Journal of Artificial Intelligence Research 71 (2021) 89-119 Submitted 01/2021; published 05/2021

Efficient Local Search based on Dynamic Connectivity Maintenance for Minimum Connected Dominating Set

Xindi Zhang ZHANGXD@IOS.AC.CN
Bohan Li LIBOHAN19@MAILS.UCAS.AC.CN
Shaowei Cai SHAOWEICAI.CS@GMAIL.COM
State Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences
Beijing, China
School of Computer Science and Technology
University of Chinese Academy of Sciences
Beijing, China

Yiyuan Wang YIYUANWANGJLU@126.COM
School of Computer Science and Information Technology
Northeast Normal University, China
Key Laboratory of Applied Statistics of MOE
Northeast Normal University, China

Abstract

The minimum connected dominating set (MCDS) problem is an important extension of the minimum dominating set problem, with wide applications, especially in wireless networks. Most previous works focused on solving MCDS problem in graphs with relatively small size, mainly due to the complexity of maintaining connectivity. This paper explores techniques for solving MCDS problem in massive real-world graphs with wide practical importance. Firstly, we propose a local greedy construction method with reasoning rule called 1hopReason. Secondly and most importantly, a hybrid dynamic connectivity maintenance method (HDC+) is designed to switch alternately between a novel fast connectivity maintenance method based on spanning tree and its previous counterpart. Thirdly, we adopt a two-level vertex selection heuristic with a newly proposed scoring function called *chrono-safety* to make the algorithm more considerate when selecting vertices. We design a new local search algorithm called FastCDS based on the three ideas. Experiments show that FastCDS significantly outperforms five state-of-the-art MCDS algorithms on both massive graphs and classic benchmarks.

1. Introduction

With a series of computational challenges brought by the rapid increase of massive data in recent years, most existing algorithms become ineffective when solving NP-hard problems on massive data sets. Thus, in the last decade, many researchers devoted their efforts to developing new algorithms to deal with massive real-world graphs. In this paper, we consider an interesting graph theory problem, namely the minimum connected dominating set (MCDS) problem, and propose a series of effective techniques focusing on solving MCDS problem in massive graphs.

Given an undirected connected graph $G = (V, E)$, a set $D \subseteq V$ is called a *dominating set* if each vertex in V either belongs to D or is adjacent to at least one vertex in D. The minimum dominating set (MDS) problem is to find a dominating set with the minimum number of vertices in the given graph. An important generalization of MDS is the MCDS problem, whose goal is to find a MDS that forms a connected subgraph in the given graph. An important application of MCDS problem is to generate a virtual backbone in wireless networks (Yu et al., 2013) such as mobile ad hoc networks (Al-Karaki & Kamal, 2008), wireless sensors networks (Misra & Mandal, 2009) and vehicular ad hoc networks (Chinnasamy et al., 2019). Specifically, MCDS problem plays a significant role in broadcast routing (Cheng et al., 2006; Ni et al., 1999; Wu & Dai, 2003), power managing (Chen et al., 2002; Deb et al., 2003) and fiber optical networks (Chen et al., 2010; Sen et al., 2008). In addition, MCDS problem has also lots of applications in many other fields, such as system biology (Milenković et al., 2011) and aviation network (Li et al., 2020b). Moreover, MCDS problem is equivalent to the maximum leaf spanning tree problem (Lucena et al., 2010; Fernau et al., 2011; Binkele-Raible & Fernau, 2012; Solis-Oba et al., 2017).

1.1 Previous Works

It is well known that MCDS problem is NP-hard (Kann, 1992). Several exact algorithms (Fomin et al., 2008; Simonetti et al., 2011; Fan & Watson, 2012; Gendron et al., 2014) and approximation algorithms (Cheng et al., 2003; Ruan et al., 2004; Khuller & Yang, 2019) have been designed for MCDS problem. For example, the improved 1-hop and 2-hop local information greedy algorithms were proposed in (Khuller & Yang, 2019), with approximation ratio of $H(\Delta_G) + 2\sqrt{H(\Delta_G)} + 1$ and $H(2\Delta_G + 1) + 1$ respectively, where H is the harmonic function and Δ_G is the maximum degree in the graph. Nevertheless, these algorithms are either too time-consuming or have poor performance in practice, especially in the context of massive graphs.

