POLITECNICO DI TORINO
Repository ISTITUZIONALE

Cooperative Multi-Heuristic Parallelization for the Maximum Common Induced Subgraph Problem

Original
Cooperative Multi-Heuristic Parallelization for the Maximum Common Induced Subgraph Problem / Cardone, L., Quer, S.. - ELETTRONICO. - (2026), pp. 605-616. (21st International Conference on Software Technologies Porto (PRT) 16/07/2026 - 18/07/2026) [10.5220/0015187500004088].

Availability:
This version is available at: 11583/3013772 since: 2026-07-30T23:24:00Z

Publisher:
SCITEPRESS

Published
DOI:10.5220/0015187500004088

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

12 October 2026

Cooperative Multi-Heuristic Parallelization for the Maximum Common Induced Subgraph Problem

Lorenzo Cardone[a] and Stefano Quer[b]
Department of Automatic and Informatic - DAUIN, Politecnico di Torino, Corso Duca degli Abruzzi 24, Torino, Italy

Keywords: Maximum Common Induced Subgraph, Mcsplit-DAL, Cooperative Parallelism, MULTI-Heuristic Search, Branch-and-Bound, Graph Algorithms.

Abstract: The Maximum Common Induced Subgraph problem is a central challenge in combinatorial optimization, with applications across diverse fields. Its NP-hard nature has led to a long line of branch-and-bound algorithms, among which the McSplit family stands out for its search-space representation and effective pruning. More recent extensions, such as McSplit-DAL, integrate Domain Action Learning to guide branching decisions using dynamic reward functions. However, they remain essentially sequential and rely on a single heuristic configuration, underutilizing modern multi-core architectures and heuristic diversification. In this work, we introduce CP-McSplitDAL, a cooperative parallel framework that extends McSplit-DAL with portfolio-style multi-heuristic search on shared-memory machines. The original recursive algorithm is reformulated as an iterative engine, enabling explicit management of search states, load sharing among threads, and controlled thread migration between heuristics. Each context couples a topological vertex-ranking metric with a specific ordering scheme and learns its own reward landscape. Cooperation is achieved through a globally shared variable that represents the size of the largest solution found so far, enabling cross-heuristic pruning and adaptive reward handling, and supporting both unified and distributed matrices. At the same time, a master evaluation periodically decays rewards and deactivates under-performing heuristics, shifting from early diversification to late exploitation. We evaluate CP-McSplitDAL on standard benchmarks, including small instances solvable to optimality and a large set of real-world graph pairs. The results show that our cooperative multi-heuristic configuration achieves lower regret in time to optimality, improves solution quality under time limits, and better exploits multi-core hardware than non-cooperative or purely sequential variants.

1 INTRODUCTION

The Maximum Common Induced Subgraph (MCIS) problem[1] is a fundamental challenge in graph theory and combinatorial optimization, seeking the largest subgraph shared by two input graphs and thus capturing structural similarity between complex objects. Its NP-hard nature[2] makes it computationally demanding. However, it underpins many applications, from cheminformatics (Skvortsova et al., 1998) and social network analysis (Milgram, 1967) to pattern recognition and malware detection via behavioral graph comparisons (Park and Reeves, 2011). In these domains, a large common subgraph offers a principled way to compare entities, uncover recurring patterns, and detect anomalies.

From a High Performance Computing perspective, exact MCIS solvers exemplify irregular, branch-and-bound-style workloads, where search dynamics, pruning patterns, and memory behavior interact closely with the structure of the input graphs. Designing parallel algorithms that can exploit modern multi-core and many-core architectures for such problems is therefore not only important for specific MCIS applications, but also foundational for the broader class of combinatorial search problems arising in data analytics, network science, and AI.

Research has therefore focused on exact algorithms that balance aggressive pruning with effective search-space exploration to guarantee optimality. The

[a] https://orcid.org/0009-0008-7553-4839
[b] https://orcid.org/0000-0001-6835-8277

[1] Often referred to as the Maximum Common Subgraph (MCS) problem.

[2] The optimization problem is NP-hard, whereas the decision one, i.e., discovering whether there is a common subgraph larger than k, is NP-complete.(Garey and Johnson, 1979)

Cardone, L. and Quer, S.
Cooperative Multi-Heuristic Parallelization for the Maximum Common Induced Subgraph Problem.
DOI: 10.5220/0015187500004088
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 21st International Conference on Software Technologies (ICSOFT 2026)*, pages 605-616
ISBN: 978-989-758-855-6; ISSN: 2184-2833
Proceedings Copyright © 2026 by SCITEPRESS – Science and Technology Publications, Lda.

state of the art is dominated by branch-and-bound methods, among which McSplit (McCreesh et al., 2016) stands out for its bidomain-based representation of the search space, which yields tight upper bounds and effective pruning, greatly reducing the number of explored states. However, despite these advances in pruning strength and robustness, most state-of-the-art MCIS solvers have been engineered as essentially sequential programs, only marginally exploiting the massive parallelism offered by contemporary shared-memory processors with dozens of hardware threads.

More recently, researchers have introduced learning-driven strategies to enhance search guidance. Reinforcement learning techniques, in particular the Domain Action Learning framework of McSplit-DAL, augment classical branch-and-bound with dynamic reward functions that steer vertex-pair choices. These choices are influenced not only by the expected upper-bound reduction, but also by the structural simplification induced by each branching decision. As a result, the algorithm learns to favor moves that both tighten the bound and decompose the problem into smaller, more manageable subproblems, thus improving search efficiency and robustness across diverse instance classes.

1.1 Branch-and-Bound Algorithms

Originally, many exact approaches to the MCS problem were indirect, reducing it to a Maximum Clique search on an association graph built from the two input graphs, where vertices represent feasible pairs and edges encode pairing consistency, so that a maximum clique corresponds to a maximum common subgraph. This reduction enabled the reuse of maximum clique solvers and established an elegant theoretical connection between the two problems. However, later work showed that this strategy is not uniformly efficient. Indeed, many real-world instances produce huge association graphs and suffer from severe performance degradation, motivating algorithms that work directly on the input graphs and exploit vertex partitioning and problem-specific structure.

