A Strengthened Branch and Bound Algorithm for the Maximum Common (Connected) Subgraph Problem

Jianrong Zhou¹, Kun He¹¹, Jiongzi Zheng¹, Chu-Min Li², Yanli Liu³
¹School of Computer Science and Technology, Huazhong University of Science and Technology
²MIS, University of Picardie Jules Verne, France
³Wuhan University of Science and Technology, China
jrzhou@hust.edu.cn, brooklet60@hust.edu.cn

Abstract

We propose a new and strengthened Branch-and-Bound (BnB) algorithm for the maximum common (connected) induced subgraph problem based on two new operators, Long-Short Memory (LSM) and Leaf vertex Union Match (LUM). Given two graphs for which we search for the maximum common (connected) induced subgraph, the first operator of LSM maintains a score for the branching node using the short-term reward of each vertex of the first graph and the long-term reward of each vertex pair of the two graphs. In this way, the BnB process learns to reduce the search tree size significantly and boost the algorithm performance. The second operator of LUM further improves the performance by simultaneously matching the leaf vertices connected to the current matched vertices, and allows the algorithm to match multiple vertex pairs without affecting the solution optimality. We incorporate the two operators into the state-of-the-art BnB algorithm McSplit, and denote the resulting algorithm as McSplit+LL. Experiments show that McSplit+LL outperforms McSplit+RL, a more recent variant of McSplit using reinforcement learning that is superior than McSplit.

1 Introduction

Graphs are important data structures used to model many real-world problems. One of the most basic graph problems is to measure the similarity of graphs. Given two graphs $G_0$ and $G_1$, the Maximum Common induced Subgraph (MCS) problem aims to find an induced subgraph $G'_0$ in $G_0$ and an induced subgraph $G'_1$ in $G_1$ such that $G'_0$ and $G'_1$ are isomorphic and the number of vertices of $G'_0$ and $G'_1$ is maximized. The vertices of $G'_0$ are said to be matched with the vertices of $G'_1$. MCS has a variant called the Maximum Common Connected induced Subgraph (MCCS), which further requires the induced subgraph to be connected. These problems, denoted as MCS problems, are widely applied in various domains, such as biochemistry [Giugno et al., 2013; Bonnici et al., 2013], cheminformatics [Raymond and Willett, 2002; Englert and Kovács, 2015; Duesbury et al., 2017], compilers [Hjort Blindell et al., 2015], model checking [Sevegnani and Calder, 2015], molecular science [Ehrlich and Rarey, 2011], pattern recognition [Solnon et al., 2015], malware detection [Bruschi et al., 2006; Park et al., 2013], video and image analysis [Shearer et al., 2001; Jiang and Ngo, 2003; Hati et al., 2016], etc.

As NP-hard problems, MCS and MCCS are computationally challenging. Many approaches have been proposed for solving them, which could be mainly divided into two categories: exact algorithms and inexact algorithms. An exact algorithm guarantees to obtain an optimal solution but runs in exponential time in the worst case. It aims to efficiently enumerate the whole search space or reduce the search space without affecting the solution optimality. The approaches for exact algorithms include Linear Programming (LP) [Bahiense et al., 2012], Branch-and-Bound (BnB) [Raymond and Willett, 2002; McCreesh et al., 2017; Liu et al., 2020], etc. An inexact algorithm aims to find a near-optimal solution within reasonable computational resources (e.g., time and memory). Approaches for inexact algorithms include heuristics [Bonnici et al., 2013; Englert and Kovács, 2015; Duesbury et al., 2017] and metaheuristics [Choi et al., 2012]. Recently, machine learning [Shearer et al., 2001] and deep learning [Zanfir and Sminchisescu, 2018; Bai et al., 2021] techniques are also adopted for solving MCS problems.

The BnB algorithms have exhibited high performance for MCS problems. Given a vertex $p$ in $G_0$, McSplit [McCreesh et al., 2017], one of the state-of-the-art algorithms for MCS, proposes a partition method to efficiently filter the set of candidate vertices of $G_1$ that can be matched with $p$, and uses a novel compact candidate set representation to dramatically reduce the memory and computational requirements during the search. McSplit+RL [Liu et al., 2020] further improves McSplit using a new branching method based on reinforcement learning.