Because of its NP-hardness, much of the research effort in the past decade concerned with solving MCDS problem has focused on heuristic algorithm aiming to obtain a good solution within a reasonable time. Two algorithms called MCDS/SA and MCDS/TS based on simulated annealing and tabu search were proposed (Morgan & Grout, 2007). Hedar and Ismail designed a simulated annealing algorithm with stochastic local search for MCDS problem (Hedar & Ismail, 2012). Later, Jovanovic and Tuba designed an ant colony optimization algorithm with a so-called pheromone correction strategy (Jovanovic & Tuba, 2013). A greedy random adaptive search procedure that incorporated a local search procedure based on a greedy function and tabu search was described (Li et al., 2017). Wu et al. used a restricted swap-based neighborhood to improve the tabu search procedure, resulting in the RNS-TS algorithm (Wu et al., 2017). Two meta-heuristics based on genetic algorithms and simulated annealing were designed to solve MCDS problem (Hedar et al., 2019). Li et al. presented a multi-start local search algorithm called MSLS based on three mechanisms including a vertex score, configuration checking, and vertex flipping (Li et al., 2019). Finally, a meta-heuristic algorithm called ACO-RVNS (Bouamama et al., 2019) was proposed, based on ant colony optimization and reduced variable neighborhood search. Moreover, ACO-RVNS has also been applied to weighted version of MCDS problem. Experiments showed that, for classic graphs with fewer than 5000 vertices, RNS-TS, MSLS,

and ACO-RVNS obtained similar state-of-the-art performance, while the performance of these algorithms was different on solving massive graphs.

1.2 Contributions

Previous MCDS algorithms have made progress in solving classic graphs, but these algorithms still cannot handle massive graphs with millions of vertices. In this work, we focus on solving MCDS problem on massive graphs and propose an efficient local search algorithm called FastCDS. The main technical contributions of this paper are as follows.

First, we propose a construction algorithm based on reasoning rule called 1hopReason. According to simple preprocessing, vertices are divided into three sets, including a set of vertices that are contained in each feasible solution, a set of vertices that are not contained in any optimal solution, and the third set consisting of the remaining vertices. The 1hopReason algorithm constructs a solution according to the reasoning rule which exploits the different parts of vertices.

The second idea, which is also the most important contribution of this work, concerns with the connectivity maintenance. For combinatorial optimization problem with connectivity constraint, a key factor to the performance is the connectivity maintenance method, especially when the graphs are large. Previous connectivity maintenance methods make the algorithm explore rather large parts of the search space, but they become futile when faced with very large search space. To overcome this issue, we propose a novel tree-based connectivity maintenance (TBC for short) method, inspired by spanning trees. Compared to previous counterparts, the TBC method has very low complexity, while limiting the algorithm to explore a relatively small part of the search space. To obtain a balance between diversification and intensification, we design the a hybrid dynamic connectivity maintenance method (HDC) method to dynamically switch between previous connectivity maintenance methods and TBC during the search process. Furthermore, HDC is enhanced by reconstructing spanning tree when the algorithm falls into local minima for certain time, resulting in HDC+ method.

Finally, we designed a two-level vertex selection heuristic. By defining a new vertex property called *safety*, which takes into account the difference among dominated vertices and the diversity of solution, we propose a novel vertex scoring function called *chrono-safety* based on different scoring functions including the *safety* property and history information of vertices. Comparing to previous MCDS algorithms which only use one scoring function, the proposed two-level vertex selection rule uses the *chrono-safety* scoring function as a supplement of traditional scoring function to make itself more considerate.

Extensive experiments are carried out to evaluate FastCDS on classic benchmarks used in previous literature and on massive graphs from real-world applications. Experimental results indicate that FastCDS outperforms five state-of-the-art MCDS heuristic algorithms on most instances, and confirm the effectiveness of our proposed strategies.

This paper is an extended and improved version of a conference paper (Li et al., 2020a), where a local search algorithm for MCDS problem called NuCDS is proposed. The HDC method and the safety-based vertex selection heuristic (Li et al., 2020a) have been presented in the conference version of this article. The new contributions in this article are summarized as follows:

- A new effective construction method called 1hopReason;
- An improved version of HDC method called HDC+;
- A two-level vertex selection method with the optimized *chrono-safety* scoring function;
- More experiments to evaluate the performance of our algorithm, comparing it with five state-of-the-art algorithm, including NuCDS;
- Comprehensive assessment for each strategy.

