An Integer Linear Programming and Biased Random-Key Genetic Algorithm for the Sequence-to-Graph Alignment Problem with Vertex Label Modification

Gabriel Pereira Falcão[1] **, Erick Rodrigues de Lemos Ribeiro**[1] **,**
Matheus Amorim Martins[1] **, Eduardo Theodoro Bogue**[1] **, Said Sadique Adi**[1]

[1] Universidade Federal de Mato Grosso do Sul – Campo Grande, MS – Brazil

{gabriel_falcao, erick_rodrigues, matheus.amorim,

eduardo.theodoro, said.sadique}@ufms.br

Abstract. *This paper addresses the Sequence-to-Graph Alignment Problem with Vertex Label Modification, where vertex labels may be altered, shifting the objective to minimizing the modification cost for an exact alignment. To tackle the computational complexity of this problem, we propose an exact Integer Linear Programming (ILP) formulation and a Biased Random-Key Genetic Algorithm (BRKGA) metaheuristic. Computational experiments reveal the complementarity between the methods: the ILP efficiently solves moderately sized instances to optimality, while the BRKGA excels in the most challenging scenarios (large, sparse graphs with long queries), improving the best-known upper bounds and reducing alignment costs by more than 50% relative to the ILP solutions.*

1. Introduction

In recent years, the field of computational biology has gained significant importance in science and public health due to advances in the area enabled, primarily, by the studies and applications of combinatorial optimization problems intrinsic to the complexity of biological systems, such as, for example, the genetic sequences of individuals of a species. Specifically, the widespread use of graph-based references (commonly referred to as sequence graphs, pangenome graphs, or variation graphs) is notable due to their greater expressiveness, compared to linear genome representations, in capturing extensive genetic variation across multiple individuals and entire populations [Dilthey et al. 2015, Dilthey et al. 2016], in addition to not suffering from reference allele bias [Dilthey et al. 2015, Garrison et al. 2018, Paten et al. 2017]. Thus, it is essential for researchers to be able to align arbitrary sequences to these graphs in order to extract information and correctly represent this data, a problem known as Sequence-to-Graph Alignment.

More formally, the Sequence-to-Graph Alignment problem considers a labeled directed graph $G = (V, E, \Sigma)$, where $\sigma : V \to \Sigma$ assigns to each vertex $v \in V$ a *character* over the alphabet Σ, and a query sequence $q \in \Sigma^m$ of length m. A walk $\pi = \langle v_0, v_1, \ldots, v_{m-1} \rangle$ in G *induces* the sequence $\sigma(v_0)\,\sigma(v_1) \cdots \sigma(v_{m-1})$. In the classical version of the problem, the goal is to minimize the edit distance between q and the sequence induced by π.

In this work, we study a variant proposed by [Amir et al. 2000], called the Sequence-to-Graph Alignment Problem with Vertex Label Modification (S2G-LM), in

which it is possible to modify vertex labels to obtain a perfect alignment. Formally, given a labeled directed graph $G = (V, E, \Sigma)$ and a query sequence $q \in \Sigma^m$, the problem consists of finding a walk $\pi = \langle v_0, \ldots, v_{m-1} \rangle$ that minimizes the substitution cost $f(\pi)$, defined as the number of mismatches between the induced sequence and q:

$$f(\pi) = \left| \{ i \in \{0, \ldots, m-1\} \mid \sigma(v_i) \neq q[i] \} \right|$$

Figure 1 displays an example of an S2G-LM instance, where the walk $\langle v_0, v_1, v_2, v_4 \rangle$ traverses the graph and is compared position by position with the sequence $q =$ ACGA. The first three positions match; the last one requires a label modification (T$\rightarrow$A), resulting in a cost of $f(\pi) = 1$.

This variant finds direct motivation in the polishing of genomic assemblies [Vaser et al. 2017], a step in which reliable reads (e.g., Illumina or PacBio HiFi) are used to correct errors in the labels of an assembly graph constructed from noisy reads. Formally, given a query read q and an assembly graph G with potentially erroneous labels, the goal is to find the minimum set $\mathcal{M}$ of label substitutions such that there is a walk in G inducing q, precisely the S2G-LM formulation, under the standard parsimony principle in error correction. Unlike standard read correction, where the errors are assumed to be in the query and the problem admits a polynomial-time solution [Jain et al. 2020], graph polishing places the errors within the graph itself, falling directly into the variant addressed in this study.