In this work, we propose two new operators to speed up the BnB process, namely Long-Short Memory (LSM) and Leaf vertex Union Match (LUM). In a BnB process for MCS, the branching consists in first selecting a vertex $p$ from the first graph $G_0$, and then matching each candidate vertex $q$ of $G_1$ in turn. LSM maintains a score for each $p$ using the short-term reward, and the long-term reward of each branching pair...
Consider a simple (undirected and unlabelled) graph $G = (V, E)$, where $V$ and $E$ represent the vertex set and the edge set, respectively. Two vertices $u, v$ are called adjacent if $(u, v) \in E$, and the degree of a vertex $u$ is the number of its adjacent vertices. An induced subgraph $G' = (V', E')$ of $G$ consists of a vertex subset $V' \subseteq V$ and the edges in $E' = \{ (u, v) \mid u, v \in V', (u, v) \in E \}$.

Given two graphs $G_0$ and $G_1$, if there is an induced subgraph $G_0'$ of $G_0$ and there exists a bijection $\phi : V_0 \rightarrow V_1$, $\Phi(G_0')$ (or $\Phi(G)$) denotes a graph that mapping all vertices in $G$ by $\phi$ is also an induced subgraph of $G_1$, then we call $(G_0', \Phi(G_0'))$ the common induced subgraph of $(G_0, G_1)$. Let $V_0' = \{ v_1, v_2, ..., v_{|V_0'|} \}$, the common induced subgraph is also represented as a list of vertex pairs $\{(v_1, \phi(v_1)), (v_2, \phi(v_2)), ..., (v_{|V_0'|}, \phi(v_{|V_0'|}))\}$.

The Maximum Common induced Subgraph (MCIS) problem aims to find the common induced subgraph such that its number of vertices is maximized. In other words, the two induced subgraphs are isomorphic with the maximum vertex cardinality. There is a variant problem of MCIS, the Maximum Common Connected induced Subgraph (MCCS) problem, which requires the induced subgraph to be connected.

### 2 Problem Definition

This section presents two of the best-performing BnB algorithms for MCIS and MCCS, McSplit [McCreesh et al., 2017] and McSplit+RL [Liu et al., 2020]. To simplify the algorithm description, we initially assume that the graphs are undirected, as the method is easy to be adapted to handle various extensions of the problem [McCreesh et al., 2017].

At each search node, the BnB algorithm first estimates an upper bound of the best solution that can be found in the current subtree. If the upper bound is not larger than the size of the current best solution, the algorithm prunes this branch and backtracks, because any better solution cannot be found under this search node. Otherwise, it selects a new vertex pair to match, updates the current solution and then runs recursively. There are three key issues for implementing the BnB algorithm: 1) estimate the upper bound, 2) design the branching strategy, 3) maintain candidate vertices of the two graphs.

To address these issues, McSplit and McSplit+RL proposed label class and a learning based scoring mechanism for the branching nodes, respectively.

### 3 Branch and Bound for MCS

This section presents two of the best-performing BnB algorithms for MCIS and MCCS, McSplit [McCreesh et al., 2017] and McSplit+RL [Liu et al., 2020]. To simplify the algorithm description, we initially assume that the graphs are undirected, as the method is easy to be adapted to handle various extensions of the problem [McCreesh et al., 2017].

At each search node, the BnB algorithm first estimates an upper bound of the best solution that can be found in the current subtree. If the upper bound is not larger than the size of the current best solution, the algorithm prunes this branch and backtracks, because any better solution cannot be found under this search node. Otherwise, it selects a new vertex pair to match, updates the current solution and then runs recursively. There are three key issues for implementing the BnB algorithm: 1) estimate the upper bound, 2) design the branching strategy, 3) maintain candidate vertices of the two graphs.

To address these issues, McSplit and McSplit+RL proposed label class and a learning based scoring mechanism for the branching nodes, respectively.

