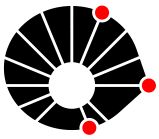

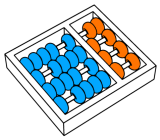

Warm-Starting Exact Solvers for Minimum Common String Partition Problems

Pedro Carvalho Cintra *Gabriel Siqueira* *Zanoni Dias*

Relatório Técnico - IC-PFG-26-24
Projeto Final de Graduação
2026 - Julho

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Warm-Starting Exact Solvers for Minimum Common String Partition Problems

Pedro Carvalho Cintra, Gabriel Siqueira, Zanoni Dias

July 2026

Abstract

The Signed Minimum Common Intergenic String Partition (SMCISP) problem is used in approximation algorithms for genome-rearrangement distances and is NP-hard even over binary alphabets. We experimentally compare two exact approaches—a fixed-parameter branching algorithm and integer linear programming (ILP)—and study whether heuristic information improves them. We consider (i) time to certify optimality under explicit safety limits and (ii) the best feasible solution available after a fixed solver budget. On small instances and on the $n = 1000$ high-alphabet families ($K = 20$ and $K = 36$), the cold ILP is substantially faster than the FPT algorithm and certifies optimality; the FPT also certifies on the small instances but frequently times out at $n = 1000$. The low-alphabet $K = 4$ family is an important exception: the ILP formulation becomes very large, while the FPT algorithm finds better feasible solutions within the tested budget. Supplying only a valid upper-bound value produces no practically meaningful runtime gain in the tested configurations. Moreover, a cost-only FPT seed may leave no materialized incumbent partition, and an ILP cutoff may leave the solver with no internal incumbent, although the heuristic partition remains available to the warm-start pipeline. Supplying the complete heuristic partition as a MIP start to the CB model avoids this loss of information: on the tested $K = 4$ intergenic instance, the incumbent available after the solver budget changes from 1234 to 864 blocks, while the reported optimality gap changes from about 50% to 28%. The ILP does not improve upon the seeded partition within 300 s; its contribution in this experiment is to provide a lower bound and certify the quality of the heuristic incumbent. Overall, the results show that the usefulness of a warm start depends on both instance structure and the information passed to the solver, and that the FPT and ILP approaches are complementary on the tested benchmark.

1 Introduction

One of the central questions in comparative genomics is determining how evolutionarily distant two species are. While nucleotide-level sequence alignment provides fine-grained information, another complementary perspective comes from examining how the *arrangement* of genes differs between organisms [1, 2]. Even when two genomes share an almost identical set of genes, those genes may appear in wildly different linear orders on the chromosomes—a consequence of large-scale mutational events such as reversals, transpositions, and insertions or deletions of genetic material. The central combinatorial object used to reason about these differences is the *genome rearrangement distance*: the minimum number of such operations needed to transform one genome into another. Section 2 makes this model precise, defining genomes, rearrangement events, and the family of common-partition problems used throughout this work.

Computing rearrangement distances has attracted sustained attention from both biologists and algorithm designers. From a biological standpoint, small distances correlate with closer evolutionary

relationships; from a computational standpoint, even the simplest formulations of the problem are surprisingly challenging: most of its variants—sorting by reversals and by transpositions, and their weighted and intergenic generalisations—are NP-hard [3, 4, 5], and the associated string-partition formulation remains NP-hard even for binary alphabets [6]. A key intermediate step in bounding the rearrangement distance is the *Minimum Common String Partition* (MCSP) problem and its signed, intergenic-region-aware extensions [7, 8], which ask for the partition of two genomes into the fewest possible common blocks. Understanding the precise computational boundary between tractable and intractable instances of this problem, and developing fast exact or approximation algorithms for the hard cases, is therefore a question of both theoretical and practical importance.

The NP-hardness of these problems means that no polynomial-time algorithm is expected to solve them exactly in the worst case. Nevertheless, exact solutions remain valuable for the genome sizes encountered in practice, since they provide certified lower bounds on the rearrangement distance. Two complementary exact approaches form the experimental core of this work. Siqueira et al. [8] showed that the signed, intergenic-region-aware partition problem is fixed-parameter tractable (FPT) when parameterised by the partition cost, and gave a branching algorithm that solves it exactly. Romeiro et al. [9] instead formulated the problem as an integer linear program (ILP), proposing two competing models that select a minimum-weight set of common blocks. Both approaches are described in detail in Section 3.

Both exact methods may benefit from an early feasible solution, but the mechanism and the potential gain differ. In the FPT algorithm, a valid upper bound can let the search prune branches whose best achievable cost cannot beat the bound. In the ILP solver, an objective cutoff can prune nodes, while a complete MIP start additionally supplies an incumbent solution the solver keeps and tries to improve. Whether either mechanism actually reduces total running time or improves fixed-budget solution quality is an empirical question, and it is the one this study sets out to answer. The MCSP-Algorithms framework of Siqueira et al. [10] provides a family of efficient greedy and metaheuristic approximations for MCSP, and a dedicated heuristic exists for the full signed, intergenic-region, and indel model. The block count produced by either family of heuristics constitutes a valid upper bound on the optimal partition cost; we communicate this bound to the FPT algorithm as an early pruning threshold, and to the ILP solver as an objective cutoff or a full MIP start, and then measure whether doing so pays off. Figure 1 sketches this warm-start pipeline; Section 3 details how each of its components is realised in practice.

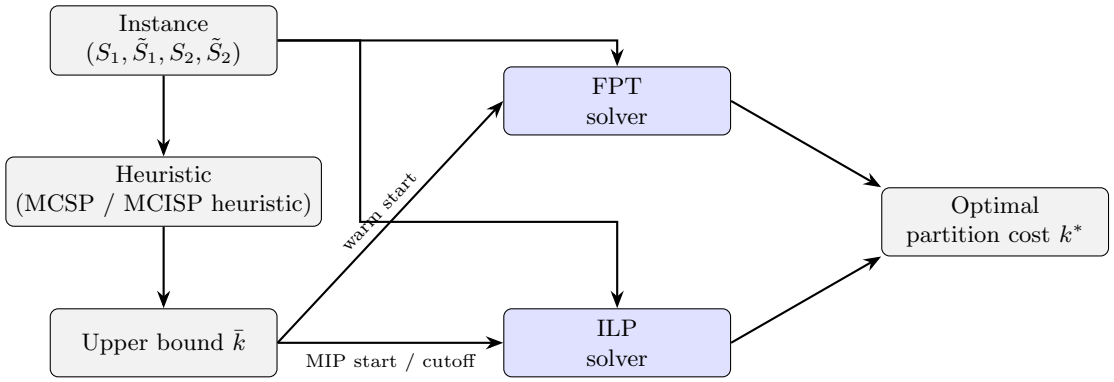

Figure 1: Warm-start pipeline. A heuristic is run first to construct a feasible partition and its objective value $\bar{k}$. This value is then communicated to the two exact solvers: the FPT interface may use it as an upper bound, whereas the ILP interface may use either an objective cutoff or the complete partition as a MIP start. These inputs can enable earlier pruning or supply an incumbent, but their practical effect on certifying the optimum k^* and on fixed-budget quality is evaluated experimentally (Section 4) rather than assumed.

Scope and contributions. This document reports an experimental study of the cold behaviour and the warm-start strategy for the two exact solvers, organised around six directional *hypotheses* that separate two evaluation regimes: *capped time to certification*—whether and how quickly each solver certifies optimality before an explicit safety limit—and *fixed-budget solution quality*—the best feasible solution available when the solver is stopped after a prescribed optimization budget. They fall into three natural pairs:

H1. *Cold ILP faster than cold FPT*: With no warm start, the ILP has lower execution time than the FPT.

H2. *FPT warm start reduces FPT runtime*: Seeding the FPT with a heuristic upper bound lowers its execution time versus a cold FPT.

H3. *ILP warm start reduces ILP runtime*: Seeding the ILP with a heuristic cutoff lowers its execution time versus a cold ILP.

H4. *Cold ILP better quality than cold FPT*: Under the same fixed solver budget, the ILP returns a better (fewer-block) solution than the FPT.

H5. *FPT warm start improves FPT quality*: Under a fixed budget, a warm-started FPT returns a better solution than a cold FPT.

H6. *ILP warm start improves ILP quality*: Under a fixed budget, a warm-started ILP returns a better solution than a cold ILP.

H1 and H4 compare the two exact methods cold, isolating their intrinsic efficiency from any warm-start effect. H2 and H3 evaluate bound-only warm starts—H2 on the FPT, H3 on the ILP—generated by the four heuristic configurations (the native MCISP2K bound and the three signed MCSP-Algorithms variants: greedy, combine, and particle-swarm). H5 and H6 instead measure fixed-budget solution quality under the single valid greedy-intergenic bound: H5 on the FPT, and H6 on the ILP through two distinct interfaces—a bound-only cutoff and a CB-specific full MIP start built from that same partition. The full MIP start is therefore a separate, CB-only experiment, not the four-heuristic cutoff sweep applied through a second interface. For each hypothesis we report, on

a common benchmark, the runtime or solution-quality effect as a *paired*, per-instance quantity with its spread, the tightness of the heuristic bound relative to the certified optimum, and the instance characteristics that govern the outcome; the statistical conventions—robust spreads for heavy-tailed ratios, completion rates for the right-censored time-to-certification data, and paired differences for fixed-budget quality—are detailed in Section 3.6.

Beyond the individual verdicts, the study yields one cross-cutting result that organises all of them: what matters is not whether the heuristic is intergenic-aware (two of the four already are), but *how* its output is communicated to the solver. A *bound-only* warm start—the classical cutoff or incumbent-cost—helps nowhere: where the cold solver is already fast the valid bound has nothing to prune, and where the solver struggles a bare cost cannot transmit the better partition the heuristic already holds (the FPT then emits a phantom, the ILP no incumbent). Turning that diagnosis into a fix is our main constructive contribution: we build the heuristic partition directly on the ILP's own variables (an intergenic-region-aware greedy construction that anticipates the solvers' block-merging) and hand it over as a *full MIP start* (for the CB model). On the one instance family where the ILP genuinely struggles—low-alphabet, high-repeat $n = 1000$—this surfaces and certifies a $\sim$30% better fixed-budget solution and roughly halves the optimality gap, precisely the case a bound alone could not touch.

The remainder of the document is organised as follows. Section 2 introduces the genome model, the family of common-partition problems, and the balanced-instance reduction used throughout. Section 3 describes the heuristics, the two exact algorithms, the warm-starting mechanism, how solution costs are compared across solvers, and the design of the experiments. Section 4 reports the results for each of the six hypotheses in turn and closes with a synthesis of the cross-cutting findings. Section 5 discusses the implications of these results and directions for future work.

2 Definitions

2.1 Genomes and Rearrangement Events

In our computational model a genome is a sequence of *genes*, each represented by an integer label. When the two orientations of a gene on the DNA strand are relevant, the gene is *signed*: a positive label $+g$ indicates the forward strand and $-g$ indicates the reverse complement. Between consecutive genes lie *intergenic regions*—stretches of non-coding nucleotides whose lengths are relevant to how rearrangement events break and rejoin chromosomes.

Formally, a genome is encoded as a pair $G = (S, \tilde{S})$, where $S = (g_1, g_2, \ldots, g_n)$ is the signed gene sequence of length n and $\tilde{S} = (\tilde{s}_1, \tilde{s}_2, \ldots, \tilde{s}_{n-1})$ is the intergenic-region vector of length $n - 1$. Each $\tilde{s}_i$ is a non-negative integer giving the length (in nucleotides) of the intergenic region that lies *between* gene g_i and gene g_{i+1}. There is no intergenic region before the first gene or after the last gene in this formal representation; any real flanking nucleotides are handled separately during a capping step that adds artificial sentinel genes at both ends of each genome. Figure 2 gives a schematic view.

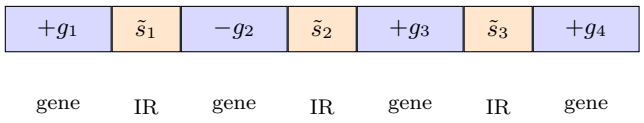

Figure 2: Genome model $G = (S, \tilde{S})$ with $n = 4$ signed genes (blue) and $n - 1 = 3$ intergenic regions (orange) strictly between consecutive genes. There is no intergenic region before g_1 or after g_4; flanking nucleotides are handled by capping.