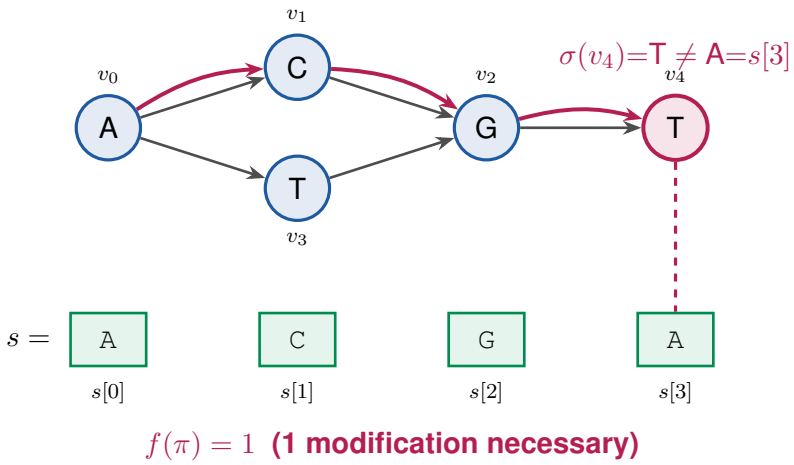


Figure 1. Example of an S2G-LM problem instance. The walk $\langle v_0, v_1, v_2, v_4 \rangle$ traverses the graph and is compared position by position with the query $q =$ ACGA, resulting in a cost of $f(\pi) = 1$.

As demonstrated by Jain et al. [Jain et al. 2020], this problem is $\mathcal{NP}$-complete for $|\Sigma| \geq 2$ even when considering only substitutions in the query sequence. Despite its relevance, to our knowledge, there are no exact or heuristic algorithms proposed in the literature for the variant studied in this work. To address this, we present a BRKGA metaheuristic and an exact algorithm based on an integer linear programming (ILP) formulation.

This article is structured as follows: Section 2 covers related work. Section 3 presents an integer linear programming formulation for the S2G-LM. Next, Section

4 details the proposed BRKGA heuristic. Finally, the computational experiments are discussed in Section 5, followed by conclusions and suggestions for future research in the last section.

2. Related work

In [Amir et al. 2000], the authors prove that the Sequence-to-Graph Alignment Problem with Vertex Label Modification is $\mathcal{NP}$-hard for $|\Sigma| \geq |V|$ based on the demonstration of the asymmetry of edit locations, allowing the reduction of the problem to the Directed Hamiltonian Path and Cycle problem, assuming alterations in the graph or simultaneously in the graph and the sequence.

In [Jain et al. 2020], in contrast to [Amir et al. 2000], whose NP-completeness proofs for alignment with graph modifications relied on the premise of an alphabet large enough to distinctly label each vertex ($|\Sigma| \geq |V|$), Jain et al. proved that the problem remains $\mathcal{NP}$-complete even for constant-size alphabets with $|\Sigma| \geq 2$. Instead of relying on unique characters per vertex, they structured the proof using repetitive binary sequence *tokens* in the query, showing that the problem's difficulty derives from the cyclic nature of graphs, not the alphabet size.

Although the literature does not contain works addressing the specific variant studied in this paper, exact solutions for the classical version of the Sequence-to-Graph Alignment problem have been investigated by several authors, including GraphAligner [Rautiainen and Marschall 2020], AStarix [Ivanov et al. 2020], Theseus [Jiménez-Blanco et al. 2026], and [Jain et al. 2019], among others. However, because these are exact solutions for an $\mathcal{NP}$-hard problem, they are limited to alignments involving smaller graphs and sequences.

3. Integer Linear Programming Formulation

This section presents an integer linear programming formulation for the S2G-LM. Consider a directed graph $G = (V, E, \Sigma)$ in which each vertex $v \in V$ carries a single-character label $\sigma(v) \in \Sigma$, where Σ is the problem alphabet, and a query sequence $q = q[0] \cdots q[m-1]$ of length m over Σ. The objective is to find a directed walk of length m in G that minimizes the number of vertices whose label must be changed so that the sequence induced by the walk matches q.

Let $y_{v,i} \in \{0, 1\}$ be binary variables such that $y_{v,i} = 1$ if vertex $v \in V$ is visited at the i-th step of the alignment, and $y_{v,i} = 0$ otherwise. Additionally, define the variable $S_v \in \{0, 1\}$, where $S_v = 1$ if the original label of vertex v must be changed to satisfy the alignment, and $S_v = 0$ otherwise. Let $\delta^-(v)$ denote the set of predecessors of v in G, and let $\mathcal{A} = \{(i, j) : i < j,\ q[i] \neq q[j]\}$ be the set of conflicting position pairs of the query, i.e., positions that require distinct characters. The problem can be formulated as an integer linear program as defined in (1)–(6).

The objective function (1) seeks to minimize the total number of vertices that require a label change along the selected walk. Constraint (2) ensures that exactly one vertex is visited at each position i of the query sequence, guaranteeing full coverage of all m alignment steps. Constraint (3) enforces the structural continuity of the walk: if vertex v is selected at step $i + 1$, then at least one of its predecessors in G must have been

visited at step i, ensuring that the sequence of selected vertices corresponds to a valid directed walk in G. Constraint (4) prohibits the same vertex v from being visited at two positions $(i, j) \in \mathcal{A}$ that require distinct characters from Σ, preserving the consistency of the unique-label-per-vertex requirement throughout the alignment. Constraint (5) activates the penalty variable S_v whenever vertex v is visited at a position i whose character $q[i]$ differs from the original label $\sigma(v)$, accounting for the editing requirement of that vertex. Finally, constraints (6) define the binary domain of all model variables. The total number of binary variables is $|V|(m+1)$, while the total number of constraints is of order $O(|V| \cdot m^2)$, dominated by the incompatibility constraints (4).