#### 3.1 The BnB Framework for MCS Problems

[McCreesh et al., 2017] proposes a BnB framework for MCIS problems, that the current best-performing algorithms McSplit and McSplit+RL both adopt. For two input graphs $G_0$ and $G_1$, the bidomain structure consists of two vertex sets $\langle V_l, V_r \rangle$, where $V_l$ and $V_r$ are subsets of $V(G_0)$ and $V(G_1)$, respectively. McSplit assigns a label to each of the bidomains, the label indicates that each vertex in the bidomain has the same connectivity to the matched vertices, and is represented by a “01”-string, see Figure 1 for an illustrative example. Therefore the two graphs can be represented as a list of bidomains $D$ during the search and any vertex pair selected from each bidomain is legal to match. Whenever a vertex pair $(p, q)$ matches, each bidomain $\langle V_l, V_r \rangle$ will be divided into two new bidomains $(V_{l0}, V_{r0})$ and $(V_{l1}, V_{r1})$, where the superscript “0” and “1” indicate the connectivity of the matched vertex pair $(p, q)$.

Obviously, a bidomain $\langle V_l, V_r \rangle$ can provide at most $\min(|V_l|, |V_r|)$ matched vertex pairs. Therefore, the algorithm estimates the upper bound of the bidomain list $D$ by the following equation:

$$\text{overEstimate}(D) = \sum_{\langle V_l, V_r \rangle \in D} \min(|V_l|, |V_r|) \quad (1)$$

Then, we introduce the flow of BnB algorithm based on
Figure 1: An example to illustrate the concepts of bidomain and label class for MCS. There are three matched vertex pairs \((1, a), (2, b)\) and \((3, c)\). According to the definitions of bidomain and label class, the remaining vertices are divided into three valid bidomains \(D(000^\prime) = \{\{6\}, \{f\}\}\), \(D(100^\prime) = \{\{7, 8\}, \{d\}\}\) and \(D(110^\prime) = \{\{4\}, \{e\}\}\), where “0” indicates that the vertex is not adjacent to the corresponding matched vertex and “1” otherwise. Note that \(D(011^\prime)\) and \(D(111^\prime)\) are invalid bidomains, as they cannot provide any matched vertex pair, and will be removed in the bidomain dividing process.

the depth-first search for MCS, as depicted in Algorithm 1. The call of \(\text{MCS}((V(G_0), \{V(G_1)\}), \alpha, \beta, \gamma, \emptyset, \emptyset)\) returns a maximum common induced subgraph of \(G_0\) and \(G_1\) where \(\alpha, \beta\) and \(\gamma\) are the heuristics. At each search node, the algorithm calculates the upper bound assisted by Eq. 1 and uses the upper bound to determine whether it is possible to find a better solution from this search node. If there exists no better solution from this node, the algorithm prunes and backtracks. Then it selects a bidomain \((V_l, V_r)\) from the current bidomain list \(D\) by heuristic \(\alpha\) and picks a vertex \(p\) from \(V_l\) by heuristic \(\beta\). The algorithm enumerates all the vertices in \(V_r\) to match with vertex \(p\) where the order of enumeration is decided by heuristic \(\gamma\). When a vertex pair \((p, q)\) is matched, the algorithm appends the matched pair to the current solution, updates the best solution, and obtains a new bidomain list \(D'\) by dividing the current bidomain list \(D\) by \((p, q)\). Afterwards, the algorithm runs recursively with the new bidomain list \(D'\). After enumerating all the possible matched vertex pairs of \(p\), the rest of the configuration is that vertex \(p\) does not appear in the matched vertex pair list, thus the algorithm removes vertex \(p\) from the current bidomain list and runs recursively.

3.2 McSplit and McSplit+RL

There are three heuristics \(\alpha, \beta\) and \(\gamma\) that represent the strategy of selecting the bidomain, the first matched vertex and the order of enumerating the second matched vertex, respectively. These heuristics are the essential components in the MCS BnB framework, leading to different BnB algorithms.