Three rearrangement-event classes considered in this report are illustrated below; in each example a gene label is deliberately repeated (gene +1), to emphasise that the model admits replicated genes.

- **Reversals (inversions).** A reversal $\rho(i, j)$, with $1 < i \leq j < n$, inverts the segment $[g_i, \ldots, g_j]$, reversing both the order and the signs of the genes in that interval. At the two cut points, the intergenic region $\tilde{s}_{i-1}$ (between g_{i-1} and g_i) is split at a position x with $0 \leq x \leq \tilde{s}_{i-1}$, and the region $\tilde{s}_j$ (between g_j and g_{j+1}) at a position y with $0 \leq y \leq \tilde{s}_j$. The inner regions $\tilde{s}_i, \ldots, \tilde{s}_{j-1}$ are reversed in order together with their genes but keep their lengths, while the two cut regions are recombined so that the new left region becomes $x+y$ and the new right region $(\tilde{s}_{i-1}-x)+(\tilde{s}_j-y)$; the total intergenic material is thereby conserved (Figure 3). Reversals that touch a genome end ($i = 1$ or $j = n$) are defined after the capping step described with the genome model: the sentinel genes added at both ends supply the boundary regions $\tilde{s}_0$ and $\tilde{s}_n$, so that every reversal is internal in the capped genome and the same split-and-recombine rule applies.

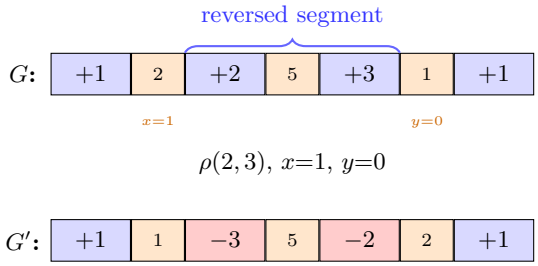

Figure 3: Reversal $\rho(2, 3)$ with cut parameters $x = 1$, $y = 0$ applied to $G = (S, \tilde{S})$ with $S = (+1, +2, +3, +1)$ and $\tilde{S} = (2, 5, 1)$ (gene +1 repeated). The segment [+2, +3] is inverted (genes in red become −3, −2) and the flanking intergenic regions are redistributed: the left region changes from 2 to $x+y = 1$, the right from 1 to $(2-x)+(1-y) = 2$; the total intergenic material $2+5+1 = 8$ is conserved.

- **Transpositions.** A transposition $\tau(i, j, k)$, with $i < j < k$, moves the segment $[g_i, \ldots, g_{j-1}]$ to a position after gene g_{k-1}, leaving gene signs unchanged; its internal regions $\tilde{s}_i, \ldots, \tilde{s}_{j-2}$ travel with it unchanged. Three boundary regions are cut—$\tilde{s}_{i-1}$ (before the segment), $\tilde{s}_{j-1}$ (after it), and $\tilde{s}_{k-1}$ (the target gap)—split respectively at parameters $x \in [0, \tilde{s}_{i-1}]$, $y \in [0, \tilde{s}_{j-1}]$, and $z \in [0, \tilde{s}_{k-1}]$, and their fragments recombine into three new boundary regions: the closed source gap $g_{i-1} \,|\, g_j$ becomes $x + (\tilde{s}_{j-1} - y)$, the region before the reinserted segment $g_{k-1} \,|\, g_i$ becomes $z + (\tilde{s}_{i-1} - x)$, and the region after it $g_{j-1} \,|\, g_k$ becomes $y + (\tilde{s}_{k-1} - z)$. These three sum to $\tilde{s}_{i-1} + \tilde{s}_{j-1} + \tilde{s}_{k-1}$, so the total intergenic length is conserved (Figure 4); transpositions that touch a genome end are defined after capping, whose sentinels supply any missing boundary region.

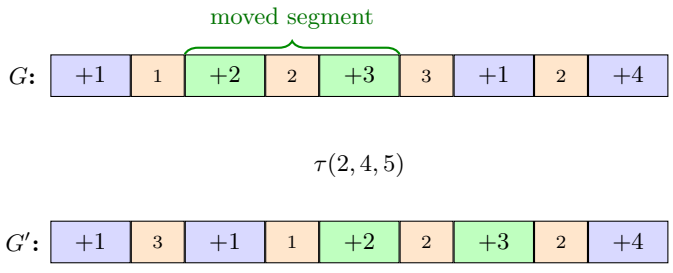

Figure 4: Transposition $\tau(2, 4, 5)$ applied to $G = (S, \tilde{S})$ with $S = (+1, +2, +3, +1, +4)$ and $\tilde{S} = (1, 2, 3, 2)$ (gene $+1$ repeated). The segment $[+2, +3]$ (green) is moved to the position after gene $+1$, giving $S' = (+1, +1, +2, +3, +4)$; signs are unchanged. The block's internal region ($\tilde{s}_2 = 2$) travels with it. The three cut regions $\tilde{s}_1 = 1$, $\tilde{s}_3 = 3$, $\tilde{s}_4 = 2$ are split at $x = y = z = 0$, so the new boundary lengths are $x + (\tilde{s}_3 - y) = 0 + 3 = 3$ (closed source gap), $z + (\tilde{s}_1 - x) = 0 + 1 = 1$ (before the reinserted segment), and $y + (\tilde{s}_4 - z) = 0 + 2 = 2$ (after it), yielding $\tilde{S}' = (3, 1, 2, 2)$ and conserving the total intergenic material $1+2+3+2 = 8$.

- **Indels (insertions/deletions).** An insertion introduces a new gene (or block of genes) that has no counterpart in the other genome, while a deletion removes such a block. These are the only events that alter the total gene count, making them conceptually distinct from the two conservative classes above. When a gene is deleted, its two flanking intergenic regions merge into one whose length is their sum (Figure 5).

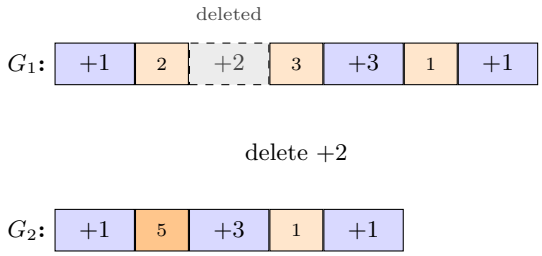

Figure 5: Deletion of gene $+2$ from $G_1 = (S_1, \tilde{S}_1)$ with $S_1 = (+1, +2, +3, +1)$ and $\tilde{S}_1 = (2, 3, 1)$ (gene $+1$ repeated), yielding G_2 with $S_2 = (+1, +3, +1)$. Removing the dashed gene $+2$ merges its two flanking regions into a single one of length $2 + 3 = 5$ (darker), so that no intergenic material is lost.

2.2 Common Partition Problems

Given two genomes $G_1 = (S_1, \tilde{S}_1)$ and $G_2 = (S_2, \tilde{S}_2)$ over the same gene alphabet, the *rearrangement distance* $d(G_1, G_2)$ is the minimum number of rearrangement events needed to transform G_1 into G_2. A key intermediate step in bounding $d(G_1, G_2)$ is identifying a *common partition*: a pair of partitions whose blocks form the same multiset of contiguous signed substrings, so that the block sequence of G_1 can be permuted—and, in the signed setting, blocks may be inverted according to the problem definition—to obtain the block sequence of G_2. The number of blocks in a common partition is related to the number of breakpoints that must be resolved by rearrangement operations. Because one operation may affect more than one breakpoint, the partition cost is not itself the rearrangement distance; instead, it can be converted into an operation-dependent bound through the approximation relations established for the corresponding rearrangement model. Figure 6 illustrates the idea for a pair of signed gene sequences.

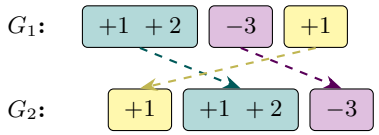

Figure 6: A common partition of two signed gene sequences into three blocks (colours), with gene +1 repeated: it appears both in the block "+1 +2" and as the singleton block "+1". The blocks of G_1 can be reordered—here by one transposition—to obtain G_2. The partition cost can be converted into an operation-dependent lower bound on the rearrangement distance; it is not, by itself, a direct upper bound on that distance.

This observation motivates the *Minimum Common String Partition (MCSP)* problem: given two strings A and B over a common alphabet Σ that are *balanced* (i.e., $|A| = |B|$ and every symbol occurs equally often in A and B), find a partition of A into the fewest possible substrings such that the same substrings, in some order, also partition B. MCSP is NP-hard even for binary alphabets [6], and the hardness persists under a variety of extensions. Allowing genes to carry orientation yields the *Signed* MCSP (SMCSP); further requiring the partition to respect intergenic-region lengths yields the *Signed Minimum Common Intergenic String Partition* (SMCISP), first studied, in its unsigned form, by Siqueira et al. [7] and extended to the signed case with indels by Siqueira et al. [8], allowing unbalanced genomes. Siqueira et al. [11] further generalise the model to allow bounded tolerance in intergenic-region matching, yielding the *Signed Minimum Common Flexible Intergenic String Partition* (SMCFISP).

Balanced sub-instances for the heuristics. The common-substring heuristics of Section 3.1 are defined only for *balanced* strings, so applying them to an unbalanced pair first requires constructing a balanced sub-instance. For each distinct gene label g, let

$$\text{shared}(g) \;=\; \min\bigl\{\#_g(S_1),\, \#_g(S_2)\bigr\},$$

where $\#_g(S)$ denotes the number of occurrences of g in S; a balanced pair keeps shared(g) occurrences of each label in each genome. With repeated genes, *which* occurrences to keep is not unique—it is an orthology/exclusion assignment—and retaining the *first* shared(g) occurrences of each label, as we do, is one deterministic choice; we do not claim it preserves the optimum. The resulting pair bal_1, bal_2 is used only to give the heuristics a feasible matching, and hence a *valid upper bound* on the cost of the original unbalanced instance.

Let $d_1 = |S_1| - |bal_1|$ and $d_2 = |S_2| - |bal_2|$ be the numbers of gene occurrences omitted from G_1 and G_2 by this construction. Re-adding each omitted gene as a singleton block extends any k-block common partition of the balanced pair to a feasible partition of the full instance with $k + d_1 + d_2$ blocks—which is what makes the heuristic block count a valid upper bound. These per-gene counts must be distinguished from D_1, D_2, the numbers of *maximal deletion intervals* that enter the NC and ILP objectives (Sections 3.2 and 3.3): consecutive omitted genes coalesce into a single interval, so $D_1 \le d_1$ and $D_2 \le d_2$ in general. The exact solvers do *not* use this balanced reduction; they handle exclusive genes natively—the FPT branches over deletion-interval configurations (Section 3.2) and the unbalanced ILP includes exclusive-block variables (Section 3.3). Figure 7 illustrates the balancing rule.

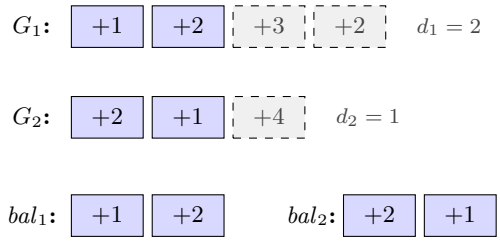

Figure 7: The deterministic heuristic balancing rule (one possible orthology assignment). $G_1 = (+1, +2, +3, +2)$ and $G_2 = (+2, +1, +4)$ are unbalanced. For each label, shared$(g) = \min\{\#_g S_1, \#_g S_2\}$ occurrences are kept: here shared(1) = shared$(2) = 1$, while gene 3 (only in G_1), gene 4 (only in G_2), and the second copy of gene 2 are omitted (dashed) — the $d_1 = 2$ and $d_2 = 1$ exclusive genes. Retaining the *first* such occurrences is one deterministic choice; the resulting balanced pair $bal_1 = (+1, +2)$ and $bal_2 = (+2, +1)$ (same multiset) is what the *heuristics* partition, yielding a valid upper bound once the omitted genes are re-added as singletons. The exact solvers do not use this rule—they handle the exclusive genes directly.