The McSplit algorithm introduced a major advance by revisiting branch-and-bound with a refined search-space representation. First, it maintains an explicit partition of candidate vertices into bidomains, which represent pairs of compatible vertices, one from each graph. Then, it combines this compact representation with a lightweight bound computation, enabling tight upper bounds and rapid pruning, while preserving structural information and maintaining a compact state representation. McSplit's search tree naturally lends itself to parallel exploration because different workers can process distinct branches independently. However, the original implementation targets a single-core, sequential execution model.

More recently, researchers have extended the McSplit framework with reinforcement learning techniques that replace fixed heuristics with learned value functions. These value functions gradually identify promising branching choices as the search progresses. McSplitRL (Liu et al., 2020) rewards vertex pairings that maximally reduce the upper bound, whereas McSplitLL (Zhou et al., 2022) incorporates LSTM models and the LUM optimization to exploit temporal patterns and handle leaf vertices more efficiently. McSplitDAL (Liu et al., 2022) goes one step further by generalizing the reward signal. It jointly accounts for bound tightening and structural simplification through bidomain fragmentation. In doing so, it favors actions that simultaneously reduce the remaining search space and decompose the problem into smaller subproblems, thereby substantially improving robustness and efficiency across diverse graph instances.

1.2 The Logic of Vertex Sorting

A critical but often underappreciated component of these approaches is the initial ordering of vertices, which in early implementations relied solely on node degree. In branch-and-bound, the pairing order not only influences how quickly an optimal solution is reached but also shapes the search tree and thus the effectiveness of pruning and the overall cost of the optimality test. Poor ordering can delay finding large common subgraphs and yield large, bushy trees, whereas a good ordering quickly exposes high-quality solutions and sharply reduces the search space.

Recent studies have therefore integrated more sophisticated topological heuristics into the McSplit-DAL framework. Rather than using only degree, they exploit centrality and clustering metrics, such as PageRank (Brin and Page, 1998), Katz centrality (Katz, 1953), betweenness centrality (Freeman, 1977), local clustering coefficient (Watts and Strogatz, 1998), and closeness centrality (Sabidussi, 1966), to identify structurally important nodes (hubs, bridges, and highly clustered vertices). Prioritizing these vertices in the pairing order often leads to earlier discovery of large common substructures and more informative branching decisions, improving search efficiency across diverse graph families.

These works are particularly critical in the context of High Performance Computing. Although modern CPUs offer increasing core counts, deep cache hierarchies, and sophisticated synchronization primitives,

branch-and-bound graph algorithms often suffer from limited scalability due to irregular search, contention on shared data structures, and poor cache locality. Bridging this gap requires explicit algorithmic and architectural design to expose sufficient parallelism, control contention, and maintain pruning power.

Building on these ideas, we present the Cooperative Parallel McSplit-DAL (CP-McSplitDAL) framework, which extends McSplit-DAL in two main ways. First, we provide, to the best of our knowledge, the first shared-memory parallel implementation of McSplit-DAL, allowing multiple branches of the search tree to be explored concurrently and reducing wall-clock time to optimality. Second, we introduce a multi-heuristic collaborative framework in which multiple, distinct heuristics operate simultaneously on the same problem, supported by dynamic thread migration and global sharing of the best solution so far, so that computation concentrates on the most promising regions of the search space.

The CP-McSplitDAL architecture is designed to maximize modern hardware utilization while preserving the robustness of an exact method and adding heuristic flexibility and adaptivity, allowing the solver to adjust to different instance characteristics during the run. Our experiments focus on Katz and betweenness centrality, combined with the original node-degree heuristic and both standard and reverse sorting orders. Tests on heterogeneous benchmark sets show that the parallel cooperative strategy significantly improves execution speed and the ability to find large common subgraphs, even on large-scale instances, with the synergy between heuristics and thread reallocation yielding strong performance across a wide range of graph topologies (Foggia et al., 2001)(Leskovec and Krevl, 2014).

2 BACKGROUND

This section introduces the fundamental notations and definitions used throughout the rest of the paper, with particular reference to the structure of the McSplit algorithm and the extensions introduced by the DAL learning framework.

2.1 Problem Definition

A graph is a pair $G = (V, E)$, where V is the set of vertices and $E \subseteq V \times V$ is the set of edges. In this work, we consider unweighted graphs, whether directed or undirected.

Given a graph $G = (V, E)$, a subgraph $H = (V_H, E_H)$ is a graph such that $V_H \subseteq V$ and $E_H \subseteq E$. A subgraph H is said to be induced by a set of vertices $V_H \subseteq V$ (often denoted by $G[V_H]$) if the set of edges E_H contains all and only those edges in G whose both endpoints belong to V_H. Formally: $E_H = \{(u, v) \in E \mid u, v \in V_H\}$.

Given two graphs G and H, a Common Induced Subgraph is a graph S isomorphic to an induced subgraph of G and an induced subgraph of H. The Maximum Common Induced Subgraph (MCIS) problem consists of finding a common induced subgraph that maximizes the number of vertices $|V(S)|$. We represent a solution as a set of matched pairs:

$$M = \{(v_1, w_1), (v_2, w_2), \ldots, (v_k, w_k)\} \quad (1)$$

where $v_i \in V(G)$ and $w_i \in V(H)$.

2.2 McSplit Algorithm

The efficiency of the McSplit algorithm derives from partitioning unpaired vertices into equivalence classes. We define an equivalence class (or a label class) as a set of vertices within the same graph that exhibit identical adjacency relationships with respect to the vertices already included in the partial solution M. Within this framework, we indicate a pair of vertex sets (V_l, V_r), with $V_l \subseteq V(G)$ and $V_r \subseteq V(H)$, such that all vertices in both sets belong to the same label class, as a bidomain bd. Formally, this procedure ensures that every vertex in V_l is a feasible match for every vertex in V_r. Moreover, at each node of the search tree, we can compute an upper bound B on the total number of admissible pairings according to the following equation:

$$B = |M| + \sum_{(V_l, V_r) \in Ev} \min(|V_l|, |V_r|) \quad (2)$$

where Ev (the environment) is the set of all current bidomains. When B is less than or equal to the size of the current best solution, we prune the branch of the search tree.