$$\text{Minimize} \quad \sum_{v \in V} S_v \tag{1}$$

$$\text{subject to} \quad \sum_{v \in V} y_{v,i} = 1, \qquad \forall\, i \in \{0, \ldots, m-1\} \tag{2}$$

$$y_{v,i+1} \;\leq \sum_{u \in \delta^-(v)} y_{u,i}, \qquad \forall\, v \in V,\ i \in \{0, \ldots, m-2\} \tag{3}$$

$$y_{v,i} + y_{v,j} \;\leq\; 1, \qquad \forall\, v \in V,\ (i,j) \in \mathcal{A} \tag{4}$$

$$S_v \;\geq\; y_{v,i}, \qquad \forall\, v \in V,\ i \in \{0, \ldots, m-1\} \text{ such that } q[i] \neq \sigma(v) \tag{5}$$

$$y_{v,i},\ S_v \;\in\; \{0,1\}, \qquad \forall\, v \in V,\ i \in \{0, \ldots, m-1\} \tag{6}$$

4. BRKGA Metaheuristic with Island Architecture

Since the worst-case complexity of algorithms based on the ILP formulation of Section 3 grows exponentially with instance size, we propose in this section a metaheuristic based on the Biased Random-Key Genetic Algorithm (BRKGA) [Bean 1994, Gonçalves and Resende 2011] to obtain high-quality solutions within viable computational time for the problem. The BRKGA operates in a continuous search space, where each individual in the population is represented by a vector of real numbers in the interval $[0, 1)$, called random keys. This abstraction allows the evolutionary engine to be problem-independent, delegating solution construction and fitness evaluation to a specialized component called the decoder.

Solution robustness is further enhanced through an island model [Alba and Tomassini 2002], which aims to mitigate premature convergence and preserve genetic diversity across generations. In this configuration, the global population is partitioned into I independent subpopulations that evolve in parallel. Periodically, every E generations, a migration event occurs in which the elite individuals of each island are transferred to the next island according to a ring topology, allowing promising solutions explored in different regions of the search space to propagate throughout the global population.

The decoding process transforms the random-key vector into a valid walk of length m in the graph $G(V, E, \Sigma)$. Unlike traditional edge-indexing approaches, the chromosome length is defined as $|V|$, where each gene $X[v]$ encodes a global genetic priority for vertex v. The walk is constructed step by step. At step i, to select the starting vertex (if $i = 0$)

or the next adjacent vertex (if $i > 0$), the decoder evaluates all valid candidate vertices. The selection is driven by a hybrid scoring function: the score of a candidate v is its base genetic key $X[v] \in [0, 1)$, augmented by a greedy heuristic bonus of 1.0 if its label $\sigma(v)$ matches the required query character $q[i]$, and reduced by a severe penalty of 10.0 if it prematurely leads to a vertex with no outgoing edges. Specifically, the score is calculated as $X[v]+\beta-\gamma$, where $\beta = 1.0$ if $\sigma(v) = q[i]$ (and 0 otherwise), and $\gamma = 10.0$ if $\delta^+(v) = \emptyset$ and $i < m - 1$ (and 0 otherwise). The penalty $\gamma = 10.0$ was chosen to strictly dominate the maximum achievable label-match bonus $\beta_{\max} = 1.0$. The candidate with the highest score is appended to the walk. This vertex-based representation significantly mitigates destructive epistasis, as the genetic value of a vertex retains its semantic utility regardless of the path taken to reach it. Once the walk is established, it is evaluated for compatibility between the label required by position i of the query sequence q and the label available at the visited vertex. If the same vertex is visited at positions requiring distinct characters, an irreducible label conflict is registered, severely penalizing the individual's fitness. If, on the other hand, the vertex is visited repeatedly but always under the same character requirement, a single label edit is recorded for that vertex regardless of how many times it is traversed. The final fitness of each individual is given by the sum of required label edits plus a weighted penalty for the number of irreducible conflicts, ensuring that conflict-free solutions are always preferred. The complete mechanics of the proposed algorithm are presented in Algorithm 1.

Algorithm 1 shows the proposed BRKGA framework. The evolutionary process begins by initializing I independent islands with uniformly distributed random-key chromosomes of dimension $|V|$. During evaluation, a decoder translates each chromosome into a graph walk W. The fitness function calculates the required label edit cost while strictly enforcing structural consistency: it tracks the newly assigned characters for each vertex to ensure that multiple visits to the same vertex do not demand contradictory labels. Any such irreducible conflict is heavily penalized by a large constant $P_{\max}$, ensuring that structurally feasible paths strictly dominate the population.