The NP-hardness of these partition problems means that no polynomial-time algorithm is expected to solve them exactly in the worst case. Certified exact solutions nevertheless remain valuable at the genome sizes encountered in practice, since they provide guaranteed lower bounds on the rearrangement distance and allow specific evolutionary scenarios to be ruled out. Section 3 describes the heuristic, FPT, and ILP methods used in this work, together with the warm-start mechanism that connects them.

3 Methodology

This section describes the algorithmic components used in the experiments and the design of those experiments. Section 3.1 describes the common-substring heuristics used to compute warm-start bounds and how they are generalised to the signed, unbalanced, and intergenic-region-aware settings introduced in Section 2. Sections 3.2 and 3.3 describe the FPT and ILP exact algorithms, respectively. Section 3.5 explains how a heuristic bound is turned into a valid warm start for each solver, and Section 3.6 describes how the resulting experiments are organised.

3.1 Common-Substring Heuristics

The MCSP-Algorithms framework of Siqueira et al. [10][1] provides a family of fast approximation heuristics for MCSP, later extended to the signed and intergenic-region-aware settings. Most of them operate on a common weighted-graph representation of the two input strings.

Graph representation. Given two strings A and B, the heuristics build a weighted graph $G_{A,B} = (V, E, \varphi)$ whose vertices are the individual symbol *occurrences* of A and B: $V = \{(A, i) : 1 \leq i \leq |A|\} \cup \{(B, j) : 1 \leq j \leq |B|\}$. An edge $\{(A, i), (B, j)\}$ is added whenever $A[i] = B[j]$, so every pair of matching symbols from opposite strings is connected. Selecting a set of pairwise non-conflicting edges (no two edges sharing a vertex) that covers every vertex of both strings is equivalent to choosing a common partition: consecutive selected edges whose endpoints are simultaneously adjacent in A and in B merge into a single block, while an isolated selected edge forms a block of length one. The weight $\varphi(\{(A, i), (B, j)\})$ rewards exactly this adjacency: it increases with the number of

[1] Implementation: `https://github.com/Problemas-de-Particao-de-Strings/MCSP-Algorithms`.

neighbouring position pairs $(A[i \pm 1], B[j \pm 1])$ that are themselves connected by an edge, so that a heuristic guided by φ favours matches that extend into longer common blocks. Figure 8 illustrates this on a minimal example.

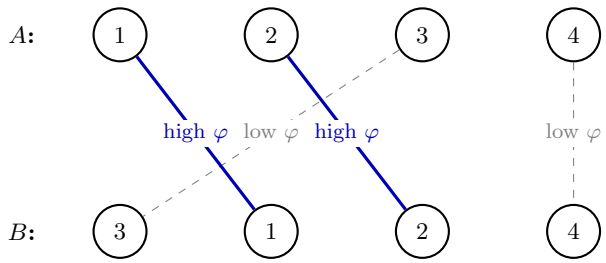

Figure 8: Graph representation $G_{A,B}$ for $A = (1, 2, 3, 4)$ and $B = (3, 1, 2, 4)$. Every matching symbol between the two strings is an edge, and each edge carries its weight φ. The solid edges $\{(A, 1), (B, 2)\}$ and $\{(A, 2), (B, 3)\}$ are mutually adjacent in both strings and receive high weight, since selecting both merges positions 1–2 of A with positions 2–3 of B into a single block "1 2". The dashed edges $\{(A, 3), (B, 1)\}$ and $\{(A, 4), (B, 4)\}$ have no adjacent partner and receive low weight, since each can only ever form a singleton block.

Combine and CombineS. The Combine heuristic maintains a max-priority queue of candidate edges ordered by *non-increasing* weight and greedily adds each edge to the matching provided it does not conflict with any previously selected edge. Because higher-weight edges correspond to matches embedded in longer mutually-consecutive runs, processing them first tends to merge adjacent single-gene matches into longer common substrings, thereby reducing the block count k. CombineS follows the same greedy selection criterion but scans edges in a fixed canonical order determined by the symbol positions in A, trading adaptivity for deterministic reproducibility. Both algorithms run in $O(|A|^2 \log |A|)$ time, dominated by sorting the edge set.

Greedy. The Greedy heuristic works directly on the substring partition rather than on the shared edge graph. It keeps a *complete* partition of each string—initialised as the two whole strings—and repeatedly extracts a common block: at each step it finds the *longest* common substring over all pairs of current blocks (trying both the direct form and, in the signed case, the reverse complement of each block), removes that substring from the two blocks that contained it, and records it as a matched block. Ties are broken deterministically—first by longer common substring, then by shorter source blocks (which tends to leave fewer leftover pieces), then by smaller block indices, with a direct match preferred over a reverse-complement one. The process stops when no common substring remains; the blocks still present are then the final unmatched pieces (single genes and, on unbalanced instances, the surplus genes counted as indels). Because the partition is complete at every step, its block count k is always well defined, and each extraction takes the locally longest available match.

Particle Swarm Optimisation. The PSO heuristic searches over *weightings of the matching-graph edges*: each particle p is a real weight vector over the edge set, decoded into a solution by selecting mutually compatible edges in weight order and reading off the induced partition, whose block count k is the particle's objective (lower is better). A swarm of N_p particles is updated for a fixed number of iterations according to

$$p_{j+1} \;=\; p_j + k_L\,(p_L - p_j)\cdot R_L + k_G\,(p_G - p_j)\cdot R_G + k_E\,R_E, \tag{1}$$

where p_L is the particle's personal-best weighting, p_G the swarm's global-best weighting, and R_L, R_G, R_E are independent random vectors perturbing the local, global, and exploratory components; the scalar weights (k_E, k_L, k_G) trade exploration against convergence. We use the framework defaults throughout: $N_p = 200$ particles, 100 iterations, initial-distribution weights $(1, 3, 3, 3)$ for seeding the first swarm, update weights $(k_E, k_L, k_G) = (2.0, 0.1, 0.1)$, and a fixed random seed, so each run is reproducible (Section 3.6). The decoded partition may optionally be post-processed with Combine, but this is disabled in our runs. Related variants replace the decoder with alternative greedy operators, yielding a family of related metaheuristics.

Generalisation to the unbalanced problem. Genome rearrangement instances are, in general, unbalanced: the two gene sequences may not share the same multiset of labels, with the exclusive genes representing indel events. The heuristics above are defined only for balanced strings, so they are applied to the balanced sub-instance (bal_1, bal_2) obtained by the reduction of Section 2.2, rather than to the raw sequences. Since every common partition of (bal_1, bal_2) with k blocks extends to a valid partition of the full instance with k common blocks and $d_1 + d_2$ singleton indel blocks, the block count k returned by any of the heuristics above is always a valid upper bound on the number of common blocks of the full, unbalanced instance—which is exactly the quantity required for a sound warm start.

Generalisation to signed genomes. Signed gene sequences carry orientation information: a gene g on the reverse strand is written $-g$. The signed variants of the heuristics operate on balanced signed sequences, preserving orientation during the balancing step. A candidate match may connect positions i in S_1 and j in S_2 whenever $|S_1[i]| = |S_2[j]|$, regardless of sign. Consecutive matches form a *direct* block when they advance in the same direction in both genomes and the matched genes share their orientation ($S_1[i] = S_2[j]$ for each pair); they form an *inverted* block when the order is reversed in one genome and every orientation is flipped—i.e. the substring in one genome is the signed reverse $(x_1, \ldots, x_L) \mapsto (-x_L, \ldots, -x_1)$ of the substring in the other, so consecutive S_1 positions match *decreasing* S_2 positions with opposite signs. Each heuristic applies these adjacency and orientation criteria—together with the intergenic-region test of the next paragraph—when scoring and selecting edges. Allowing inverted blocks enlarges the feasible set, so the signed *optimum* can only be equal to or smaller than the unsigned one; this does not, however, guarantee that any individual signed heuristic run returns a bound at least as tight as any unsigned run.

Generalisation to intergenic regions. Two complementary strategies make the heuristic bound aware of intergenic-region (IR) lengths. The first leaves the heuristics themselves unchanged and instead *lifts* the gene-only block count k they return to a valid bound on the cost of the intergenic-region-aware exact problem, since the plain graph representation above never inspects IR values while constructing a matching. Three lifting strategies of increasing tightness are used. The conservative formula $2k + d_1 + d_2 - 1$ requires no IR data at all; it treats every common block as contributing a breakpoint at each of its two endpoints in the worst case, in which no two adjacent gene pairs happen to share compatible IR values. The intergenic-verified strategy instead takes the complete occurrence matching (or block decomposition) returned by the gene-level heuristic—not merely its scalar block count k—and applies the same IR-compatibility and block-merging rule used by the target exact formulation (Section 3.2), using the instance's actual intergenic-region values to decide which consecutive matched genes may remain in the same common block. The resulting complete partition is feasible for the intergenic instance, so its objective value is a valid upper bound. The greedy-intergenic strategy bypasses the plain heuristic altogether and builds a gene matching directly, extending the current block whenever the next gene is consecutive in both genomes *and* the adjoining IR values agree; this often yields the tightest bound on highly rearranged instances. These rules do

not share a single cost convention: the formula $2k + d_1 + d_2 - 1$ is stated in the FPT NC convention (it carries the same -1 boundary term as Equation (2)), whereas the intergenic-verified and greedy-intergenic strategies return a complete partition whose cost is the total block count $C + D_1 + D_2$. Before any bound is compared across solvers or passed as a warm start it is converted to a common convention via Equation (3) (Section 3.4); the experiments here use the greedy-intergenic bound, already in the $C + D_1 + D_2$ convention. The second, complementary strategy instead makes the heuristic itself IR-aware, gating edge admission in the graph representation on IR compatibility between the matched positions, so that a match embedded in an IR-incompatible context is never selected in the first place; the two strategies may be combined.

3.2 The FPT Algorithm

The FPT algorithm for SMCISP, due to Siqueira et al. [8],[2] is parameterised by the solution cost k and solves the problem exactly in time $O(o^{2c'+\Delta_\Sigma}\, k^2\, n)$, where o is the maximum number of occurrences of any gene, k is the optimal partition cost, and c' and Δ_Σ are structural parameters of an auxiliary *sample graph* that capture how many genes are non-singleton or unbalanced between the two genomes. The sample graph records, for each gene occurrence, the set of positions in the other genome it could still be matched to; forced pairings are recorded as black edges, potential pairings as red or green edges, and incompatible pairings are crossed out. At each node of the search tree, a set of data-reduction rules simplifies the current instance; the algorithm then either identifies a gene occurrence with a unique remaining candidate and commits to it, branches on the endpoints of an ambiguous alternating path in the sample graph, or concludes that only cycles and even paths remain and assembles a complete solution directly.

Cost modes. Two cost functions are supported. In **CC mode** (Common-Cost), the objective is to minimise the number of common blocks, corresponding directly to the pure common-partition objective of Section 2.2. In **NC mode** (Neighbour-Cost), the objective additionally accounts for the indel structure of the unbalanced instance:

$$\mathrm{NC} \;=\; \mathrm{computedK} + D_1 + D_2 - 1, \tag{2}$$

where computedK is the number of common blocks and D_1, D_2 are the numbers of maximal consecutive runs of exclusive (deleted) genes in G_1 and G_2, respectively. NC mode measures the total number of intervals a genome is partitioned into—common blocks plus deletion intervals—minus one for the boundary shared by all intervals, and is the natural objective when the exact partition cost is meant to approximate the full rearrangement distance, including indel events. Because an optimal NC solution may require a specific configuration of deletion intervals that is not reachable while branching purely on common blocks, the search is run in two phases: a first phase considers only common-block branches, and a second phase repeats the search with deletion-interval branching also enabled.

3.3 The ILP Models

The integer linear programming formulations for MCSP and its signed and intergenic-region extensions, due to Romeiro et al. [9],[3] cast the partition problem as a selection problem over a finite set of candidate *common blocks*. Two models are proposed, differing in how they index the candidate blocks; both apply uniformly to MCSP, SMCSP, SMCISP, and SMCFISP.

[2]Implementation: `https://github.com/compbiogroup/Signed-Rearrangement-Distances-Considering-Repeated-Genes-Intergenic-Reg`
The same code base provides the native MCISP2K heuristic bound used as a warm start (Section 3.5).