2.3 Domain Action Learning (DAL)

The DAL framework extends the classic branch-and-bound algorithm by recasting the choice of which vertex pair to explore next as a learning problem rather than a fixed heuristic rule. Instead of relying solely on handcrafted criteria, the solver observes the effect of its branching decisions on the search process and updates its preferences accordingly, gradually learning which moves tend to be most beneficial.

The environment, usually denoted by Ev, represents the current state of the search. We define it as

the set of active bidomains that remain to be explored and, by extension, encodes both the unexplored portion of the search space and the current partial solution. Each change in the partial matching or in the bidomain structure corresponds to a transition to a new environment state.

The reward R is a scalar value associated by the system to each candidate pairing (v,w). Instead of following classical reinforcement learning schemes that focus solely on how much a decision improves a single objective (such as tightening an upper bound), DAL designs the reward to capture two complementary effects of each branching choice. It measures both how strongly the choice tightens the upper bound and how much it simplifies the remaining problem structure (for example, by fragmenting or shrinking the bidomains). In practice, the framework learns to favor pairings that both reduce the search potential and decompose the instance into smaller, more manageable subproblems. This mechanism promotes more effective pruning and a better-informed exploration of the search tree. The reward is defined as follows:

$$R(v,w) \quad = \quad \Delta B + |Ev'| \tag{3}$$

where:

- $\Delta B = B_{pre} - B_{post}$ represents the reduction in the pairing potential of the upper bound following the choice made (i.e., the difference between the sum of the minimums of the bidomains before and after the pairing).
- $|Ev'|$ is the number of bidomains generated in the new environment.

A higher value of R indicates a choice that not only tightens the bounds but also increases the fragmentation of the search space, simplifying future subproblems.

McSplit-DAL performs a depth-first search over a branch-and-bound decision tree. As illustrated in the reference pseudo-code, the algorithm operates according to the following logical steps:

- Potential Evaluation: At each node of the tree, the algorithm computes the upper bound B. When B does not exceed the size of the current best solution (*MaxSol*), the algorithm prunes the entire branch.
- Strategic Selection: The algorithm selects the smallest bidomain (fail-first heuristic) and a vertex v in the first graph. The choice of vertex is guided by accumulated rewards, alternating between RL and DAL policies. When the algorithm has not yet visited any potential matches in the bidomain, it falls back to pairing nodes in the order in which they were initially computed.

Algorithm 1: McSplit-DAL: The original, i.e., sequential version.

Procedure `McSplitDAL` $(Ev, curSol, MaxSol)$ **:**
  $bound \leftarrow |curSol| + \sum_{(V_p,V_t)\in Ev} \min(|V_p|,|V_t|)$;
  if $bound \leq |MaxSol|$ **then**
    Return $MaxSol$;
  end
  $(V_p, V_t) \leftarrow$ `SelectDomain`(Ev);
  $v \leftarrow$ `select`$(V_p,$ VertexRewards$)$;
  foreach $w \in V_t$ *in reward-based order* **do**
    $V_t \leftarrow V_t \setminus \{w\}$;
    $curSol \leftarrow curSol \cup \{(v,w)\}$;
    if $|curSol| > |MaxSol|$ **then**
      $MaxSol \leftarrow curSol$;
    end
    $Ev' \leftarrow$ `UpdateDomains`(Ev,v,w);
    $R(v,w) \leftarrow \Delta B + |Ev'|$;
    `UpdateRewards` (v, w, R(v, w));
    $MaxSol \leftarrow$ `McSplitDAL`$(Ev', curSol)$;
    $curSol \leftarrow curSol \setminus \{(v,w)\}$;
    $V_t \leftarrow V_t \cup \{w\}$;
  end
  $Ev_{new} \leftarrow Ev \setminus \{v\}$;
  $MaxSol \leftarrow$ `McSplitDAL`$(Ev_{new}, curSol)$;
  return $MaxSol$;

- Exploration and Learning: For each vertex w in the second graph that is compatible with v, the algorithm attempts to pair it with v. At each step, it updates the environment Ev by decomposing the bidomains. It computes the reward $R(v,w) = \Delta B + |Ev'|$, which guides the algorithm toward pairings that simplify the problem more quickly.
- Backtracking: Once all possibilities for a pair (v,w) have been explored, the algorithm removes v from the current options and continues the search to ensure that solutions that do not include v are explored, thereby ensuring the completeness of the exact search.

2.4 Heuristics and Sorting Configuration

Before the search begins, the vertices are sorted according to one of many metrics X. The work by Calabrese et al. (Calabrese et al., 2026) defines three sorting configurations:

- Standard: Vertices sorted in descending order of X ($v_{max} \rightarrow v_{min}$).
- Reverse: Vertices sorted by increasing values of X ($v_{min} \rightarrow v_{max}$).
- Inverse: A control configuration in which the vertices of G are in standard order and those of H in

reverse order (or vice versa). Used to validate the statistical significance of the pairing based on the chosen metric.

3 METHODOLOGY

This section describes the architecture of the Cooperative Parallel McSplitDAL (CP-McSplitDAL), tracing its evolution from a rigid, recursive sequence into a dynamic, parallel, and more efficient tool. By breaking free from sequential constraints, the model transforms independent computations into a unified collaborative search, leveraging diverse heuristic strategies simultaneously to accelerate discovery.

3.1 Iterative Parallel Framework

The key step in enabling parallelism in McSplit-DAL is transforming the search logic from recursive to iterative. While the original algorithm relies on the system stack, our implementation uses an explicit stack of Step objects, where each object encapsulates the complete state of a node in the search tree (current subdomains, partial solution, and branching cycle index). This architectural choice is motivated by three technical requirements:

- Control Management: It allows execution to be suspended at any time (e.g., due to a timeout or synchronization) without losing the current reached state.
- Load Sharing: It evenly distributes the workload across available threads.
- Thread Migration: A mechanism by which an execution thread leaves a stagnant heuristic context to move to a more promising one. It enables an efficient transfer of unexplored portions of the search tree between different contexts. This operation is considerably more difficult in the original purely recursive implementation, but reasonably manageable in our new implementation with an explicit stack and a separate task pool.