McSplit [McCreesh et al., 2017] implements the three heuristics as follows:

- \(\alpha\): McSplit defines the value of \(\max(|V_l|, |V_r|)\) as the size of the bidomain. It selects a bidomain with the smallest size from the bidomain list and uses the largest vertex degree in \(V_l\) to break ties.
- \(\beta\): Picks the largest degree vertex from \(V_l\) as the first matched vertex.
- \(\gamma\): Enumerates the second matched vertex in \(V_r\) in decreasing order of vertex degree.

McSplit+RL [Liu et al., 2020] introduces a learning based branching heuristic, so as to choose a branching pair with the largest bound reduction and reach the pruning condition as early as possible. McSplit+RL regards the BnB algorithm as an agent and a vertex pair selection as an action. When a vertex pair \((p, q)\) is matched, and the current bidomain list \(D\) is divided into \(D'\), McSplit+RL uses the estimated bound reduction of the bidomain list as the reward \(r(p, q)\) of taking action \((p, q)\).

\[
r(p, q) = \text{overEstimate}(D) - \text{overEstimate}(D').
\]  

McSplit+RL maintains two score lists \(S_0(\cdot)\) and \(S_1(\cdot)\) for each vertex in \(G_0\) and \(G_1\), and records the accumulated rewards of each vertex. Specifically, it updates the score lists as follows:

\[
\begin{align*}
S_0(p) &\leftarrow S_0(p) + r(p, q) \\
S_1(q) &\leftarrow S_1(q) + r(p, q)
\end{align*}
\]  

Then McSplit+RL replaces heuristics \(\beta\) and \(\gamma\) based on the two score lists as follows:

- \(\beta\): Picks a vertex \(p\) with the largest score \(S_0(p)\) from \(V_l\) as the first matched vertex.
- \(\gamma\): Enumerates the second matched vertex \(q\) in \(V_r\) in decreasing order of the \(S_1(q)\) score.

4 The Proposed McSplit+LL Algorithm

We first analyze the limits of the existing branching heuristic, and then propose a new branching heuristic called Long-Short Memory (LSM). Furthermore, we propose a Leaf vertex Union Match (LUM) method for the vertex-vertex mapping based algorithm. We implement both LSM and LUM on top of McSplit, and call the resulting algorithm McSplit+LL.

4.1 Further Discussion on McSplit(+RL)

In McSplit, the vertex-pair selection heuristics \(\beta\) and \(\gamma\) are straightforward but not adaptive, hence its branching strategy may not be the best choice to minimize the search tree size. In McSplit+RL, the main idea for branching is to use the bound reduction to evaluate each vertex. Then it selects a vertex with the largest evaluation score to match, aiming to reduce the bound as much as possible and quickly reach the pruning condition. The score is accumulated during the entire BnB algorithm, i.e., is the summation of all the historical evaluation values. However, with the increment on the number of recursion and backtracking, the situation of the current solution and unmatched vertices (the bidomains) in the two graphs change drastically. Accumulating the scores evenly may lead to a bias caused by a large proportion of historical evaluation values when the current configuration differs from the historical configuration. Hence, we need a mechanism that can reasonably eliminate the influence of historical evaluations.

1910
4.2 Long-Short Memory Branching Heuristic

To overcome the limits of existing BnB methods, we apply the reinforcement learning method in McSplit+RL to evaluate the benefit of matching each vertex on the reduction of the search tree size, and further propose a mechanism to eliminate the historical evaluations by decaying part of the evaluation values when they reach a predetermined upper limit.

Our method maintains a score list \( S_t(\cdot) \) as McSplit+RL does, and further maintains a score table \( S_t(p, q) \) of each vertex pair \( (p, q) \) simultaneously. The initial value of the score table \( S_t(p, q) \) is set to 0 for each vertex pair \( (p, q) \). Whenever action \( (p, q) \) is performed, \( S_t(p, q) \) will be updated by the following equation:

\[
S_t(p, q) = S_t(p, q) + r(p, q).
\] (4)

Our method replaces the \( \gamma \) heuristic in McSplit+RL with:

- \( \gamma \): Enumerates the second matched vertex \( q \) in \( V_r \) in decreasing order of the \( S_t(p, q) \) score.

