*, 12:1–17.