Overall, our framework coordinates a pool of parallel threads. Each thread manages its own local stack and interacts with global data structures for pruning and learning.

3.2 Multi-Heuristic Approach

The key innovation of CP-McSplitDAL is its rejection of the "one-size-fits-all" heuristic. Rather than relying on a single vertex-ranking metric (such as node degree or PageRank), the framework activates a battery of Heuristic Contexts, each representing a search engine instance configured with a specific heuristic and sorting policy. This multi-pronged strategy prevents any single perspective from bottlenecking the search.

We explain the mechanics of this competitive yet collaborative process in the following sections.

3.2.1 Search Diversification

Each heuristic explores the search tree using a specific combination of a topological metric ($\mathcal{H}$) and an ordering scheme (O). This approach builds on experimental evidence showing that the topological properties of input graphs (such as density, diameter, and centrality) strongly influence the effectiveness of a given heuristic. By applying multiple strategies simultaneously (for example, Katz Standard Sorting and Betweenness Reverse Sorting), the system avoids getting stuck in regions of the search tree that are difficult to reach for a specific metric.

3.2.2 Cooperation and Synergy

Unlike simply running different algorithms in parallel, our multi-heuristic approach is cooperative. The various heuristics do not merely compete to find the solution first, but collaborate by sharing critical information and migrating the workload. To maximize search effectiveness, CP-McSplitDAL implements two reward management modes, allowing a choice between close collaboration or greater exploratory independence.

We derive the close (direct) collaboration from classic multi-threaded synergy models, and we use a single global reward matrix shared across all contexts. Although this strategy promotes rapid convergence, it also risks limiting search diversity. Since the McSplitDAL algorithm always favors pairings with higher rewards, the first thread to explore a fruitful path "influences" the others to follow the same branching choices. This behavior partially undermines the benefit of having multiple initial orders.

Under the independent exploration scheme, we utilize distributed matrices to ensure structural autonomy. By equipping each heuristic context with its own local reward matrix (Q_{loc}), we foster a radically diverse search environment. This isolation prevents premature convergence, allowing every strategy to map the graph's topology through its own lens without being skewed by the trajectories of neighboring instances.

Our working hypothesis is that preserving independent reward landscapes allows different heuristics to specialize in distinct regions of the search space. Rather than converging toward a common set

of branching preferences, each context develops its own search behavior, increasing portfolio diversity and improving the likelihood that at least one heuristic will rapidly discover high-quality solutions. The experimental results presented in Section 4 support this hypothesis, showing that the distributed-matrix configuration consistently outperforms the unified-matrix alternative.

To avoid losing the information computed during independent explorations, when the worst-performing heuristics are pruned (as detailed in Section 3.3.1), the system does not discard the learned weights. During thread migration, the system automatically integrates the rewards generated by a pruned heuristic into the surviving ones. This inheritance allows surviving threads to absorb the strategic successes and failures of their predecessors, ensuring that the search continuously evolves. Although this method provides many benefits, the distributed matrix model incurs a heavy memory tax, as each matrix scales quadratically (specifically $O(|V_G| \cdot |V_H|)$) with the vertex count. When navigating large-scale graph instances, the sheer volume of data required to sustain a dedicated matrix for every heuristic context can reach prohibitive levels.

3.2.3 Global Best Solution

The system stores the size of the best solution found so far as a shared atomic variable, regardless of the chosen reward model. This design enables instantaneous cross-heuristic pruning: as soon as one context finds evidence of a better solution, the system prunes branches that appear only locally promising in other contexts.

3.3 Coordination and Pruning Dynamics

The framework achieves peak efficiency by rejecting a static approach to resource allocation. More specifically, rather than sustaining every heuristic indefinitely, it employs a dynamic load-management regime. This mechanism continuously evaluates the utility of active search instances, identifying and decommissioning underperforming heuristics to free up computational resources. By pruning unproductive branches of the search tree in real time, the system ensures that computational energy is never wasted on stagnant paths. Instead, the system immediately redirects these reclaimed resources toward the most promising trajectories, turning the search from a rigid parallel execution into a fluid, efficient pursuit of the optimal solution. The following subsections highlight the main characteristics of this process.

3.3.1 Master Evaluation and Global Decay

The system performs a Master Evaluation after N nodes have been explored. The value of N strictly defines the system's behavior:

- A high value provides sufficient time for all heuristics to visit a significant portion of the graph, allowing for deeper exploration of branches without the constant pressure to change strategy.
- A low value favors an extremely dynamic approach. In this scenario, threads frequently switch between heuristics to react to even the slightest signs of stagnation. However, this dynamism comes at the cost of increased resource waste from context-switching overhead and the potential loss of locality in heuristic learning.

We identify a good value of N to ensure stability without sacrificing responsiveness. Thus, we set $N = 10^5$ nodes.

The framework further refines its strategic focus by applying a Global Decay Factor, a temporal penalty that maintains search agility. This parameter systematically erodes the accumulated rewards of heuristics that fail to yield incremental improvements over a set duration. By actively devaluing stale intelligence, the system must periodically re-evaluate its vertex prioritization, which effectively disrupts any emerging biases. This renewed assessment restores diversity in the search, pushing it to leave exhausted regions of the graph and redirect its effort toward unexplored, high-potential areas.

3.3.2 Dynamic Pruning and Thread Migration

The system employs pruning milestones (typically at 30% and 70% of the expected computation time or the time limit) to progressively deactivate the least effective heuristics. Although these thresholds are identified experimentally, they are crucial for balancing exploration and exploitation of the search space.

During the initial phase (which includes up to 30% of the total time dedicated to the experiments), the system prioritizes heuristic diversity. In this phase of extensive exploration, the goal is to map different contexts to distinct regions of the tree, enabling the rapid identification of a high-quality common subgraph.