Before introducing our decaying mechanism, we first provide insights on the scoring mechanism. Firstly, score \( S_t(p) \) is accumulated by reward \( r(p, q) \) where vertices \( p \) and \( q \) are in the same bidomain. The bound reduction is seriously influenced by matching to which vertex \( q \), and the bidomains are changed frequently that means the vertices matched to \( p \) are very dynamic. It will lead the evaluation score to be outdated quickly. Thus, it is very inaccurate to use score \( S_t(p) \) to evaluate the bound reduction of taking action \( (p, q) \). Secondly, the evaluation score on vertex pair \( S_t(p, q) \) records the reward \( r(p, q) \) properly, which is more accurate than \( S_t(p) \).

So the historical rewards accumulated by \( S_t(p, q) \) have significant reference value. So we propose an LSM strategy to make the score list \( S_0(\cdot) \) focus on the short-term reward, and the score table \( S_t(\cdot, \cdot) \) focus on the long-term reward.

In LSM, a short-term threshold value \( T_s \) (10^5 by default) and a long-term threshold value \( T_l \) (10^9 by default) are used to implement our mechanism. At each search node, when the algorithm is branching and scores \( S_t(p) \) and \( S_t(p, q) \) are updated, if score \( S_t(p) \) is greater than \( T_s \), then all the scores in the score list \( S_t(\cdot) \) will be halved. And we regard the score table \( S_t(\cdot, \cdot) \) as \( |V(G_0)| \) score lists, i.e., \( S_t(\cdot, \cdot) = S_t(v_1, \cdot), S_t(v_2, \cdot), ..., S_t(v_{|V(G_0)|}, \cdot) \). If score \( S_t(p, q) \) is greater than \( T_l \), then all the scores in the score list \( S_t(p, \cdot) \) will be halved. The decaying operation in each score list of the score table is independent.

4.3 Leaf Vertex Union Match Strategy

We first provide a definition of the leaf vertex in an undirected graph. A vertex is regarded as a leaf if it is adjacent to exactly one vertex in the graph and the leaves of vertex \( u \) indicate the leaves adjacent to \( u \). Then, we provide the main theory to support our LUM strategy.

Theorem 1. In the MCS BnB framework, when a vertex pair \( (p, q) \) is matched, we can match as many leaf pairs as possible from the unmatched leaves of vertices \( p \) and \( q \) without affecting the solution optimality.

Proof. Assume the current solution has \( k \) matched vertex pairs and the \( i \)-th pair is \( (p, q) \). According to the label class definition in Section 3.1, the labels of the unmatched leaves of \( p \) and \( q \) are all the same as “0...010...0” which has exactly one “1” in the \( i \)-th position, because the leaves are only adjacent to the vertex \( p \) and \( q \) respectively. Thus, these leaves are always partitioned in the same bidomain. It guarantees that no matter what the matching configuration of other vertices is, the leaves of \( p \) and leaves of \( q \) are always legal to match as long as \( p \) and \( q \) match together. If arbitrary pair of leaves are matched, all the remaining unmatched vertices are not adjacent to both matched leaves. Matching any leaf pair does not divide a new bidomain from the current bidomain list. Therefore, such an operation does not affect the solution optimality of the MCS BnB algorithm.

Our LUM strategy can be extended to general graph matching problems. Consider a directed and labelled graph, where both vertices and arcs are labelled. Let \( L_v = \{a_1, a_2, a_3, ...\} \) and \( L_e = \{b_1, b_2, b_3, ...\} \) represent the vertex label set and arc label set, respectively. We give each leaf an attribute \( (a_i, \pm b_j) \) where \( a_i \) is the vertex label, \( b_j \) is the label of arc connected to the leaf and the sign represents the direction of arc (+ means leaf is head, − means tail). We partition the leaves into different groups according to the attributes. Leaves in the corresponding group can be matched, therefore it can match as many leaf pairs as possible in each group. We also provide an example as shown in Figure 2. Note that the leaf grouping is independent of the bidomain dividing.