Following evaluation, each subpopulation evolves through standard BRKGA dynamics: the fittest individuals are preserved as the elite cohort, random mutants are introduced to inject genetic diversity, and the remainder of the population is generated via biased crossover, inheriting alleles from the elite parent with probability p_e. To foster global convergence while preventing premature stagnation, a ring-topology migration occurs every E generations, wherein elite individuals from island k replace the worst individuals of the adjacent island $k' = (k \bmod I) + 1$. This evolutionary cycle repeats until the stopping criterion $G_{\max}$ is met, returning the overall best valid alignment.

5. Computational Results

Both methods were implemented in C++ and evaluated on 360 random Erdős–Rényi graphs [Erdős and Rényi 1959] varying in size ($|V| \in \{60, 80, 100, 120\}$), edge density ($d \in \{10\%, \ldots, 60\%\}$), and relative query length ($q \in \{10\%, \ldots, 90\%\}$). The ILP was solved via Gurobi 13.0 with a $600\,\mathrm{s}$ time limit.

BRKGA parameters were partitioned into two groups. The *dynamic* parameters were set according to the relative query length q, reflecting the distinct computational profiles induced by short and long queries. For $q \leq 30\%$, the walk covers at most

Algorithm 1: Island-based BRKGA

Input: Graph $G(V, E, \Sigma)$, query sequence q of length m,
number of islands I, subpopulation size p,
elite fraction ρ_e, mutant fraction ρ_m,
migration frequency E, maximum generations $G_{\max}$.

Output : Minimum label-edit cost and corresponding optimal walk W^*.

Initialize I subpopulations, each with p random chromosomes in $[0, 1)^{|V|}$
$g \leftarrow 0$
$f^* \leftarrow +\infty$; $W^* \leftarrow \emptyset$
while $g < G_{\max}$ **do**
  for *each island* $k \in \{1, \ldots, I\}$ **do**
    for *each individual* X *in the subpopulation of island* k **do**
      $W \leftarrow \textsc{DecodeWalk}(X, G, q, m)$
      Edits $\leftarrow 0$; *Conflicts* $\leftarrow 0$
      LabelMap $\leftarrow \emptyset$
      for $i \leftarrow 0$ ***to*** $m - 1$ **do**
        $v \leftarrow W[i]$
        if $v \notin$ *LabelMap* **then**
          LabelMap$[v] \leftarrow q[i]$
          if $q[i] \neq \sigma(v)$ **then**
            Edits $\leftarrow$ *Edits* $+ 1$
        else
          if *LabelMap*$[v] \neq q[i]$ **then**
            Conflicts $\leftarrow$ *Conflicts* $+ 1$
      Fitness$(X) \leftarrow$ *Edits* $+$ *Conflicts* $\times P_{\max}$
    Sort the subpopulation of island k in non-decreasing order of *Fitness*
    Preserve the $\lfloor \rho_e \cdot p \rfloor$ elite individuals unchanged
    Insert $\lfloor \rho_m \cdot p \rfloor$ mutants generated uniformly at random in $[0, 1)^{|V|}$
    Fill remaining individuals via biased crossover between elite and non-elite parents
    if *Fitness*(*best of* k) $< f^*$ **then**
      $f^* \leftarrow$ *Fitness*(best of k); $W^* \leftarrow$ corresponding walk
  if $g \bmod E = 0$ **then**
    for *each island* $k \in \{1, \ldots, I\}$ **do**
      $k' \leftarrow (k \bmod I) + 1$
      Replace the worst individuals of island k' with the elite individuals of island k
  $g \leftarrow g + 1$
return f^*, W^*

30% of $|V|$ steps, limiting combinatorial depth and making label conflicts rare; a small configuration of 1000 generations, 4 islands, and 200 individuals per island suffices to explore the solution space adequately. For $q > 30\%$, longer walks substantially increase the likelihood of revisiting the same vertex at positions requiring distinct characters, rendering the fitness landscape more rugged and prone to premature convergence. The expanded configuration of 2000 generations, 6 islands, and 500 individuals maintains broader genetic diversity and allows the ring topology migration to propagate promising solutions before stagnation sets in. The threshold at $q = 30\%$ was identified empirically on a held out validation set, where it consistently separated the regime in which the small configuration saturated early from the one in which the additional budget translated into measurable cost reductions.

The *fixed* parameters follow the canonical values established by Gonçalves and Resende [Gonçalves and Resende 2011] and widely adopted in the BRKGA literature [Londe et al. 2024]: 15% elite partition, 10% mutants, 70% elite allele inheritance probability, migration of 3 individuals every 50 generations, $P_{\max} = m + 1$, and a full restart after 100 stagnant generations.

We refer to the solver's dual bound at termination as the *lower bound* (LB) and to the best primal solution as the *upper bound* (UB). Gaps are defined as: $\text{gap}_{\text{ILP}} = (UB - LB)/UB \times 100\%$, $\text{gap}_{\text{LB}} = (f_{\text{BRKGA}} - LB)/LB \times 100\%$ (where $LB > 0$), and $\text{gap}_{\text{UB}} = (f_{\text{BRKGA}} - UB)/UB \times 100\%$ (where $UB > 0$).