During the intermediate and final phases (that is, once we reach about 70% of the total time allowed), the strategy shifts toward intensive exploitation. Once the system meets the milestones, it reduces resource

dispersion by directing computational power only to the most efficient heuristic.

When the system abandons a heuristic, it marks all associated threads as orphaned and triggers thread migration. These orphaned threads then discard their local stacks, adopt the context of the "winning" heuristic, and start collaborating, carrying over the knowledge stored in their local reward matrices to strengthen the final search phase and accelerate the optimality test.

3.4 Computational Complexity and Resource Requirements

The proposed framework preserves the exact search semantics of McSplit-DAL and therefore inherits its worst-case exponential time complexity. As with all branch-and-bound algorithms for MCIS, the practical execution time depends primarily on the effectiveness of pruning rather than on asymptotic bounds. Consequently, the objective of CP-McSplitDAL is not to alter the theoretical complexity of the search, but rather to improve the exploration efficiency through parallelism, heuristic diversification, and cooperative pruning.

From a memory perspective, the dominant data structures are reward matrices, graph representations, and search stacks maintained by each thread. Let $|V_G|$ and $|V_H|$denote the number of vertices in the two input graphs. A single DAL reward matrix requires $O(|V_G| \cdot |V_H|)$ storage. Therefore, the unified-matrix configuration maintains a memory complexity of $O(|V_G| \cdot |V_H|)$, whereas the distributed-matrix configuration requires $O(H \cdot |V_G| \cdot |V_H|)$, where H is the number of active heuristic contexts. This increase in memory consumption constitutes the main cost of preserving independent reward landscapes across heuristics.

The unified-matrix architecture introduces additional synchronization overhead because all threads update the same reward structure. Although individual updates are lightweight, frequent accesses to shared atomic variables can generate contention, cache-coherence traffic, and memory serialization. These costs increase with the number of active threads and can ultimately limit scalability on shared-memory systems. Conversely, the distributed-matrix model eliminates most of this contention at the expense of higher memory consumption.

Thread migration introduces additional overhead not present in the original recursive implementation. However, migration events occur only at predefined evaluation milestones and involve transferring a limited amount of contextual information. Consequently, their computational cost remains negligible compared with the overall cost of exploring the search tree. In practice, the benefits obtained by concentrating computational resources on the most promising heuristic contexts largely outweigh the migration overhead.

Overall, CP-McSplitDAL introduces a trade-off between memory consumption and synchronization overhead. The unified-matrix model minimizes memory requirements but may suffer from contention at high thread counts. In contrast, the distributed-matrix model consumes additional memory to maximize heuristic independence and parallel efficiency. The experimental results presented in Section 4 confirm that, for the considered benchmark sets, the distributed-matrix configuration generally provides the most favorable balance between memory overhead, search diversity, and parallel scalability.

4 EXPERIMENTAL RESULTS

In this section, we analyze the performance of the CP-McSplitDAL framework, focusing on scalability, multi-heuristic search efficiency, and the impact of the memory architecture.

4.1 Test Setup and Measurement Methodology

We ran our tests on a workstation with an Intel Core i9-10900KF CPU and 64 GBytes of DDR4 RAM. All our algorithms are written in C++ and compiled with GCC version 11.4. The test system is a multi-core machine supporting C++17 atomic instructions and configured to handle up to 20 simultaneous threads.

We intentionally conducted all experiments on a commodity workstation rather than on a large shared-memory server. The goal was to evaluate whether the proposed cooperative framework can deliver practical benefits on hardware commonly available to researchers and practitioners. Although larger servers could potentially provide additional parallelism, our scalability analysis shows that synchronization costs and memory contention already become relevant on 20 hardware threads. Consequently, understanding and mitigating these bottlenecks is a prerequisite before scaling the approach to many-core systems.

We conduct our evaluations using standard and publicly available benchmarks. We divided these benchmarks into two separate sets to better focus our experiments:

- The "SMALL" set consists of 750 graph pairs, each with up to 100 vertices generated by the tool

of Foggia et al. (Foggia et al., 2001), and freely available online[3]. This dataset allows exhaustive (maximal) solutions for most instances, enabling direct comparisons of runtimes.

- The "LARGE" set (AS-733) comprises real-world graphs with 500 to 6,000 nodes, and is publicly available[4]. Because the underlying problem is NP-hard and exact solutions for such instances would require impractically long runtimes, we instead evaluate algorithms by the quality of the solutions they obtain within a fixed time budget.

In this section, to demonstrate the effectiveness of our approach, we compare the two most promising heuristics, i.e., the Katz centrality and the betweenness centrality, with the original Node Degree heuristic. We test each of them in both their standard and reverse sorting configurations. Furthermore, we run our approach with all these heuristics and sorting configurations, using both the single-matrix and distinct-matrix approaches, to compare them. Notice that in all the figures in this section, all the heuristic names have been shortened to three or four characters to make the representation more readable. Please recall that all tests run with all 20 threads available and a timeout of 60 seconds.

4.2 Scalability and Speedup Analysis on the SMALL Dataset

The scalability analysis reveals nonlinear behavior, highlighting the limitations of shared-memory architectures when applied to the DAL framework.

We first run experiments by adopting the unified-matrix approach. In this case, all threads update a single global data structure. As the number of threads grows, contention on this shared structure increases, leading to rapid degradation in speedup. With 1 to 4 threads, we obtain a reasonable acceleration, achieving a speedup of about 2.5×. However, beyond 4 threads, the implementation encounters a hardware-induced bottleneck, and even with 20 threads, the speedup barely exceeds 3×. This behavior stems from synchronization on the atomic variables of the reward matrix, which act as a persistent point of contention among the cores, although it's not the only one.

To extend this analysis to the multiple-matrices configuration, we cannot rely on a simple comparison of runtime speedups. By design, this algorithm requires at least one thread per heuristic. Consequently, it is impossible to test a stable heuristic set across an increasing number of threads without primarily measuring how reward separation impacts efficiency; a topic addressed in later sections. Therefore, we shift our focus to comparing the iteration speed of the two modes, as this metric remains largely unaffected by the specific choice of heuristics.