5 Experiments

We compare our proposed McSplit+LL algorithm with McSplit+RL [Liu et al., 2020], which is the state-of-the-art BnB exact algorithm for MCS. We test the algorithms on both the MCS and MCCS problems. The experimental results show that McSplit+LL outperforms McSplit+RL for solving these problems. Further ablation studies show the effectiveness of our proposed methods, including the LSM branching heuristic and the LUM strategy. The source code of McSplit-LL is available at https://github.com/JHL-HUST/McSplit-LL.

5.1 Experimental Setup

Experiments were performed on a server with Intel® Xeon® E5-2650 v3 CPU and 256GBytes RAM. The algorithms were
implemented in C++ and compiled using g++ 5.4.0. The cut-off time is set to 1,800 seconds for each instance, which is the same as in [Liu et al., 2020].

There are two parameters, $T_s$ and $T_l$ for the LSM branching heuristic. We sample some instances solved between 10 to 30 minutes and the parameter tuning domain is set to $10^k$, $k \in [2, 9]$. Finally, the default setting of two parameters $T_s$ and $T_l$ are $10^5$ and $10^9$, respectively.

5.2 Benchmark Datasets

The benchmark datasets\(^1\) include 25,552 instances that are divided into several compositions:

- Biochemical reaction [Gay et al., 2014] includes 136 directed bipartite graphs (with vertices between 9 and 386), and describe biochemical reaction networks originated from the biomodels.net. It provides 9,180 instances obtained by pairing each of the graphs.
- Images-PR15 [Solnon et al., 2015] includes a target graph (with 4,838 vertices) and 24 pattern graphs (with vertices between 4 and 170), which are generated from segmented images. It provides 24 instances.
- Images-CVIU11 [Damiaand et al., 2011] includes 43 pattern graphs (with vertices between 22 and 151), and 146 target graphs (with vertices between 1,072 and 5,972), which are generated from segmented images. It provides 6,278 instances.
- Meshes-CVIU11 [Damiaand et al., 2011] includes 6 pattern graphs (with vertices between 40 and 199), and 503 target graphs (with vertices between 208 and 5,873), which are generated from meshes modelling 3D objects. It provides 3,018 instances.
- Scalefree [Zampelli et al., 2010; Solnon, 2010] includes 100 instances. Each instance is composed of a target graph (with vertices between 200 and 1,000) and a pattern graph (contains 90% of the vertices of the target graph). These instances are randomly generated from scale-free networks.
- Si [Zampelli et al., 2010; Solnon, 2010] includes 1,170 instances. Each instance is composed of a target graph (with vertices between 200 and 1,296) and a pattern graph (with vertices between 20% and 60% of the target graph). Among these instances, there are bounded valence graphs and modified bounded valence graphs.

\(^1\)Available at http://liris.cnrs.fr/csolnon/SIP.html.
4D meshes, and randomly generated graphs.

- LV [McCreesh et al., 2017] includes 49 pattern graphs and 48 target graphs (with vertices both between 10 and 128). It provides 2,352 instances.
- LargerLV [McCreesh et al., 2017] includes 49 pattern graphs (with vertices between 10 and 128) and 70 target graphs (with vertices between 138 and 6,671). It provides 3,430 instances.

5.3 Comparisons on MCS

The McSplit algorithm is efficient enough and there are lots of small scale instances in the benchmark datasets that can be solved in several seconds. The results on these small scale instances cannot really show the gap of differences on the algorithm efficiency. Hence, we group the 7,226 small scale instances which could be solved by all the tested algorithms within 10 seconds into the easy set, and only provide the average solving time. We also exclude another set of tough instances which cannot be solved by any of McSplit+RL, McSplit+LL and their variants discussed in the ablation study within the time limit. Thereafter, we have 2,309 remaining instances that can be solved by at least one of the tested MCS algorithms within the time limit. We denote these medium hard instances as the moderate instances, and will use them for detailed performance comparison.