Table 1 consolidates results across all 360 instances. The ILP proved optimality in 284 cases (78.9%) and hit the time limit in 76 (21.1%). BRKGA achieved perfect alignments ($f = 0$) in 11.9% of instances and matched the ILP upper bound in 36.1%.

The complementarity between the two methods is most evident on the hardest instances. BRKGA strictly improved the best-known upper bound in 14 cases, 11 of which occurred at $q = 90\%$. Conversely, on the 284 optimally solved instances, the ILP certifies global optimality while BRKGA's average gap relative to the proven optimum is 45.0%, confirming that the exact method is strictly superior within viable problem sizes. These findings establish the two approaches as complementary: the ILP dominates on small-to-medium instances, while BRKGA provides the most reliable feasible solutions on large, hard instances.

Table 1. General summary of computational results over the 360 instances.

Algorithm	Performance Metric	Value
ILP	Instances solved to optimality	284 (78.9%)
	Instances reaching time limit	76 (21.1%)
	Gap on time-limit instances (Mean / Max)	46.3% / 96.2%
	Mean runtime (optimally solved)	34.57 s
BRKGA	Perfect alignments ($f = 0$)	43 (11.9%)
	Matched ILP upper bound	130 (36.1%)
	Improved ILP UB (time-limit instances)	14 (18.4% of those)
	Mean gap relative to ILP (UB / LB)	45.0% / 95.3%
	Mean runtime (all instances)	187.95 s

5.1. Influence of Query Sequence Size

Table 2 and Figure 2 present results stratified by query size q. Query length is the dominant difficulty factor for both methods.

Table 2. Aggregated results by relative query size (q).

q	n_{inst}	%OPT	$\overline{\text{gap}}_{\text{ILP}}(\%)$	$\bar{f}_{\text{BRKGA}}$	$\overline{\text{gap}}_{\text{LB}}(\%)$	$\overline{\text{gap}}_{\text{UB}}(\%)$	$\bar{t}_{\text{ILP}}(\text{s})$	$\bar{t}_{\text{BRKGA}}(\text{s})$
10%	72	100.0	—	1.44	8.67	8.67	0.1	13.2
30%	72	94.4	22.36	4.53	26.84	25.29	45.9	34.3
50%	72	80.6	31.44	7.89	67.34	53.46	154.7	303.2
70%	72	65.3	44.86	10.38	126.04	59.29	246.7	363.3
90%	72	54.2	56.62	13.58	216.60	66.72	322.8	225.8

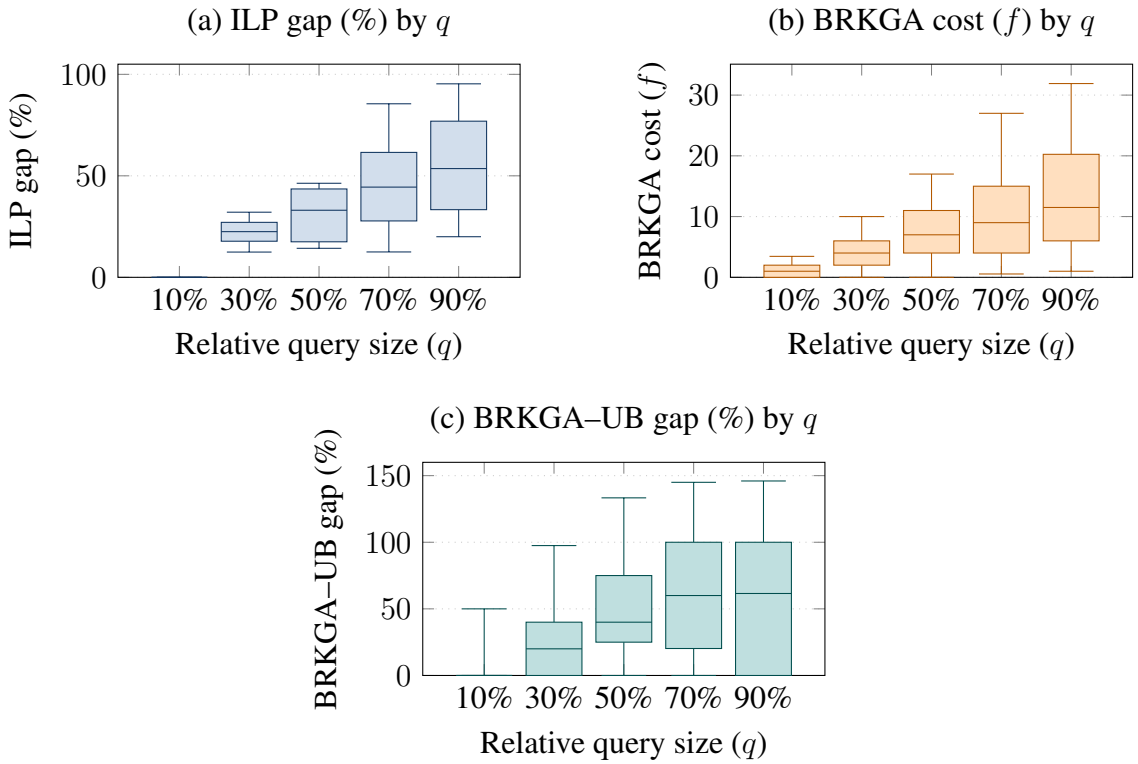


Figure 2. Distributions of key indicators by q. Boxes: IQR (Q1–Q3); line: median; whiskers: 5th–95th percentiles. Panel (a) covers only time-limit instances (ILP gap undefined at $q = 10\%$, all solved optimally).