Table 1 illustrates that separating the matrices effectively removes a critical bottleneck. When the threads do not compete for the same reward matrix (at least during the initial phases of the computation, when the common subgraph size needs to be updated frequently), we observe a significant improvement in the maximum achievable speedup. However, it is important to note that iteration speed and overall solution speed do not scale proportionally. While the single-matrix structure barely achieves a 1.7× increase in iteration speed over the baseline, the actual runtime speedup reaches ∼3×. This phenomenon, further explored in McCreesh (McCreesh, 2017), characterizes a superlinear speedup scenario. When threads concurrently explore different regions of the search space, they can discover the optimal solution much sooner than one would predict from the raw iteration rate alone, assuming a deterministic visitation order as in the baseline.

4.3 Search Efficiency and Regret Analysis on the SMALL Dataset

To evaluate the effectiveness of the multi-heuristic strategy on exhaustively solved instances, we used the Geometric-mean Regret metric, which measures how much an algorithm deviates from the performance of the best possible algorithm.

4.3.1 Regret in Time to Find the Optimum (Best Time)

The CP-McSplitDAL framework with separate matrices (BKD_Distinct) exhibits the lowest regret, as Figure 1 shows. By running multiple heuristics in parallel, the probability of quickly finding the correct solution increases dramatically, reducing the risk of stalling on an inefficient heuristic for that specific topology.

4.3.2 Regret at end Time

In the optimality test, the algorithms tend to converge, showing less variation in regret between approaches, while still maintaining the effectiveness of the version of our approach with separate matrices. Fig-

[3]Please refer to https://github.com/jamestrimble/ijcai2017-partitioning-common-subgraph/tree/master/mcs-instances.

[4]Please refer to https://snap.stanford.edu/data/as-733.html.

Table 1: Speedup of the average iteration with the different approaches as a function of the number of threads.

Approach	1 thread	2 threads	4 threads	8 threads	16 threads	20 threads
Single matrix	1	1.40	1.70	1.70	1.70	1.70
Multiple matrices	1	1.45	2.10	2.40	2.75	2.75

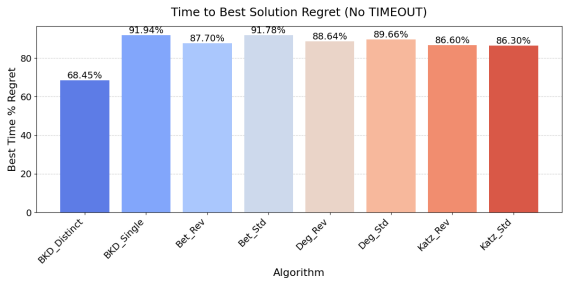

Figure 1: Regret computed over time to find the optimal solution on the solved instances of the SMALL Dataset, a lower result means better performance.

ure 2 shows this behavior. However, the unified matrix shows higher regret. The short duration of the experiments and the early heuristic interference pose serious obstacles to the algorithm's efficiency.

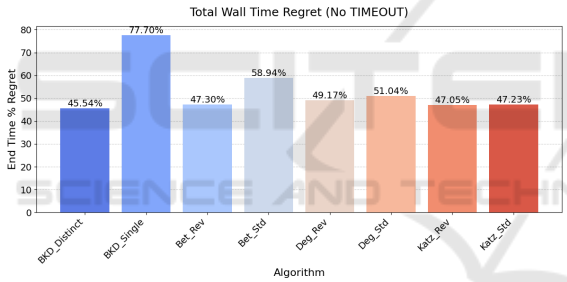

Figure 2: Regret computed over time to fully explore the search space on the solved instances of the SMALL Dataset, a lower result means better performance.

4.3.3 Solution Quality for Timed-out Instances

As Figure 3 shows, if not solved within the timeout, the version with separate matrices outperforms all other approaches. The version with a combined matrix, despite the initial overhead, maintains solution quality comparable to that of any single heuristic. Each cell of the confusion matrix shows, on average, how large the solution produced by the column algorithm is compared to the one produced by the row algorithm. Numbers close to zero indicate that the two approaches produce relatively similar results, while negative numbers indicate that the row algorithm finds larger solutions. Because the graph pairs used in these tests are generally small, all approaches tend to produce solutions of similar size, so the confusion matrix rarely reports large deviations between heuristics. However, this gap will increase on the LARGE Dataset.

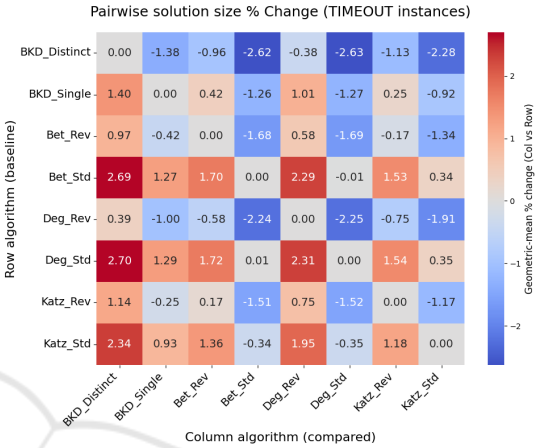

Figure 3: Confusion matrix comparing the average solution size of the column heuristic versus the row heuristic. A higher value indicates that the column heuristic performs better on average. Note that the matrix reports only timed-out instances, because all instances that terminate within the time limit yield identical solutions.

4.4 Performance on the LARGE Dataset (AS-733)

An analysis of large-scale instances confirms the robustness of the cooperative multi-heuristic model. Figure 4 shows the aggregated solution-size regret for the runs on the entire dataset. The bar chart allows, at a glance, to quantify the distance of each approach from the best-performing one, i.e., the one with the shortest bar.

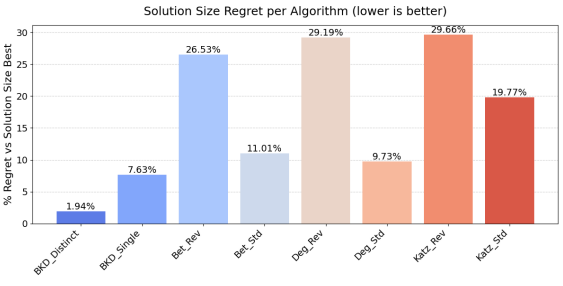

Figure 4: Regret computed over the largest solution found by the algorithm within the allotted time on the instances of the LARGE Dataset, a lower result means better performance.