We compare McSplit+RL with McSplit+LL as illustrated in Figure 3. Each point \((t, n)\) in a curve indicates the algorithm solves (finds and proves the optimal solution, or only proves the optimal solution) \(n\) instances in \(t\) seconds, the same in Figures 4, 5, and 6. McSplit+RL solves 2,013 moderate instances while McSplit+LL could solve 2,208 moderate instances. Besides, the average runtimes of McSplit+RL and McSplit+LL on the easy set of instances are 0.83s and 0.51s, respectively. In other words, McSplit+LL solves 9.69% more moderate instances than McSplit+RL, and McSplit+LL is also faster than McSplit+RL in solving the easy instances. The results demonstrate that McSplit+LL outperforms McSplit+RL significantly on the MCS problem.

5.4 Comparisons on MCCS

We also apply McSplit+LL to solve the MCCS problem. As the basic version of McSplit does not support solving MCCS on the directed graph, we exclude the directed graph instances (9,180 Biochemical reaction instances) from the datasets.

Using the same datasets processing as in MCS, we first exclude the 2,110 easy instances that solved by two algorithms within 10 seconds and the tough instances that cannot be solved by any of the two algorithms, and use the remaining 1,166 moderate instances as benchmarks for detailed comparison. The results are illustrated in Figure 4. McSplit+RL solves 1,064 moderate instances, while McSplit+LL solves 1,158 moderate instances (8.83% more than McSplit+RL). Also, the average runtimes of McSplit+RL and McSplit+LL on the easy instances are 2.03s and 1.36s, respectively. The results demonstrate that McSplit+LL also outperforms McSplit+RL significantly on the MCCS problem.

5.5 Ablation Study

This subsection does ablation studies to analyze the effectiveness of the two proposed operators, LSM and LUM.

We first compare three algorithms, McSplit+RL, McSplit+SM and McSplit+LSM, on the MCS instances and the results are illustrated in Figure 5. McSplit+SM is a variant of McSplit+RL that applies our decaying operation on the score lists \(S_0(t)\) and \(S_1(t)\) with the short-term threshold value \(T_s\), called Short Memory (SM). McSplit+LUM is a variant that applies LSM on top of McSplit. McSplit+SM and McSplit+LSM solves 2,079 and 2,179 moderate instances respectively, which are 3.28% and 8.25% more than McSplit+RL. Besides, the average runtimes of McSplit+SM and McSplit+LSM on the easy instances are 0.82s and 0.53s, respectively. The results show that McSplit+RL can be improved by applying our decaying mechanism, and using LSM instead of SM yields better performance.

We then implement LUM on top of McSplit+RL and obtain a variant algorithm called McSplit+LUM. We compare McSplit+LUM with McSplit+RL on the MCS instances and the results are illustrated in Figure 6. It solves 2,059 moderate instances which is 2.29% better than McSplit+RL. Besides, the average runtime of McSplit+LUM on the easy set of instances was 0.81s.

The ablation studies demonstrate that LSM and LUM could improve the performance and efficiency of the BnB algorithm for both MCS and MCCS problems.

6 Conclusion

We address the Maximum Common induced Subgraph (MCS) and Maximum Common Connected induced Subgraph (MCCS) problems, and propose an strengthened Branch-and-Bound (BnB) algorithm called McSplit+LL that incorporates two new operators into the effective BnB algorithm McSplit. The first one is a new branching heuristic called Long-Short Memory (LSM). LSM finds the different properties of scoring each vertex and each vertex pair, and applies a reasonable policy to make the score of each vertex focus on the short-term reward and the score of each vertex pair focus on the long-term reward. The second one is a general operator called Leaf vertex Union Match (LUM). LUM allows the common subgraph match multiple leaf pairs whenever a vertex pair is matched, so as to speed up the overall matching while not affecting the solution optimality. Both LSM and LUM can improve the efficiency of BnB algorithm. Besides, LUM is a general vertex-vertex mapping method and could be applied to other graph matching problems.

We do extensive experiments on public instances to evaluate the performance of the proposed algorithm McSplit+LL, and the effectiveness of the two proposed operators LSM and LUM. The results show that McSplit+LL significantly outperforms the best-performing algorithm McSplit+RL for both MCS and MCCS, and both LSM and LUM can improve the efficiency and performance of the BnB algorithm.

Acknowledgements

This work is supported by National Natural Science Foundation of China (62076105).
References