The ILP solves all 72 instances at $q = 10\%$ in under 1 s; BRKGA attains a mean gap of only 8.67% relative to LB. As q grows, the ILP optimality rate falls monotonically to 54.2% at $q = 90\%$, where the average residual gap reaches 56.62% (maximum 96.20%), evidencing weak LP relaxations on long queries with sparse graphs. BRKGA's average cost rises from 1.44 to 13.58 over the same range, but critically, it outperforms the ILP upper bound in 11 of the hardest $q = 90\%$ instances. The temporal crossover also occurs here: BRKGA is faster than the ILP at $q = 90\%$ (225.8 s vs. 322.8 s), while at intermediate

values ($q \in \{50\%, 70\%\}$) the ILP is faster because it solves many instances quickly and BRKGA always runs its full generation budget.

5.2. Influence of Graph Density

Tables 3–5 and Figure 3 present results by edge density d. The ILP is severely impacted by sparsity: at $d = 10\%$ only 38.3% of instances are solved to optimality, with a mean gap of 53.43% and mean runtime of 384.3 s. This occurs because sparse graphs offer few alternative paths, leading to a vast search space and weak lower bounds. Conversely, denser graphs provide greater connectivity and the solver reaches 100% optimality for $d \geq 50\%$, solved in 6.8 s on average at $d = 60\%$.

Table 3. Aggregated results by edge density (d).

d	n_{inst}	%OPT	$\overline{\text{gap}}_{\text{ILP}}(\%)$	$\bar{f}_{\text{BRKGA}}$	$\overline{\text{gap}}_{\text{LB}}(\%)$	$\overline{\text{gap}}_{\text{UB}}(\%)$	$\bar{t}_{\text{ILP}}$(s)	$\bar{t}_{\text{BRKGA}}$(s)
10%	60	38.3	53.43	16.72	164.80	5.00	384.3	102.3
20%	60	56.7	41.11	10.58	115.95	44.57	295.9	153.2
30%	60	81.7	35.45	7.15	88.21	56.88	158.6	195.7
40%	60	96.7	41.67	5.00	54.26	50.25	57.6	226.1
50%	60	100.0	—	3.25	70.96	70.96	20.9	256.3
60%	60	100.0	—	2.68	52.75	52.75	6.8	194.1

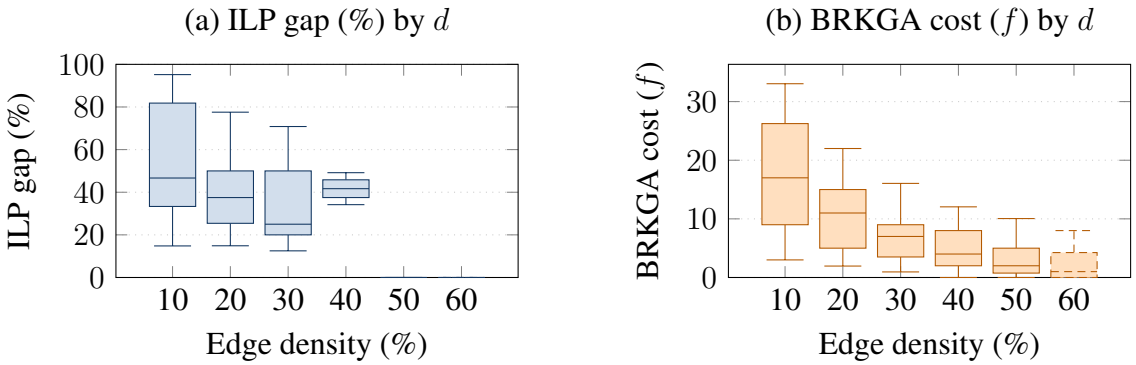


Figure 3. Distributions of ILP gap and BRKGA cost by edge density d. Panel (a) covers only time-limit instances; at $d \geq 50\%$ all instances were solved to optimality (ILP gap identically zero). At $d = 40\%$ only 2 instances reached the time limit.