As evidenced by the aggregated results, both proposed parallel approaches exhibit lower regret than any single-heuristic run. The separate-matrix approach is confirmed as the best performer, with extremely low regret ($\sim$1.94%).

4.4.1 Analysis of Regret Evolution with Graph Size

To better understand how structural complexity affects performance, we analyzed the trend of regret as a function of the number of vertices in the smaller graph of the pair.

The blue curve (our multi-heuristic, distinct-matrices approach) remains consistently close to zero across all graph sizes, while the single-matrix strategy lags a little behind. The latter never excels, but it also never suffers from a high regret. This behavior demonstrates that the system is extremely resilient: even when graphs become massive (5,000 nodes or more), the collaboration of several heuristics almost always guarantees the identification of the best possible solution. Conversely, classical single-heuristic approaches exhibit unstable trends. This pattern confirms that no single metric is "universal"; our approach solves the problem by avoiding the necessity to choose a strategy a priori.

Figure 5 shows the averaged rolling window of the regrets over the solution size. Although we have a clear trend, rolling windows are necessary to smooth the results and show the behavior of each approach; without them, the graph would be too noisy to yield meaningful results.

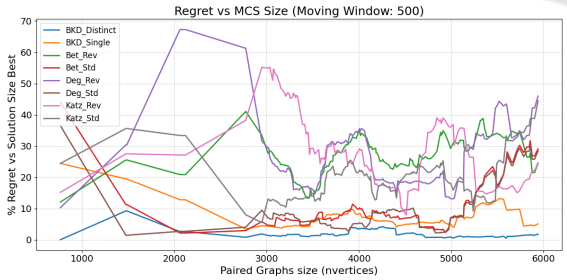

Figure 5: Curve showing regret evolution with increasing graph size (averaged rolling window). Smaller values indicate a more efficient methodology.

4.4.2 Pairwise Analysis of Solution Size

To further investigate the advantages of the new approaches on complex graphs, we analyzed the average ratio of the best solution size (best_sol) found by each pair of algorithms.

The aggregated data in the heatmap in Figure 6 confirms two clear trends.

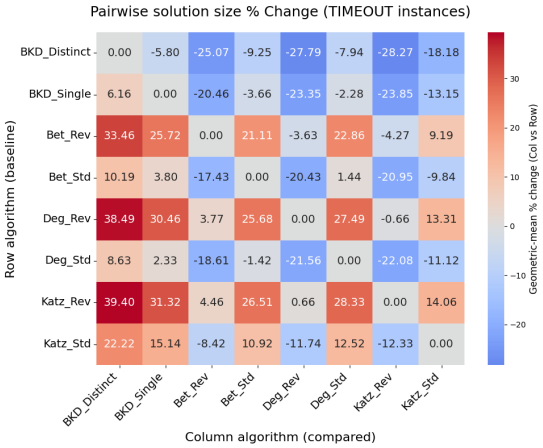

Figure 6: Confusion matrix comparing the average solution size of the column heuristic vs. the row heuristic; a higher value indicates that the column heuristic performs better on average.

First of all, the two new parallel frameworks, the one with a unique matrix and the one with separate matrices, perform particularly well compared to all competitors. Consistently low values in the row of a single heuristic (e.g., BKD_Distinct with values ranging from ~ -5.80, all the way to ~ -28.27) indicate that it finds solutions that are, on average, much larger. These considerations indicate a clear dominance of cooperative approaches.

Secondly, within the parallel frameworks, the superiority of the separate-matrix model is particularly evident. Although the multi-heuristic approach within unified-matrix variants already outperforms individual heuristics thanks to strategy diversification, it becomes less efficient when many threads contend to update the global information. Conversely, the separate-matrix model (BKD_Distinct) emerges as the most effective configuration thanks to both the reduced resource contention and clearer reward distinction. By isolating the rewards for each heuristic, the system avoids the "confusion" of global learning. It allows each context to explore its own space without interference, leading to systematically superior common subgraphs.

4.4.3 Memory Impact

On small graphs, memory usage is not a concern, but on larger instances, it becomes a factor that deserves careful attention. Memory mainly serves three purposes: storing the graph structure, recording rewards, and maintaining backtracking data.

The information required for backtracking is tightly interwoven with the code, enabling retracing the search and refining the current solution. To repre-

sent the graphs while minimizing memory usage, we use adjacency lists. The reward matrix forms another essential component that we cannot eliminate. It can consume a substantial amount of memory, especially when the algorithm keeps multiple copies, since its size grows proportionally to $|V_{G1}| \cdot |V_{G2}|$.

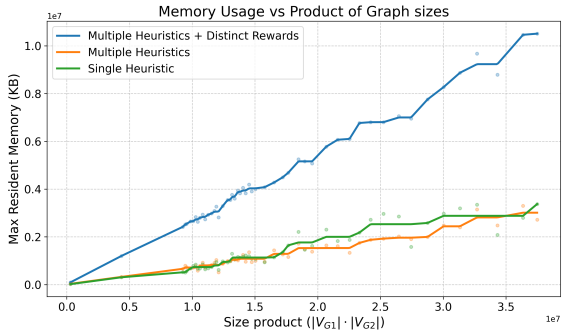

Figure 7: Plot comparing the memory usage of the different approaches vs. the size of reward matrices.

Figure 7 shows how the memory footprint of the approaches evolves with increasing size of the reward matrix and, by extension, the complexity of the MCS problem. Assuming two graphs containing 6,000 vertices each, the reward matrix contains approximately 36 million entries. Using 64-bit values, a single matrix requires approximately 275 MB. When six heuristic contexts are active simultaneously, the distributed-matrix approach may require up to 1650 MB exclusively for reward storage. Furthermore, multiple-heuristics approaches also require storing the additional descriptors for each node and the required information for thread migration. Although this memory footprint remains manageable on modern workstations, it highlights the trade-off between heuristic independence and memory consumption. Conversely, the unified-matrix model significantly reduces memory requirements but introduces synchronization overhead due to frequent atomic updates. Consequently, the choice between the two architectures represents a trade-off between memory efficiency and parallel scalability.