[3]Implementation: `https://github.com/fmromeiro/string-partition-experiments`. This repository also hosts the warm-start bridge and experimental harness used throughout Section 3.6.

The CB model. In the Common-Blocks (CB) model, the candidate set contains every feasible common block: a triple $(t_i, k_{1,i}, k_{2,i})$ for each (possibly signed) substring t_i common to S_1 and S_2, at each pair of starting positions $k_{1,i}$ in S_1 and $k_{2,i}$ in S_2 where it occurs. A binary variable $x_i \in \{0, 1\}$ indicates whether that paired occurrence is selected. Because a CB variable already couples one occurrence in S_1 with one in S_2, the model needs only *coverage* constraints: every retained position of each genome must belong to exactly one selected common or (in the unbalanced case) exclusive block. No separate equality-of-counts constraint by substring type is required—that coupling is instead the central idea of the CS model below. The objective minimises the total number of selected common and exclusive blocks (equivalently $C + D_1 + D_2$ on an unbalanced instance). The candidate set is highly instance-dependent and grows rapidly when the same substring recurs, as it does under a small alphabet; Section 4.4 shows the resulting model becoming prohibitively large on the low-alphabet family.

The CS model. The Common-Strings (CS) model mitigates this by decoupling the choice of *which* substrings to include from the choice of *where* to place them. For each distinct substring t and each position k in S_1 (resp. S_2), a binary variable $x^1_{t,k}$ (resp. $x^2_{t,k}$) indicates whether substring t starting at position k in S_1 (resp. S_2) is selected. As in the CB model, coverage constraints require every position of each genome to belong to exactly one selected block. In addition, the central coupling constraint requires that, for every substring type t, the number of selected occurrences in S_1 equals the number in S_2:

$$\sum_k x^1_{t,k} \;=\; \sum_k x^2_{t,k}.$$

This generalises the common-partition requirement without explicitly enumerating cross-genome pairs. The objective $\sum_{t,k} x^1_{t,k}$ counts the total number of selected blocks in S_1, equal by the coupling constraint to the count in S_2. Romeiro et al. report that the CS model generally performed better than CB on their benchmark—tighter LP-relaxation bounds and faster solving across the tested instance families—an empirical observation for that data rather than a formal dominance relation. We accordingly report CB and CS results separately (Section 4); the full MIP start of Section 3.5 is currently built only for CB, as the substring-based CS model would require an analogous variable-level construction (left to future work, Section 5).

Unbalanced extension. The two models above describe the balanced sub-problem; on an unbalanced instance each genome also carries surplus (indel) genes that no common block can match. Following Romeiro et al. [9], the models add *exclusive*-block variables: for each genome h, every contiguous substring e of a maximal run of genes that occur more often in genome h than in the other is a candidate deletion block, with a binary variable $y^h_{e,k}$ marking whether it is selected at position k. The coverage constraints then range over both common and exclusive variables, so every position of each genome belongs to exactly one selected block—common or exclusive. Minimising the total number of selected blocks therefore yields $C + D_1 + D_2$: the C common blocks (counted once, via the CS coupling) plus the D_1 and D_2 maximal deletion intervals chosen in the two genomes—the quantity minimised on unbalanced instances and aligned against the FPT cost in Section 3.4.

3.4 Comparing Solution Costs Across Solvers

Because the FPT and ILP solvers report their objective differently, a like-for-like comparison of solution *quality* (needed for hypotheses H4–H6) requires care. The FPT in NC mode reports the cost of Equation (2), NC $= C+D_1+D_2-1$, where C is the number of common blocks and D_1, D_2 the numbers of deletion intervals in the two genomes. The ILP minimises, and reports as its objective,

the total block count $C + D_1 + D_2$ (common blocks plus indel blocks in both genomes). The two conventions therefore differ only by the boundary term, and we align them throughout by comparing

$$\mathrm{FPT}_{\mathrm{total}} \;=\; \mathrm{NC} + 1 \qquad \text{against} \qquad \mathrm{ILP}_{\mathrm{obj}} \;=\; C + D_1 + D_2. \tag{3}$$

Implementation note. Three quantities must be kept distinct: the minimised *objective*, the *feasible solution* it certifies, and a human-readable *printed* block list. In the ILP implementation used here, the printed list enumerates the common blocks together with the exclusive (deletion) blocks of only *one* genome, so its line count equals $C+D_1$—not the objective. On a hard instance the two can differ by nearly a factor of two (e.g. a printed count of 599 against an objective of 1234). Throughout, we therefore read the cost from the solver's reported *objective* $C + D_1 + D_2$, which sums the selected block variables over *both* genomes and can be reconstructed from those variables—the C common blocks and the D_1, D_2 deletion intervals—to confirm that objective $= C + D_1 + D_2$; the printed line count is retained only as a debugging field, never as a solution-quality metric. Because that objective sums partition blocks across *both* genomes, it can exceed the per-genome gene count n. A further requirement, discussed in Section 3.1, is that the heuristic's gene-level block count be lifted to a bound that is valid for the *intergenic* problem before it is used as a warm start.

3.5 Warm-Starting the Exact Solvers

Both exact methods accept an externally supplied upper bound on the optimal cost, and both can exploit it in essentially the same way: any branch of the search whose partial cost has already reached or exceeded the supplied bound can be discarded, since it cannot lead to a solution that improves on a bound that is already known to be achievable.

For the FPT algorithm, the warm start installs the heuristic value $\bar{k}$ as an externally certified upper bound; the implementation stores only this cost, not the heuristic's partition. It is therefore not an *incumbent solution* in the usual sense—no materialized feasible partition is associated with it. At each branching step the accumulated cost of the current partial solution is compared against $\bar{k}$, and a branch is pruned as soon as it can no longer *strictly* improve on $\bar{k}$. This preserves the correctness of the reported *objective value*: the algorithm still returns the true optimum whenever it is strictly below $\bar{k}$. It does not, however, guarantee a returned *partition*: whenever the search finds no partition strictly better than $\bar{k}$ within its budget—because $\bar{k}$ is already at or below every partition the FPT can construct in time—strict-improvement pruning discards those partitions and none is materialized, so the FPT reports the bound $\bar{k}$ with an empty solution (the "phantom" of Section 4.5). Retaining the heuristic's complete partition alongside its cost would close this gap, by letting the FPT fall back on that partition when it cannot beat $\bar{k}$; this is the FPT-side future work of Section 5.

For the ILP solver we use two warm-start modes. The weaker one supplies the bound $\bar{k}$ through Gurobi's objective *cutoff*. Because the objective $C + D_1 + D_2$ is integer-valued, we set the `Cutoff` parameter to $\bar{k} + 0.5$: for this minimisation Gurobi discards any subproblem that cannot yield an objective below the cutoff, so integer solutions of value at most $\bar{k}$ are admitted while those of value at least $\bar{k} + 1$ are rejected. The half-integer offset places the threshold strictly between consecutive attainable costs, making the boundary unambiguous regardless of Gurobi's strict/non-strict convention (the solver logs report this value directly, e.g. `Cutoff 864.5`). This is always valid and needs no alignment with the model, but it is only a *filter*: it conveys no solution, merely rejecting every candidate costlier than $\bar{k}$.

The stronger mode, which we implement for the CB formulation, is a full *MIP start*: an explicit, feasible assignment of the CB model's binary variables, with objective value $\bar{k}$, handed to the solver as an initial incumbent via the variables' `Start` attribute, which the solver keeps and tries to improve. The obstacle noted in prior work—mapping a heuristic's *balanced, relabelled* block

decomposition back onto the ILP's original variable indexing—we avoid entirely by *constructing* the feasible partition directly on the ILP's own gene sequences with the greedy-intergenic procedure of Section 3.1. Because that partition is expressed in the model's own coordinates, each of its common blocks maps to an exact candidate block (t, k_1, k_2) of the CB model and each deletion run to an exact exclusive block, yielding a complete, feasible assignment of objective $\bar{k} = C + D_1 + D_2$ that the solver accepts without modification. This construction, and the H6 results that build on it (Section 4.6), are specific to the CB model; the substring-based CS model would require an analogous construction on its own variables, which we leave to future work (Section 5), so we make no claim about a full MIP start for CS. In both modes, the tighter the bound relative to the true optimum, the more of the search space is pruned; only the MIP start also *supplies* a solution. Section 4.6 compares the two modes empirically.

3.6 Experimental Design

The six hypotheses of Section 1 are evaluated in two regimes. The *capped time-to-certification* regime (H1–H3) measures whether and how quickly each solver certifies optimality before an explicit safety limit; the *fixed-budget quality* regime (H4–H6) caps every solver at a common budget T and compares the quality of the best solution returned. Table 1 summarises the six.

Table 1: The six hypotheses. "Regime" distinguishes capped time-to-certification from fixed-budget solution quality (budget T).

	Claim (the stated hypothesis)	Solver(s)	Regime
H1	cold ILP runtime < cold FPT runtime	FPT vs. ILP	capped time-to-cert.
H2	FPT+heuristic runtime < cold FPT	FPT	capped time-to-cert.
H3	ILP+heuristic runtime < cold ILP	ILP	capped time-to-cert.
H4	cold ILP quality > cold FPT quality	FPT vs. ILP	fixed-budget T
H5	FPT+heuristic quality > cold FPT	FPT	fixed-budget T
H6	ILP+heuristic quality > cold ILP	ILP	fixed-budget T

Benchmark families. Two instance families are used. The *small* family (SREV) consists of signed genomes of $n \approx 100$ genes, in matched *balanced* (no indels) and *unbalanced* (indels) variants, parameterised by the occurrence multiplicity O and the reversal percentage P (e.g. `O50_P80`). The *hard* family (`smcisp`) consists of $n = 1000$-gene instances whose key difficulty knob is the *alphabet size* $K \in \{4, 20, 36\}$ distinct gene labels: a small alphabet forces heavy gene repetition. As shown in Section 4, this knob sends the two solvers into opposite failure modes, so it is the most informative axis at scale.

Solvers, heuristics, and warm-start mechanisms. Both exact solvers are run cold and warm. Four heuristic configurations supply warm bounds: the FPT repository's native *MCISP2K* bound, and the three signed MCSP-Algorithms variants *signedGreedy*, *signedCombine*, and *signedPso*. For the FPT the bound is installed as an initial incumbent (Section 3.5); for the ILP the bound is applied as a Gurobi objective *cutoff*—the runtime lever measured in H3—and, for the CB formulation, additionally as a full *MIP start* built from the greedy-intergenic partition on the model's own variables, whose effect on fixed-budget solution quality is the subject of H6 (Section 3.5). The four-heuristic sweep is the *runtime* study—H2 for the FPT and H3 for the ILP cutoff (across both the CB and CS models); the *fixed-budget quality* studies (H5 for the FPT, H6 for the ILP) instead use the single valid greedy-intergenic bound, and the full MIP start is evaluated for the CB model only. Reported runtime is the solver time defined below; speedup is the ratio of cold to warm runtime (> 1 meaning

the warm run is faster). For H3 we also report the honest *end-to-end* cost, charging the heuristic's own runtime to the warm side.

Timing quantities and budget protocols. For each run we distinguish three timing quantities: the *heuristic time* (running the warm-start heuristic; Table 3), the *construction time* (building the solver's data structures—for the ILP, generating the candidate blocks and assembling the model; for the FPT, its $O(n^2)$ preprocessing), and the *solver time* (the optimization/search phase proper, as reported by Gurobi's own runtime and by the FPT's internal timer). We also record the *wall-clock* time—construction plus solver time—with a monotonic timer around the whole invocation. Every time limit acts on the *solver* phase only, so we make the distinction between two budget protocols explicit. Under the *solver-budget protocol* the budget bounds the solver time alone—construction and the heuristic run beforehand and are not charged against it—and this is the protocol under which all six hypotheses are stated. Under the *pipeline-budget protocol* the same wall-clock budget would instead cover construction, the heuristic, and the solver together; this is the fairer end-to-end view of the methods as a user would run them, and we report it for H3 (Section 4.3) and discuss it for the hard low-alphabet ILP, where—as Section 4.4 quantifies—model construction alone is on the order of the solver budget itself, so solver-reported time materially understates the ILP's end-to-end cost. Every claim of "the same budget", speedup, or dominance below is made under the solver-budget protocol unless it is explicitly labelled end-to-end; a full pipeline-budget sweep across all hypotheses is left to future work (Section 5).