BRKGA shows the opposite trend on solution cost, average cost falls from 16.72 at $d = 10\%$ to 2.68 at $d = 60\%$, showing that denser graphs facilitate high-quality walks. Notably, at $d = 10\%$ the BRKGA–UB gap is only 5.00%, because the ILP's primal solutions are themselves poor on sparse graphs; the two methods jointly struggle but BRKGA matches those weak bounds efficiently. The density-quality interaction is further quantified in Tables 4–5: the worst case is $d = 10\%$, $q = 90\%$ (zero ILP optimals, $\bar{f}_{\text{BRKGA}} = 30.83$), while at $d = 60\%$, $q = 10\%$ both methods perform near-perfectly ($\bar{f}_{\text{BRKGA}} = 0.33$).

Table 4. Percentage of instances solved to optimality by the ILP.

	Relative query size (q)				
Density (d)	10%	30%	50%	70%	90%
10%	100.0	66.7	25.0	0.0	0.0
20%	100.0	100.0	58.3	25.0	0.0
30%	100.0	100.0	100.0	66.7	41.7
40%	100.0	100.0	100.0	100.0	83.3
50%	100.0	100.0	100.0	100.0	100.0
60%	100.0	100.0	100.0	100.0	100.0

Table 5. Average BRKGA cost by d and q.

	Relative query size (q)				
Density (d)	10%	30%	50%	70%	90%
10%	3.25	9.33	16.58	23.58	30.83
20%	2.17	6.42	10.75	14.00	19.58
30%	1.17	4.50	7.42	10.67	12.00
40%	1.33	3.33	4.42	6.42	9.50
50%	0.42	1.67	4.83	3.92	5.42
60%	0.33	1.92	3.33	3.67	4.17

5.3. Influence of Graph Size

Table 6 summarizes results by number of vertices $|V|$. Larger graphs make the ILP progressively harder: optimality rate drops from 88.9% at $|V| = 60$ to 70.0% at $|V| = 120$, the average gap on time-limit instances rises from 32.89% to 58.44%, and mean ILP runtime nearly triples ($81.7\,\mathrm{s}$ to $214.2\,\mathrm{s}$). On the other hand, BRKGA finds better solutions on larger graphs (average cost decreases from 8.80 to 6.31): holding density and query fraction constant, a larger graph offers more diverse walks, improving alignment quality. BRKGA runtime grows with $|V|$ (100.0 to $270.0\,\mathrm{s}$), but more gently than the ILP.

Table 6. Aggregated results by number of vertices ($|V|$).

$\lvert V\rvert$	n_{inst}	%OPT	$\overline{\text{gap}}_{\text{ILP}}(\%)$	$\bar{f}_{\text{BRKGA}}$	$\overline{\text{gap}}_{\text{LB}}(\%)$	$\overline{\text{gap}}_{\text{UB}}(\%)$	$\bar{t}_{\text{ILP}}$(s)	$\bar{t}_{\text{BRKGA}}$(s)
60	90	88.9	32.89	8.80	44.40	37.10	81.7	100.0
80	90	78.9	34.00	8.21	66.22	48.35	146.4	159.0
100	90	77.8	48.33	6.93	91.46	39.97	173.8	222.8
120	90	70.0	58.44	6.31	201.54	56.82	214.2	270.0

Tables 7–8 cross $|V|$ with q for execution time. The ILP is virtually instantaneous at $q = 10\%$ (max $0.3\,\mathrm{s}$) but runtimes escalate sharply from $q = 30\%$ onward, especially at $|V| \geq 100$. BRKGA's peak occurs at $|V| = 120$, $q = 70\%$ ($557.2\,\mathrm{s}$); its growth with $|V|$ is gentler than the ILP's at long queries, reinforcing the practical value of the metaheuristic for large-scale instances.

Table 7. Mean ILP execution time (s) by $|V|$ and q.

	Relative query size (q)				
$\lvert V\rvert$	10%	30%	50%	70%	90%
60	0.0	0.5	24.6	133.8	249.6
80	0.1	1.0	148.2	281.6	301.4
100	0.2	59.1	211.9	248.5	349.2
120	0.3	122.9	234.0	323.0	390.9

Table 8. Mean BRKGA execution time (s) by $|V|$ and q.

	Relative query size (q)				
$\lvert V\rvert$	10%	30%	50%	70%	90%
60	9.3	21.4	173.5	178.4	117.4
80	12.7	31.9	274.9	292.6	182.9
100	16.7	37.5	373.7	424.9	261.1
120	14.2	46.3	390.5	557.2	341.9

5.4. Extreme Cases

The hardest instances combine $d = 10\%$ and $q = 90\%$: the ILP proved no optimals in any of the 12 cases, with residual gaps from 46.2% to 96.2% and UB values of 26–81 mismatches against LB values of only 3–14. BRKGA consistently found solutions of 24–42 mismatches on these instances, outperforming the ILP upper bound for $|V| \in \{100, 120\}$. At $|V| = 120$, $d = 10\%$, and $q = 90\%$, BRKGA reduced alignment costs by more than 50% compared to the ILP.

6. Conclusion

We introduced an Integer Linear Programming formulation and a Biased Random-Key Genetic Algorithm metaheuristic for the Sequence-to-Graph Alignment Problem with Vertex Label Modification. To the best of our knowledge, these are the first algorithms proposed for this specific problem variant in the literature.