4.4.4 Positioning with Respect to Existing MCS Solvers

Table 2 positions the proposed framework with respect to representative exact MCIS solvers. Early approaches, such as McSplit, focused primarily on improving branch-and-bound efficiency through compact bidomain representations and stronger pruning mechanisms. Subsequent works introduced learning-based guidance, culminating in McSplit-DAL, which combines reinforcement learning with domain-specific reward functions to improve branching decisions. However, these approaches remain fundamentally sequential and rely on a single heuristic configuration during execution. Although these heuristic variants effectively shrink the search tree, our experiments indicate that their richer decision mechanisms introduce additional computational overhead per explored node compared with the original McSplit. This consideration is particularly true for smaller and medium instances. Consequently, using these slower variants as baselines would distort the comparison, making our method appear stronger than it genuinely is.

CP-McSplitDAL extends this line of research along three complementary directions. First, it introduces a shared-memory parallel execution model based on an iterative search engine and explicit task management. Second, it simultaneously evaluates multiple vertex-ordering heuristics, avoiding dependence on a single ranking metric. Third, it enables cooperation among heuristic contexts through shared pruning information, reward management, and thread migration. To the best of our knowledge, CP-McSplitDAL is the first MCIS solver that combines learning-based search guidance, cooperative parallelism, and multi-heuristic exploration within a unified exact framework.

5 CONCLUSIONS

In this paper, we introduced CP-McSplitDAL, a cooperative multi-heuristic parallel framework for the Maximum Common Induced Subgraph problem, built on top of the McSplit-DAL learning-based branch-and-bound solver. By reformulating the recursive procedure as an iterative engine and orchestrating multiple heuristic contexts in parallel, our approach exploits both domain-specific structure and modern shared-memory architectures within a single exact framework.

Algorithmically, CP-McSplitDAL combines an explicit stack-based representation of search states, a portfolio of vertex-ordering heuristics based on topological metrics (Katz centrality, betweenness centrality, and node degree with standard or reverse sorting), and a cooperative DAL reward scheme with either a unified global matrix or separate matrices (which the process can merge during migration).

Experiments on standard MCIS benchmarks, including SMALL instances and the large AS-733 dataset, show that cooperation and heuristic diversity yield tangible gains: the separate-matrix configuration attains lower regret in time to optimality than

Table 2: Comparison of the main characteristics of representative exact MCIS solvers and the proposed CP-McSplitDAL framework.

Approach	Learning	Parallel	Multi Heuristic	Cooperative
McSplit	No	Yes	No	No
McSplitRL	Yes	No	No	No
McSplitLL	Yes	No	No	No
McSplitDAL	Yes	No	No	No
CP-McSplitDAL	Yes	Yes	Yes	Yes

any single-heuristic solver or unified-matrix variant. In large instances, CP-McSplitDAL consistently finds larger common subgraphs and maintains very low regret across a wide range of graph sizes.

Overall, these results highlight cooperative multi-heuristic parallelism as a promising direction for exact graph matching and learning-augmented branch-and-bound algorithms, motivating future work on distributed-memory extensions, adaptive online heuristic selection, and richer learned models within the DAL framework.

Specifically, upcoming research could leverage machine learning to dynamically blend contributions from multiple vertex orderings or introduce a probabilistic factor that occasionally bypasses rewards in favor of topological heuristics, thus balancing exploration and exploitation. Furthermore, we would like to mitigate the identified architectural bottlenecks by implementing local thread-level memories to aggregate reward updates, reducing the frequency of expensive atomic operations on the global matrices.

REFERENCES

Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117.

Calabrese, A., Cardone, L., Licata, S., Porro, M., and Quer, S. (2026). Improving state-of-the-art vertex sorting algorithms to compute the maximum common induced subgraph. *Journal of Heuristics*, 32(1):1.

Foggia, P., Sansone, C., and Vento, M. (2001). A Database of Graphs for Isomorphism and Sub-Graph Isomorphism Benchmarking. In -, page 176–187.

Freeman, L. C. (1977). A set of measures of centrality based on betweenness. *Sociometry*, pages 35–41.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York.

Katz, L. (1953). A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43.

Leskovec, J. and Krevl, A. (2014). SNAP Datasets: Stanford large network dataset collection.

Liu, Y., Li, C.-M., Jiang, H., and He, K. (2020). A Learning Based Branch and Bound for Maximum Common Subgraph Related Problems. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(03):2392–2399.

Liu, Y., Zhao, J., Li, C.-M., Jiang, H., and He, K. (2022). Hybrid Learning with New Value Function for the Maximum Common Subgraph Problem.

McCreesh, C. (2017). *Solving hard subgraph problems in parallel*. PhD thesis, University of Glasgow.

McCreesh, C., Ndiaye, S. N., Prosser, P., and Solnon, C. (2016). Clique and constraint models for maximum common (connected) subgraph problems. In *Lecture Notes in Computer Science*, pages 350–368. Springer International Publishing.

Milgram, S. (1967). The Small World Problem. *Psychology today*, 2(1):60–67.

Park, Y. and Reeves, D. (2011). Deriving common malware behavior through graph clustering. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '11, page 497–502, New York, NY, USA. Association for Computing Machinery.

Sabidussi, G. (1966). The centrality index of a graph. *Psychometrika*, 31(4):581–603.

Skvortsova, M. I., Baskin, I. I., Stankevich, I. V., Palyulin, V. A., and Zefirov, N. S. (1998). Molecular Similarity. 1. Analytical Description of the Set of Graph Similarity Measures. *Journal of Chemical Information and Computer Sciences*, 38(5):785–790.

Watts, D. J. and Strogatz, S. H. (1998). Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442.

Zhou, J., He, K., Zheng, J., Li, C.-M., and Liu, Y. (2022). A Strengthened Branch and Bound Algorithm for the Maximum Common (Connected) Subgraph Problem.