Time limits, and how the heuristic is charged. Every solver invocation—cold or warm, FPT or ILP—runs under an explicit per-instance solver-time limit, and the warm and cold runs of a given instance always share the *same* limit, so warm-versus-cold is a like-for-like comparison. In the capped time-to-certification regime the FPT is given a 300 s budget (enforced inside its own timer) and the ILP a 3600 s limit; the latter is never approached, since the cold ILP's solver phase finishes every instance in seconds, so this cap only bounds the FPT's non-terminating runs on the hard instances (itself a reported finding). In the fixed-budget regime the limit *is* the independent variable, $T \in \{10, 60, 300\}$ s. Consistent with the solver-budget protocol, the heuristic's own runtime is *not* deducted from T: the heuristic runs beforehand and the solver then receives its full limit, so the fixed-budget quality results—including the FPT "phantom" of Section 4.5—reflect what each solver does with a complete solver budget. The only place the heuristic's time is folded back in is the *end-to-end* comparison of H3, which explicitly charges it to the warm side. One further caveat for the hard instances: the FPT polls its timer only *between* branch nodes, and a single node at $n = 1000$ can run for minutes, so the effective stop time overshoots T; this can only make the FPT look better, and does not affect the reported verdicts.

Valid bounds. Each heuristic's gene-level block count is lifted to a bound valid for the intergenic problem before it is used (Section 3.4); we use the greedy-intergenic lift, which builds an IR-consistent matching and so always supplies a feasible upper bound.

Reproducibility. All experiments were run on a single MacBook Pro with an Apple M3 Max processor (12 performance + 4 efficiency cores) and 64 GB of RAM, under macOS 26.5 (Darwin 25.5.0). The FPT solver is implemented in Java and was compiled and executed with the OpenJDK HotSpot JVM 25.0.2 (LTS); each instance was solved in its own JVM process, one at a time, with an 8 GB heap (`-Xmx8G`) and a single-threaded branching search, under the per-instance time limits stated above. The ILP experiments use Python 3.12 with `gurobipy` and Gurobi 12.0.3; the only non-default Gurobi parameters are the time limit (regime-dependent) and `Threads`=1, with the random seed left at its default—so, being single-threaded, both the Gurobi runs and the FPT search are deterministic. The

heuristics—*signedGreedy*, *signedCombine*, *signedPso*, and the FPT's native *MCISP2K* bound—come from a separate code base; *signedPso* is the only stochastic component and was run with a fixed random seed recorded in the artifact. The benchmark instances are produced by the repository's deterministic generators with fixed seeds, so the instance set is fully regenerable; the accompanying manifest lists every instance with its size, alphabet size, balance status, and the solver configurations in which it appears (the instance families are those of Section 3.6). The artifact also provides the solver and pipeline source, the command-line interfaces and the intermediate file formats used to communicate bounds between the heuristic, FPT, and ILP components, and the raw per-run logs together with the scripts that produce every table and figure in this paper.

4 Results

We report the six hypotheses of Table 1 in turn. Each subsection states the claim, recalls the mechanism that motivates it, presents the measurements, and closes with a verdict. Section 4.7 then draws the results together. Throughout, solution quality is compared using the aligned cost of Equation (3), with warm bounds taken from the valid greedy-intergenic lift, when necessary (Section 3.6). Aggregate runtimes are reported as *means* over the instances of each group, with the *median* given alongside; the two diverge only when a handful of timeouts skew the mean, and we flag those cases explicitly. Every speedup is computed *per instance* as the ratio of the two runtimes on that instance ($t_i^{\mathrm{FPT}}/t_i^{\mathrm{ILP}}$ for H1, $t_i^{\mathrm{cold}}/t_i^{\mathrm{warm}}$ for the warm-start studies) and then aggregated across the group; it is therefore *not* the ratio of the aggregate runtimes shown in the adjacent columns, which the heavy right skew makes materially smaller (for the balanced group, 25× against $0.026/0.0016 \approx 16\times$). A solver that reaches its time limit contributes its elapsed time at that limit—including the FPT's between-node overshoot on the hard instances—so timed-out instances yield very large ratios that lift the mean far above the median. Because these ratio distributions are heavy-tailed, we summarise them by both the mean and the (more representative) median, and give the interquartile range as a paired spread.

Two further conventions follow from the structure of the data. First, the time-to-certification measurements are *right-censored*: when a solver reaches its budget without certifying optimality, its time is only a lower bound, so a mean over such values would be biased. We therefore summarise this regime by *completion rates*—how many instances each solver certifies within the budget (e.g. the FPT certifies 0/12 hard instances)—rather than by a mean solve time. Second, the fixed-budget quality comparisons (H4–H6) are reported as *paired*, per-instance differences or relative gaps between the two costs on the same instance, not as differences of separately-aggregated means. Finally, because the benchmark is a *structured* set of chosen configurations and fixed seeds rather than a random sample of all possible instances, every spread or rate we report should be read as uncertainty over the tested instances and seeds, not as a universal population guarantee; correspondingly, the hypotheses below are directional predictions we assess on this benchmark rather than claims of statistical significance over an instance population.

4.1 H1: The Cold ILP Is Faster Than the Cold FPT

Hypothesis. With no warm start, the ILP has lower execution time than the FPT.

The two solvers embody opposite strategies—exhaustive parameterised branching versus branch-and-cut over a pre-enumerated block model—so their cold runtimes measure which strategy copes better with the structure of real instances. Table 2 aggregates 22 *runs*—each run being one solver executed over one instance-generation configuration's full instance set—covering 102 instances in total (90 small and 12 hard at $n = 1000$); Figure 9 plots the mean runtimes on a logarithmic scale.

Table 2: H1: cold FPT versus cold ILP. Each runtime cell reports the *mean* with the *median* in parentheses, in seconds of solver execution time; "exact" counts instances solved to a certified optimum within budget; "speedup" is the *per-instance* ratio $t_{\mathrm{FPT}}/t_{\mathrm{ILP}}$, aggregated as mean/median (not a ratio of the tabulated means). Hard instances have $n = 1000$ genes. On the small unbalanced group four instances hit the 300 s FPT limit, which inflates the mean far above the median—the median is the representative figure there. The FPT's nominal per-instance limit is 300 s; because it polls its timer only between branch nodes, a single hard-instance node can overshoot it (the K=36 unbalanced run reached 3505 s of wall-clock before stopping), so the hard-instance entries mark censored, non-certifying runs rather than 300 s completions.

Group	#inst	FPT exact	ILP exact	FPT runtime (s) mean (median)	ILP runtime (s) mean (median)
Small, balanced	45	45/45	45/45	0.026 (0.022)	0.0016 (0.0011)
Small, unbalanced	45	41/45	45/45	31.8 (0.42)	0.0030 (0.0024)
Hard, K=20 (bal.)	3	0/3	3/3	timeout	1.50 (1.45)
Hard, K=20 (unb.)	3	0/3	3/3	300 (at limit)	0.16 (0.16)
Hard, K=36 (bal.)	3	0/3	3/3	timeout	0.092 (0.089)
Hard, K=36 (unb.)	3	0/3	3/3	3505 (overshoot)	0.10 (0.083)

Speedup $t_{\mathrm{FPT}}/t_{\mathrm{ILP}}$, mean/median [IQR]: bal. 25/25 [15, 33], unb. 16 900/151 [19, 2536].
Hard (n=1000, all 12): FPT feasible 2/12, certified 0/12; ILP exact 12/12 in < 2 s.

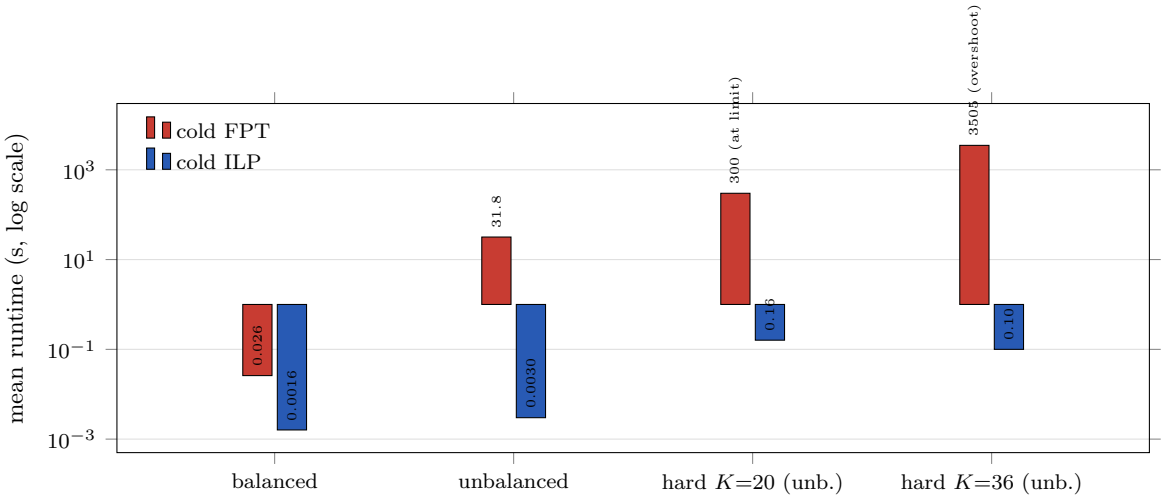

Figure 9: H1: mean cold *solver* time of the FPT and ILP across the displayed instance groups. The vertical axis is logarithmic, but every bar label shows the original time in *seconds* (e.g. 0.026, 31.8, 300, 3505), not its logarithm. The gap widens from $\sim 25\times$ on balanced small instances to a qualitative collapse on the hard $n = 1000$ instances, where the FPT reaches no complete solution: its two hard bars are censored at the 300 s nominal limit (on K=36 a single branch node overshot to 3505 s before the timer was next polled), while the ILP finishes in under two seconds. The two hard bars show the *unbalanced* groups; on the balanced hard groups the FPT likewise certifies none and the ILP stays exact (Table 2). The small-unbalanced FPT bar reflects the mean, lifted by four 300 s timeouts; the typical (median) time there is only 0.42 s (Table 2).

Discussion. The ILP is faster on every axis. On balanced small instances both solvers are sub-second and exact, yet the ILP is already $\sim 25\times$ faster; on unbalanced small instances the FPT degrades sharply (median $151\times$ slower, and a mean of $\sim 16\,900\times$ inflated by four instances that hit the 300 s limit). At $n = 1000$ the difference becomes qualitative. For the high-alphabet families included in this time-to-certification comparison—$K = 20$ and $K = 36$, the only $n = 1000$ families here (6 instances each, balanced and unbalanced)—the FPT produced no solution on 10 of the 12 hard instances and no *certified* optimum on any, whereas the ILP certified all 12 to optimality in under two seconds of optimization time. The low-alphabet $K = 4$ family is deliberately excluded from this statement—there the ILP's model generation and search are substantially harder—and is analysed separately in Section 4.4. We verified that the FPT failure is compute-bound, not memory-bound (live heap $\sim$ 3–4 GB of 8 GB, GC $< 2\%$, CPU pinned): at $n = 1000$ the branching tree is simply too large to reach a first complete leaf in time. The timer overshoot to 3505 s only ever helped the FPT and it still lost by four orders of magnitude.

Verdict: H1 is strongly supported. For this set of instances, there is no axis on which the cold FPT is competitive with the cold ILP on execution time.

4.2 H2: Warm-Starting the FPT Does Not Reduce Its Runtime

Hypothesis. Seeding the FPT with a heuristic upper bound lowers its runtime.

A warm bound installs an incumbent at node 0, so the pruning cutoffs fire immediately rather than only after the first complete leaf. We measured this over 64 small-instance runs (320 instances) across the four heuristics and both cost modes, plus hard $n = 1000$ samples.