The ILP solved 78.9% of 360 instances to optimality with a mean runtime of 34.57 seconds, certifying solution quality on the bulk of the benchmark. On the remaining 21.1% where the time limit was reached, BRKGA improved the best-known upper bound in 18.4% of those cases. In the most extreme scenario ($|V| = 120$, $d = 10\%$, $q = 90\%$), BRKGA reduced feasible costs by over 50% relative to the ILP's primal solutions. Together, the two methods cover the full spectrum: the ILP excels on small-to-medium, moderately dense instances, while BRKGA is the preferred solver for large, sparse, long-query instances.

One result that deserves explicit reflection is the BRKGA's average gap of 45.0% relative to the proven optima on the 284 instances where the ILP converged. This is higher than one might expect, but it has a straightforward explanation: the ILP tends to solve instances quickly precisely when the optimal walk requires very few label modifications. In these cases, even a single suboptimal vertex choice by the BRKGA may double or triple the cost, making tight solutions hard to reach through evolutionary pressure alone. Rather than undermining the metaheuristic, this asymmetry reinforces the complementarity between the two methods. A natural next step is to incorporate ILP dual information, such as reduced costs or variable fixing bounds, into the BRKGA decoder, which could narrow this gap on moderate instances without compromising scalability on large ones.

Several directions remain open for future work. On the exact side, Constraint Programming solvers such as CP-SAT may handle the walk-consistency and label-conflict constraints more naturally than an ILP, and their propagation mechanisms could yield tighter bounds on the sparse instances where the LP relaxation currently struggles. Branch-and-price is another exact avenue worth investigating. On the heuristic side, adding a local search operator over elite individuals and a Path-Relinking procedure between islands are the most direct fixes. Hybrid schemes, such as using BRKGA solutions to warm-start the ILP or terminating the metaheuristic early when the gap to the ILP lower bound falls below a threshold, seem particularly promising given the complementarity observed in this study.

References

Alba, E. and Tomassini, M. (2002). Parallelism and evolutionary algorithms. *IEEE transactions on evolutionary computation*, 6(5):443–462.

Amir, A., Lewenstein, M., and Lewenstein, N. (2000). Pattern matching in hypertext. *Journal of Algorithms*, 35(1):82–99.

Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA journal on computing*, 6(2):154–160.

Dilthey, A., Cox, C., Iqbal, Z., Nelson, M. R., and McVean, G. (2015). Improved genome inference in the mhc using a population reference graph. *Nature genetics*, 47(6):682–688.

Dilthey, A. T., Gourraud, P.-A., Mentzer, A. J., Cereb, N., Iqbal, Z., and McVean, G. (2016). High-accuracy hla type inference from whole-genome sequencing data using population reference graphs. *PLoS computational biology*, 12(10):e1005151.

Erdős, P. and Rényi, A. (1959). On random graphs i. *Publicationes Mathematicae Debrecen*, 6:290–297.

Garrison, E., Sirén, J., Novak, A. M., Hickey, G., Eizenga, J. M., Dawson, E. T., Jones, W., Garg, S., Markello, C., Lin, M. F., et al. (2018). Variation graph toolkit improves read mapping by representing genetic variation in the reference. *Nature biotechnology*, 36(9):875–879.

Gonçalves, J. F. and Resende, M. G. (2011). Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, 17(5):487–525.

Ivanov, P., Bichsel, B., Mustafa, H., Kahles, A., Rätsch, G., and Vechev, M. (2020). Astarix: Fast and optimal sequence-to-graph alignment. In *International Conference on Research in Computational Molecular Biology*, pages 104–119. Springer.

Jain, C., Misra, S., Zhang, H., Dilthey, A., and Aluru, S. (2019). Accelerating sequence alignment to graphs. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 451–461. IEEE.

Jain, C., Zhang, H., Gao, Y., and Aluru, S. (2020). On the complexity of sequence-to-graph alignment. *Journal of Computational Biology*, 27(4):640–654.

Jiménez-Blanco, A., López-Villellas, L., Moure, J. C., Moreto, M., and Marco-Sola, S. (2026). Theseus: Fast and optimal affine-gap sequence-to-graph alignment. *bioRxiv*, pages 2026–02.

Londe, M. A., Pessoa, L. S., Andrade, C. E., and Resende, M. G. (2024). Early years of biased random-key genetic algorithms: a systematic review. *Journal of Global Optimization*, pages 1–24.

Paten, B., Novak, A. M., Eizenga, J. M., and Garrison, E. (2017). Genome graphs and the evolution of genome inference. *Genome research*, 27(5):665–676.

Rautiainen, M. and Marschall, T. (2020). Graphaligner: rapid and versatile sequence-to-graph alignment. *Genome biology*, 21(1):253.

Vaser, R., Sović, I., Nagarajan, N., and Šikić, M. (2017). Fast and accurate de novo genome assembly from long uncorrected reads. *Genome research*, 27(5):737–746.