How the four heuristics are compared. Each heuristic produces a partition that is turned into a single valid intergenic upper bound $\bar{k}$ before it is handed to the solver, so the four configurations are compared on a common footing—the same objective $C+D_1+D_2$ (Section 3.4) and the same warm-start interface. Two of them are already intergenic-aware and therefore need no formula lift: the native MCISP2K bound is read directly from an intergenic partition, and the greedy-intergenic construction builds an IR-consistent matching from scratch; the purely gene-level variants (signedCombine, signedPso) have their block count lifted to a valid intergenic bound as described in Section 3.1. On the FPT side the resulting valid bounds coincide in tightness—all four sit a similar distance above the optimum—which is why the four series are visually indistinguishable in Figure 10. Their *running times*, however, differ by orders of magnitude and do not enter the FPT bound cheaply everywhere: on the small instances the heuristics are cheap (from < 0.1 s for combine to $\sim$13 s for PSO), whereas on the $n = 1000$ instances their times span approximately 5 s (combine), 17 s (MCISP2K), 748 s ($\sim$12 min, greedy), and 2060 s ($\sim$34 min, PSO). This gap is immaterial to the FPT solver-time comparison but decisive for the honest end-to-end accounting of H3 (Table 3).

Table 3: Mean wall-clock time for each heuristic to produce one warm bound. On small instances all are cheap; at $n = 1000$ they diverge by three orders of magnitude, which is what makes the end-to-end warm-start accounting of H3 so unfavourable for greedy and PSO in particular.

Heuristic	small ($n \approx 100$)	hard ($n = 1000$)
signedCombine	< 0.1 s	≈ 5 s
MCISP2K	≈ 5 s	≈ 17 s
signedGreedy	≈ 0.3 s	≈ 748 s ($\sim$12 min)
signedPso	≈ 13 s	≈ 2060 s ($\sim$34 min)

Across the small instances the mean FPT-solve speedup is $1.04\times$ (median $1.01\times$), and on the

unbalanced instances—the ones that actually take seconds to minutes—it is only $\approx 1.0\times$ (Figure 10, FPT series). The four heuristic configurations give near-identical ratios (their FPT small-instance means span only 1.03–1.05), so each bar in that figure aggregates them as a single mean per solver/formulation; the reason is the one just given. On the hard $n = 1000$ instances the speedup is exactly $1.00\times$: both the cold and the warm run exhaust the 300 s budget. The reason differs by regime. On the small instances, where the cold FPT already finds a good complete solution early, the initial bound leaves little additional pruning opportunity. On the hard instances the situation is the opposite: within the budget the cold FPT often materialises *no* complete partition at all (Section 4.1), and the valid intergenic bound is looser than the optimum, so the cost-only seed—which conveys a number but not a partition—has nothing to turn into a solution; here it is the absence of a stored heuristic partition, not any early discovery by the cold FPT, that leaves the warm start inert.

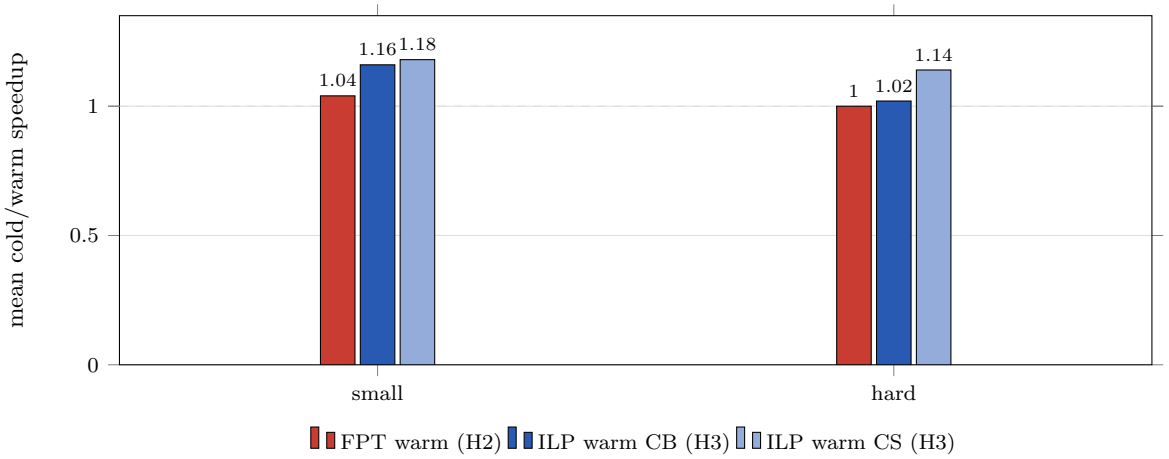

Figure 10: Mean cold/warm *solver-only* speedup for each exact-solver formulation (dashed line marks parity at $1.0\times$; values > 1 mean the warm run is faster). Each bar is the mean over the four heuristic configurations, which agree to within a few percent (e.g. the FPT small-instance means span 1.03–1.05); medians are within 0.03 of the means shown. Every configuration sits within a few percent of parity—no heuristic gives either solver a meaningful runtime reduction, and the gain all but vanishes on the hard $n = 1000$ instances.

Verdict: H2 is not supported. Heuristic warm start gives the FPT no meaningful runtime reduction ($\approx 1.0\times$), for any heuristic, at any scale.

4.3 H3: Warm-Starting the ILP Does Not Reduce Its Runtime

Hypothesis. Seeding the ILP with a heuristic cutoff lowers its runtime.

The cutoff lets branch-and-cut discard any subproblem whose relaxation already exceeds the bound. We measured 163 runs (739 instances) across both the CB and CS formulations, intergenic on and off. Two findings emerge, both pointing the same way.

First, the *solver-only* speedup is marginal—mean $1.16\times$ (CB) / $1.18\times$ (CS) [medians $1.13\times$ / $1.15\times$], and only $1.02\times$ / $1.14\times$ on the hard $n = 1000$ instances (Figure 10, ILP series)—because the cold ILP already solves the high-alphabet instances—including $n = 1000$ at $K \in \{20, 36\}$—in seconds, leaving almost no search tree for a cutoff to prune. Second, the honest *end-to-end* comparison, which charges the heuristic's own runtime to the warm side, is decisively negative: the

median of $t^{\text{cold}}/(t^{\text{heur}} + t^{\text{warm}})$ is ≈ 0.011 (mean ≈ 0.020), i.e. warm start is about $90\times$ *slower* once the heuristic is paid for (Figure 11)—and far worse for the expensive heuristics, since the heuristic's own runtime ranges from $\sim$5 s (combine) to $\sim$12 min (greedy) and $\sim$34 min (PSO) at $n = 1000$.

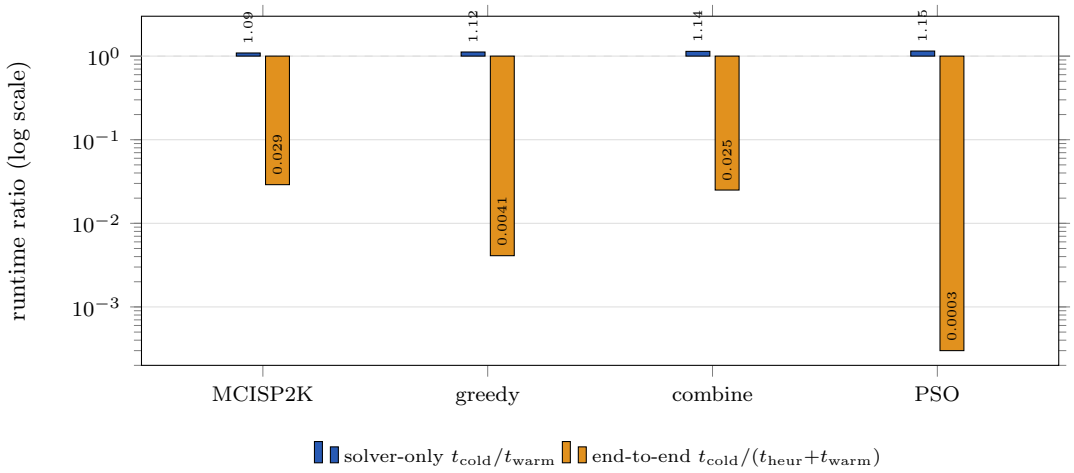

Figure 11: H3, *per heuristic*: the ILP cutoff gives a marginal solver-only speedup $t_{\text{cold}}/t_{\text{warm}} \approx 1.1$ for all four heuristics (near the parity line at 1.0), but the honest end-to-end ratio $t_{\text{cold}}/(t_{\text{heur}}+t_{\text{warm}})$ falls far below parity and—crucially—differs by orders of magnitude across heuristics, because it is dominated by the heuristic's own runtime (Table 3): from $\approx$0.025–0.029 for the cheap MCISP2K and combine bounds down to $\approx$0.004 (greedy) and $\approx$0.0003 (PSO), i.e. roughly $35\times$ to $3000\times$ slower end-to-end. Pooling all four gives a median of 0.011 ($\sim 90\times$ slower), meaningful only alongside the per-heuristic values shown here. Bar labels are the ratios themselves, not their logarithms.

Verdict: H3 is not supported. The cutoff gives at best a marginal solver-only gain and is net $\sim 90\times$ slower end-to-end once the heuristic is paid for.

4.4 H4: Under a Fixed Budget, ILP Quality Exceeds FPT Quality—Except at Low Alphabet

Hypothesis. Given the same solver budget T, the ILP returns a better (fewer-block) solution than the FPT.

On small instances the answer is a clean yes-or-tie. For this fixed-budget quality comparison we use three of the small configurations of Section 3.6, fixed in advance and independently of any solver output: the balanced control `O20_P60` and the two unbalanced configurations `O50_P60` and `O50_P80`, each consisting of the five generated instances of that configuration (so "5/5" below refers to those five, not a sample drawn from a larger pool). We do not repeat the full 18-configuration small sweep of H1 here, because every small instance is solved to the optimum by *both* solvers regardless of budget—so additional small configurations add no discriminating power, and the quality differences this hypothesis targets appear only on the hard $n = 1000$ family; the per-instance identifiers and generation seeds are listed in the artifact. On the balanced control both solvers are optimal (5/5 ties); on each unbalanced configuration the ILP is strictly better on 3/5 instances and ties on 2/5, at *every* budget $T \in \{10, 60, 300\}$ s. The FPT's quality is time-flat: it finds its best solution quickly and then only tries—often failing—to *prove* optimality, so extra time does not close the gap.

The hard instances reveal a reversal driven by the alphabet-size knob K (Table 4). For $K \in \{20, 36\}$ the ILP is optimal in seconds while the cold FPT finds nothing, so the ILP wins by default.

But for $K = 4$—a four-letter alphabet, hence extreme gene repetition—the ILP's candidate-block model explodes ($\sim$ 167M nonzeros), and building it alone takes $\sim$300 s—on the order of the solver budget itself, so the end-to-end wall-clock cost is roughly 600 s at $T = 300$ s and the solver-budget figures below understate the ILP's true cost on this family. Even so it stalls at a $\sim 50\%$ optimality gap with a near-trivial incumbent, while the FPT returns a genuinely better solution. Figure 12 shows the three $K = 4$ sub-instances: the ILP's own incumbent ($\approx$ 1234 blocks) is about 41–44% larger than the FPT's feasible solution ($\approx$ 858–873)—a ratio of roughly 1.41–1.44—and both bracket the true optimum together with the ILP's proven lower bound ($\approx$ 608–626).

Table 4: H4 on hard $n = 1000$ instances, by alphabet size K, at $T = 300$ s. All entries are *cost bounds* in total blocks $C+D_1+D_2$: the cold FPT column is the best feasible solution it found (an upper bound on the optimum), and the cold ILP column is its best incumbent, with the optimality gap to the ILP's own proven lower bound shown in parentheses; "–" means no solution was found within the budget. Lower is better.

Instance (K)	cold FPT	cold ILP	Winner
K=4 (extreme repeats)	**858–873**	1234–1236 (gap $\approx$ 50%)	**FPT**
K=20	– (no solution)	optimal, < 2 s	ILP
K=36	– (no solution)	optimal, < 2 s	ILP

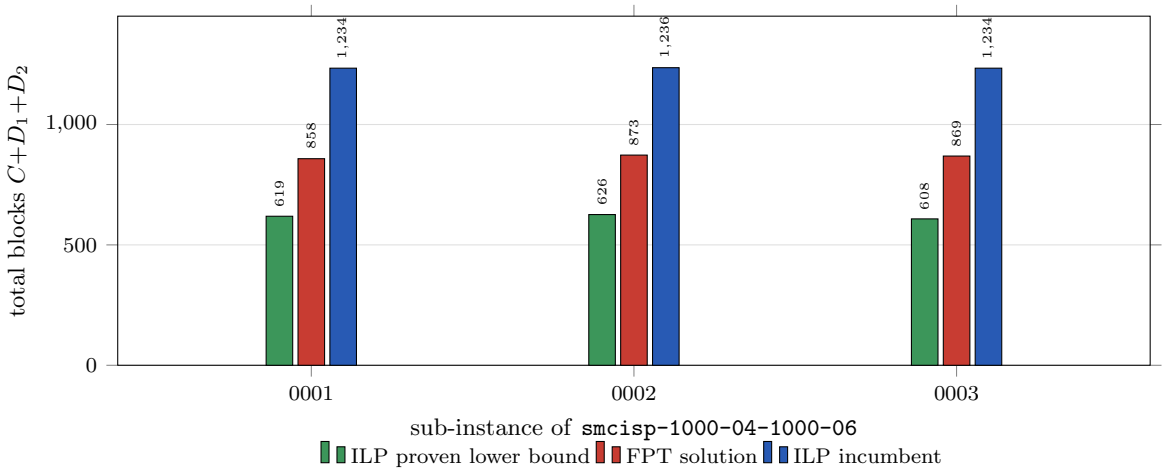

Figure 12: H4 at low alphabet (K=4). The optimum lies between the ILP's proven lower bound (green) and the FPT's feasible solution (red); the ILP's own incumbent (blue) is substantially worse—about 42% larger—because its block model explodes under heavy repetition. The FPT's solution is the best certified-feasible one known for these instances.

Verdict: H4 is partially supported—the outcome is instance-structure-dependent. The ILP returns equal-or-better solutions on all small instances and wins decisively at high alphabet; but at low alphabet its model explodes and the FPT wins. The two solvers are *complementary* under a fixed budget rather than strictly ordered.

4.5 H5: Warm-Starting the FPT Does Not Improve Its Solution Quality

Hypothesis. Under a fixed budget, a warm-started FPT returns a better solution.

The answer splits by regime. On the small instances, where the cold FPT already reaches the optimum within budget, the warm run returns exactly the same solution: the valid intergenic bound is looser than what the FPT already finds, so warm = cold throughout.

The hard instances require a careful reading, because there the warm run *reports* a strictly smaller cost than the cold run—and yet this is not a genuine FPT improvement. On the three sub-instances 0001/0002/0003 of the low-alphabet family `smcisp-1000-04-1000-06` (the $K = 4$ case), the cold FPT returns 858/873/869 while the warm run reports 858/849/866; on the $K = 20$ instance the cold FPT returns 1002 and the warm run 997 (Table 5, upper block; costs are the aligned $C+D_1+D_2$ of Equation (3)). The explanation is the warm-start mechanism itself: the FPT is seeded with only the heuristic's *cost* $\bar{k}$, recorded as a bound, not as a partition. When $\bar{k}$ is below the cost of any complete solution the FPT can assemble within the budget—as it is on these hard instances—every branch is pruned against $\bar{k}$ and the search terminates without ever materialising a partition, so the FPT simply echoes $\bar{k}$ back as an empty "phantom" solution. The reported 849, 866, and 997 are therefore the heuristic's own (valid) bounds, not solutions the FPT constructed; the FPT itself produces no better partition than it does cold. Notably these heuristic bounds are genuinely good—997 on $K = 20$ is below the non-exact cold FPT's 1002, and we verified they are valid upper bounds ($\geq$ the true optimum)—which is precisely why a warm start that conveyed the *partition*, not just the number, could help here (Section 5). This forces a distinction between two notions of the "solution returned": (i) the best feasible partition available to the complete warm-start *pipeline*—which includes the heuristic's own partition, itself a genuine feasible solution—and (ii) the best partition the FPT *search itself* materialises. The phantom affects only (ii); under (i) the heuristic's partition is a real solution and must be counted.

Verdict: H5 depends on how the returned solution is defined. Under metric (ii)—the best partition the FPT search itself materialises—a warm start never improves the cold result: on the tractable instances the valid bound is too loose to change anything, and on the hard instances the bound-only seed is echoed as an empty phantom rather than turned into a partition, so by this measure H5 is *not supported*. Under metric (i)—the best feasible partition available to the complete pipeline—the heuristic's own partition already counts as a solution and is better than the struggling non-exact cold FPT on three of the four hard cases (849, 866, and 997 against the cold FPT's 873, 869, and 1002; on the fourth, sub-instance 0001, the heuristic's 864 is worse than the cold FPT's 858, so the pipeline keeps the FPT's partition). By that measure the complete method *does* return a better solution on the hard instances—though the gain is the heuristic's, surfaced by the pipeline, not produced by the FPT's search; the two metrics coincide on every tractable instance (both optimal). Once the FPT is extended to retain the seed as a materialised initial partition (Section 5), metric (i) becomes the FPT's own outcome and the more meaningful practical measure.

4.6 H6: A Full MIP Start Preserves and Certifies a Better Heuristic Incumbent, Whereas a Cutoff Does Not

Hypothesis. Under a fixed budget, a warm-started ILP returns a better solution.

On every instance that the cold ILP solves to optimality—all small instances and the hard $K \in \{20, 36\}$ ones—there is no room for a warm start to improve quality: whether we supply a cutoff or a full MIP start, warm = cold (Table 5, lower block). The decisive case is the one family where the ILP genuinely struggles—the low-alphabet $K = 4$ instances (`smcisp-1000-04-1000-06`), where within the T=300 s budget the cold ILP reaches only a poor incumbent (objective ≈ 1234) at a $\sim 50\%$ gap. There the two warm-start modes behave in *opposite* ways.

A bound-only cutoff hurts the solver, not the pipeline. The heuristic supplies a valid bound—we verified it is the cost of a real feasible partition (e.g. 864, well below the cold incumbent 1234). But a Gurobi objective cutoff is a *filter*, not a seed: it discards every solution at least as costly

as the cutoff, including the crude incumbent the solver's own rounding heuristic would otherwise keep, and on this enormous model the solver cannot then construct anything below the cutoff in the remaining time. Two states must therefore be reported separately: the *solver* incumbent is *none* (cost $= \infty$), whereas the *pipeline* incumbent is still the heuristic's feasible partition of cost 864—the very solution whose value defined the cutoff. The degradation is thus in the *interface*, not the available solution: relative to its own cold run the solver ends worse off (no incumbent instead of 1234), yet the complete method still holds a partition of cost 864, better than cold. The cutoff is simply a poorer communication channel than a MIP start—it fails to *import* the known feasible solution, but it does not make that solution cease to exist.

A full MIP start preserves and certifies the heuristic incumbent. Supplying the same greedy-intergenic partition to the CB model as an explicit feasible incumbent (objective 864)—rather than only its value—makes the heuristic's solution available to the solver: the warm run *reports* 864 in place of the cold run's internally generated incumbent 1234, a $\sim 30\%$ better returned solution, and, sharing the same proven lower bound of 619, certifies it to within a $\sim 28\%$ gap (down from the cold run's $\sim 50\%$) (Figure 13). The non-intergenic $K = 4$ instance behaves identically (warm 857 vs cold 1201; gap 44% vs 60%). This is exactly the mechanism of Section 3.5: the cutoff only prunes, whereas the MIP start also *supplies* the solution the huge model cannot build on its own within the budget.

Two honest qualifications keep this result in proportion. First, within the fixed budget the ILP does not *improve* on the partition it is handed: the warm incumbent is exactly the seeded 864, and the solver explores a single node (the root LP alone consumes the 300 s on this ∼167M-nonzero model). The gain over cold is therefore the heuristic's own solution *surfaced and certified*—the ILP contributes the lower bound 619, proving that solution within 28% of optimal, not a better incumbent. Second, this is a budget-limited outcome, not a ceiling: given more time the branch-and-bound can start from the seeded 864 and search *below* it—something neither the cold run (stuck at 1234) nor the bound-only cutoff (no incumbent) can even begin to do—so a larger budget is the natural way to turn "surfaced and certified" into "genuinely improved". Closing the gap from the other side is harder: pushing the lower bound with aggressive settings (`MIPFocus=2`, `Cuts=2`) did not help—the extra cut generation exhausts the budget before any bound is even established—so certified optimality at low alphabet remains out of reach here.

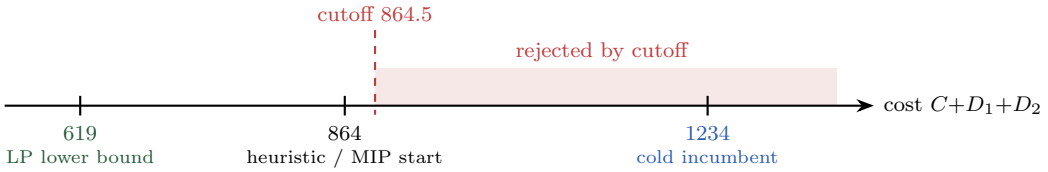

Figure 13: H6 on the struggling low-alphabet (K=4, intergenic) instance, on the cost axis. The greedy-intergenic heuristic supplies a feasible partition of cost 864, and the LP relaxation proves a lower bound of 619. A *bound-only cutoff*—Gurobi's `Cutoff` parameter set to $\bar{k}+0.5 = 864.5$—rejects every integer solution of cost ≥ 865, including the cold ILP's own incumbent (1234)—so the solver ends with no incumbent of its own, even though the heuristic's feasible partition of cost 864 still exists in the pipeline. A *full MIP start* instead installs the 864 partition as the incumbent, so the warm run *preserves and certifies* it—returning 864 and bracketing the optimum in $[619, 864]$ (gap 28% vs the cold run's 50%; within the budget the ILP does not search below 864); the non-intergenic instance is analogous (857 vs 1201, gap 44% vs 60%).

Table 5: H5 (FPT) and H6 (ILP) fixed-budget quality with valid greedy-intergenic warm bounds. Where the cold solver is already optimal there is no room to improve (warm = cold). On the hard low-alphabet instances the two solvers differ: the FPT (bound-only) echoes the bound as an empty "phantom" partition—the lower H5 warm values are feasible partitions the heuristic makes available to the pipeline, not ones the FPT search materialises (Section 4.5); the ILP, given only a *cutoff*, stores no incumbent of its own, but given a full *MIP start* improves markedly. Costs are total blocks $C+D_1+D_2$; the three K=4 values are sub-instances 0001/0002/0003 of `smcisp-1000-04-1000-06`.

	Group	Warm vs. cold (outcome)
H5	Small (bal. + unb.)	warm = cold — no room (cold optimal)
	Hard K=4	cold 858/873/869, warm 858/849/866 — lower values are the heuristic bound echoed as a phantom
	Hard K=20	cold 1002, warm 997 — phantom (heuristic bound, empty partition)
H6	Small (all)	warm = cold — no room (optimal)
	Hard K=20	warm = cold (855) — no room (optimal)
	Hard K=4 (cutoff)	cold 1234, warm ∞ (solver) — cutoff imports no incumbent; the pipeline still holds the heuristic's 864
	Hard K=4 (MIP start)	cold 1234, warm **864** — full MIP start improves (∼30%; gap 28% vs 50%)

Verdict: H6 is nuanced, and turns on what "improves" means. A bound-only cutoff does not improve the ILP's fixed-budget quality—and on the struggling low-alphabet family it hurts the solver, which returns no incumbent (though the pipeline still holds the heuristic's 864). A full MIP start, built from the greedy-intergenic partition on the CB model's own variables, makes the warm run *report* a markedly better solution there (864 vs 1234, gap 28% vs 50%). Strictly, this shows that the MIP-start interface *preserves* the better heuristic incumbent and lets the ILP *certify* its quality; within the budget the ILP does not itself generate an incumbent below the seeded 864. That the returned fixed-budget solution improves at all—the one regime where warm-starting pays off—is the paper's warm-start contribution.

4.7 Synthesis

Table 6 collects the six verdicts. Read together, they tell a single coherent story with two threads.

Table 6: Summary of the six hypotheses and their verdicts.

	Hypothesis	Verdict
H1	cold ILP faster than cold FPT	**strongly supported** ($\sim 25\times$ to several orders of magnitude)
H2	FPT warm start reduces runtime	not supported ($\approx 1.0\times$)
H3	ILP warm start reduces runtime	not supported ($\sim 90\times$ slower end-to-end)
H4	cold ILP better quality than FPT	partially supported (complementary regimes)
H5	FPT warm start improves quality	depends on definition (FPT output: phantom; pipeline: heuristic partition)
H6	ILP warm start improves quality	nuanced: cutoff hurts; **MIP start preserves** (864 vs 1234, K=4)

Thread 1: a bound-only warm start does not help—but a full MIP start does. Every *bound-only* warm-start result (H2, H3, H5, H6-cutoff) follows from two facts about *how* the bound is used, not from any absence of intergenic-aware heuristics—two of the four (the native MCISP2K bound and the greedy-intergenic construction) are fully intergenic-aware and supply the bound

directly. First, wherever the cold solver is fast—all small instances, and the ILP even at $n = 1000$ for $K \in \{20, 36\}$—it already reaches a good or optimal solution before a bound could help, so the valid (necessarily conservative) bound has essentially nothing to prune: warm equals cold in quality and $\approx 1.0\times$ in time, and once the heuristic's own runtime is charged in, warm start is net slower. Second, on the hard low-alphabet instances where the solver *does* struggle, a bound-only warm start still cannot help, because it conveys only a *number*—an incumbent cost to the FPT, an objective cutoff to the ILP—and not a solution. Given only the number, the FPT prunes every branch and echoes the bound back as an empty phantom partition (H5), while the ILP discards its own fallback incumbent against the cutoff and finds nothing below it in time (H6), returning ∞. In both cases the heuristic in fact *knows* a better solution than the struggling solver produces—its bound is a valid, and often tighter, feasible cost—but the bound-only interface cannot transmit it. Acting on this, we replace the cutoff with a full *MIP start* that hands the ILP the heuristic's entire partition, constructed on the CB model's own variables; on the struggling K=4 family—exactly the case the channel was blocking—this lets the warm run return (and the ILP certify) a solution of 864 instead of 1234, roughly halving the optimality gap (H6, Figure 13). The channel, not the heuristics' intergenic-awareness, was the obstacle; closing it is what finally makes a warm start pay off.

Thread 2: the FPT and ILP are complementary, not ordered. H1 and H4 show the ILP dominating the FPT on runtime everywhere and on quality almost everywhere—the one exception being the low-alphabet, high-repeat regime ($K = 4$), where the ILP's candidate-block enumeration explodes and the FPT delivers the better solution. Within this synthetic benchmark, alphabet size—and the associated level of gene repetition—is strongly associated with this crossover: the ILP performs best on the tested $K = 20$ and $K = 36$ families, whereas the FPT returns better feasible solutions on the three tested $K = 4$ sub-instances. This motivates retaining both approaches—and testing the crossover on further generators, seeds, sizes, and real genomic data—rather than establishing a universal solver-selection rule.

5 Conclusion

This work set out to evaluate whether fast common-substring heuristics can meaningfully accelerate two exact solvers—an FPT branching algorithm and an ILP model—for the signed, intergenic-region-aware minimum common partition problem, by supplying each with a certified upper bound before the exact search begins. The methodology of Section 3 establishes that a heuristic bound *for the appropriate problem* is provably valid for this purpose, and describes two complementary ways of tightening it: simulating the exact solver's own block-merging rule after the fact, or making the heuristic itself sensitive to intergenic-region compatibility while it constructs a matching. The six hypotheses of Section 4 then answer the three questions this study posed.

First, *how much does a warm start help on the tested synthetic instances?* The answer depends on both the instance family and *what* is communicated to the solver. A *bound-only* warm start helps essentially nowhere: for the FPT the runtime speedup is $\approx 1.0\times$ (H2) and the fixed-budget quality is unchanged (H5); for the ILP the solver-only speedup is a marginal 1.16–1.18× that is erased—by roughly two orders of magnitude—once the heuristic's own runtime is charged to it (H3), with fixed-budget quality unchanged where the solver is tractable and, on the hardest family, degraded to no solver incumbent at all (H6, cutoff). A full *MIP start*, by contrast, helps precisely where it matters: on the low-alphabet $K = 4$ family it lets the warm run return (and certify) a solution of 864 instead of 1234 (∼30% better), roughly halving the optimality gap (H6). Second, *why the difference?* Where the cold solver is fast—all small instances, and the ILP even at $n = 1000$ for $K \in \{20, 36\}$—it reaches a good or optimal solution before a bound could help, so the valid (and necessarily conservative) bound has nothing to prune. Where the solver struggles—the low-alphabet $n = 1000$ family—a

bound-only warm start still does not help here, because it transmits only a *number* (an incumbent cost or an objective cutoff) and not a solution: the FPT then echoes the bound as an empty phantom partition and the ILP, given only a cutoff, discards its own incumbent and returns none. Supplying the heuristic's whole partition as a MIP start removes exactly this obstacle for the ILP. Third, *how do the two solvers compare cold?* The ILP dominates the FPT on runtime by one to five orders of magnitude (H1) and on fixed-budget quality almost everywhere (H4)—with the sole, sharp exception of low-alphabet, high-repeat instances, where the ILP's candidate-block model explodes and the FPT returns the better solution. On this benchmark the two solvers are thus complementary rather than ordered, and the crossover—observed here between the $K \in \{20, 36\}$ families and the three $K = 4$ sub-instances—justifies maintaining both; whether it generalises to other generators, sizes, and real genomic data remains to be tested.

The practical implication is twofold. In the tested configurations, adding a *bound-only* interface did not provide a favorable end-to-end trade-off for either exact solver: it is redundant where the solver is already fast and inert—or harmful—where it is not. But the more valuable investment, a better *channel*, does pay off: constructing the heuristic's partition directly on the CB model's own variables (the greedy-intergenic construction of Section 3.1) and handing it over as a full MIP start turns the ILP's worst warm-start case—no incumbent at all—into its best, a solution $\sim 30\%$ better than the cold run. The same construction thus supplies both the valid bound and the feasible partition the MIP start needs.

From bound to solution, and what remains open. The decisive lever was *what* is communicated to the solver, not only how tight it is. Replacing the bound-only cutoff with a full MIP start—the heuristic's entire partition, expressed as a feasible assignment of the CB model's binary variables—turned the ILP's failure mode on the low-alphabet $K = 4$ family (no incumbent within budget) into its best warm-start outcome (objective 864 against the cold run's 1234, gap roughly halved; Section 4.6). The construction that makes this possible is the greedy-intergenic partition of Section 3.1, built on the ILP's *own* gene sequences so that every block maps to an exact model variable—sidestepping the occurrence-selection and relabelling performed by the heuristic balancing step, which had made variable recovery from the heuristic's output impractical.

Four threads remain open. *(i) The FPT side.* Its pruning is driven by the incumbent *cost* alone, so on hard instances it still emits a phantom rather than the heuristic's better partition; giving the FPT a full initial solution (so it can *report* that partition instead of an empty one) is the natural counterpart of the ILP MIP start and is not yet implemented. *(ii) The CS formulation.* Our MIP start targets the CB model; the substring-based CS model would need an analogous variable-level construction. *(iii) Closing the gap.* The MIP start improves the *incumbent*, but proving optimality on $K = 4$ requires lifting the *lower bound*, and aggressive settings (`MIPFocus=2`, `Cuts=2`) did not help—on the $\sim$167M-nonzero model the added cut generation exhausts the budget before a bound is even established. Tightening that bound (through a stronger formulation or problem-specific cuts) is the main obstacle to certified optimality at low alphabet. *(iv) A pipeline-budget evaluation.* All six hypotheses are stated under the *solver-budget protocol* (Section 3.6): the fixed budget T bounds the solver phase alone, and model construction and the heuristic run outside it—only H3's end-to-end comparison folds the heuristic time back in. A stricter evaluation would adopt the *pipeline-budget protocol* throughout, running each method under a *single* wall-clock budget that covers construction, the heuristic, and the solver together (deducting all of them from T). This would especially reshape the low-alphabet $K = 4$ picture, where model construction alone consumes on the order of the whole budget, and it would measure the warm *pipeline* as a whole (under which a phantom counts as the pipeline having found the partition via the heuristic)—the natural complement to the present per-solver view.

A further direction is extending this warm-start methodology to the flexible-tolerance setting

of SMCFISP [11], in which intergenic-region lengths need only match within a bounded tolerance rather than exactly. The ILP models of Romeiro et al. [9] already support this setting, but no FPT algorithm has yet been developed for it; the heuristic bounds and warm-start mechanism described here should transfer directly to the ILP side of that problem, while an FPT counterpart remains an open algorithmic question.

References

[1] Guillaume Fertin et al. *Combinatorics of Genome Rearrangements*. 1st ed. Computational Molecular Biology. London, England: The MIT Press, 2009. ISBN: 9780262258753. DOI: 10.7551/mitpress/9780262062824.001.0001.

[2] Andre Rodrigues Oliveira et al. "Rearrangement Distance Problems: An updated survey". In: *ACM Computing Surveys* 56.8 (Apr. 2024). ISSN: 0360-0300. DOI: 10.1145/3653295.

[3] Andrew J. Radcliffe, Alex D. Scott, and Elizabeth Wilmer. "Reversals and Transpositions Over Finite Alphabets". In: *SIAM Journal on Discrete Mathematics* 19.1 (Jan. 2005), pp. 224–244. ISSN: 0895-4801, 1095-7146. DOI: 10.1137/s0895480103433550.

[4] Laurent Bulteau, Guillaume Fertin, and Irena Rusu. "Sorting by Transpositions Is Difficult". In: *SIAM Journal on Discrete Mathematics* 26.3 (Jan. 2012), pp. 1148–1180. ISSN: 0895-4801, 1095-7146. DOI: 10.1137/110851390.

[5] Alexsandro Alexandrino et al. "On the Complexity of Some Variations of Sorting by Transpositions". In: *JUCS - Journal of Universal Computer Science* 26.9 (Sept. 2020), pp. 1076–1094. ISSN: 0948-6968, 0948-695X. DOI: 10.3897/jucs.2020.057.

[6] Avraham Goldstein, Petr Kolman, and Jie Zheng. "Minimum Common String Partition Problem: Hardness and Approximations". In: *Proceedings of the 15th International Symposium on Algorithms and Computation (ISAAC'2004)*. Berlin, Heidelberg, 2005, pp. 484–495. DOI: 10.1007/978-3-540-30551-4_43.

[7] Gabriel Siqueira et al. "Approximation algorithm for rearrangement distances considering repeated genes and intergenic regions". In: *Algorithms for Molecular Biology* 16.1 (Oct. 2021), p. 21. ISSN: 1748-7188. DOI: 10.1186/s13015-021-00200-w.

[8] Gabriel Siqueira, Alexsandro Oliveira Alexandrino, and Zanoni Dias. "Signed rearrangement distances considering repeated genes, intergenic regions, and indels". In: *Journal of Combinatorial Optimization* 46 (Sept. 2023), p. 16. DOI: 10.1007/s10878-023-01083-w.

[9] Felipe Romeiro et al. "ILP Models for String Partition Considering Intergenic Regions and Indels". In: *Bioinformatics and Computational Biology (BICoB'2025)*. Ed. by Tamer Aldwairi, Hisham Al-Mubaid, and Oliver Eulenstein. Cham: Springer Nature Switzerland, 2025, pp. 55–67. ISBN: 978-3-031-94039-2.

[10] Gabriel Siqueira et al. "Heuristics for Genome Rearrangement Distance with Replicated Genes". In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 18.6 (2021), pp. 2094–2108. ISSN: 1545-5963, 1557-9964, 2374-0043. DOI: 10.1109/tcbb.2021.3095021.

[11] Gabriel Siqueira et al. "Partition Based Algorithms for Rearrangement Distances with Flexible Intergenic Regions". In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 22.2 (2025), pp. 455–468. DOI: 10.1109/tcbb.2024.3467033.