1.3 Paper Organization

Some preliminary knowledge is introduced in Section 2. In Section 3, we propose an efficient construction method with reasoning rule called 1hopReason. In Sections 4 and 5, we describe the two main ideas in the local search process, including the connectivity maintenance method HDC+ and the two-level vertex selection heuristic with *chrono-safety* as the secondary scoring function. We describe the FastCDS algorithm in Section 6. Experimental results and further analyses are presented in Section 7. Finally, we give some concluding remarks in Section 8.

2. Preliminaries

In this section, we introduce the basic definitions and notations that will be used in this paper, and then we briefly overview two strategies adopted in FastCDS, called two-level configuration checking and best from multiple selection.

2.1 Basic Definitions and Notation

An undirected graph $G = (V, E)$ consists of a vertex set V and an edge set E. For an edge $e = \{u, v\}$, vertices u and v are the endpoints of the edge. The distance between two vertices u and v, denoted by $dist(u, v)$, is the number of edges in a shortest path from u to v. For vertex v, its ith level neighborhood is $N_i(v) = \{u | dist(u, v) = i\}$, and its ith level closed neighborhood is $N_i[v] = N_i(v) \cup \{v\}$. We denote $N^k(v) = \bigcup_{i=1}^{k} N_i(v)$. Also, the ith level neighborhood and the closed neighborhood of a vertex set $S \subseteq V$ are defined as $N_i(S) = \bigcup_{v \in S} N_i(v) \setminus S$ and $N_i[S] = \bigcup_{v \in S} N_i[v]$, respectively. The first level neighborhood symbol N_1 is usually denoted as N. The degree of a vertex v in G, denoted as $d(v)$, is defined as $|N(v)|$. For the given graph G, Δ_G is the corresponding maximum degree. $G[S] = (V_S, E_S)$ is a subgraph in G induced by S such that $V_S = S$ and E_S consists of all the edges in E whose endpoints are in S.

An undirected graph $G = (V, E)$ is connected when it has at least one vertex and there is a path between every pair of vertices. For the given graph G, a clique C of G is a subset of V where each pair of vertices in C is adjacent.

Definition 1 *Given an undirected connected graph G, a vertex in G is an articulation vertex iff removing it, together with the edges connected to it, disconnects the graph. The articulation vertex set of G is denoted as $A(G)$.*

Given a graph $G = (V, E)$, for a vertex set $D \subseteq V$, a vertex $v \in V$ is dominated by D if $v \in N[D]$, and is non-dominated otherwise. If D dominates all vertices, then D is a dominating set, and if its induced subgraph is also connected then it is a connected dominating set. For a given graph G, the aim of the minimum connected dominating set (MCDS) problem is to find the connected dominating set D with the smallest size.

During the search process, we use $D \subseteq V$ to denote a candidate solution. If the algorithm only considers vertices in the $N_i(D)$ to be moved (i.e. added into or removed from the candidate solution), then we say that it uses i-hop information.

2.2 Two-Level Configuration Checking

Configuration checking (CC) (Cai & Su, 2011) is a strategy aiming to handle the cycling problem in local search, i.e., revisiting a candidate solution that has been visited recently. It can be briefly described as below. The configuration of a vertex v refers to the states (being selected or not) of all its neighboring vertices. If the configuration of v remains the same as the last time it was removed from the candidate solution, then v is forbidden to be added into the candidate solution. Typically, CC is implemented with a Boolean array $ConfChange$ whose size equals the number of vertices in the given graph.

A variant of CC named two-level configuration checking (CC^2) was proposed to improve a local search algorithm for MWDS (Wang et al., 2017). In CC^2, the configuration of a vertex v is defined to be a vector consisting of states of vertices in $N^2(v)$. A vertex is considered configuration-changed, if the value of any bit of the vector has changed. The CC^2 strategy forbids any vertex to be added into the candidate solution if it is not configuration-changed since its last removal from the candidate solution.

The CC^2 strategy works as follows.

Updating rules:

1. At the beginning of the local search, for each vertex v, $ConfChange[v]$ is set to 1;
2. When vertex v is removed (from current candidate solution), $ConfChange[v]$ is reset to 0 , and $ConfChange[u]$ is set to 1 for all $u \in N^2(v)$;
3. When vertex v is added (into current candidate solution), $ConfChange[u]$ is set to 1 for all $u \in N^2(v)$.

Using rule: When choosing an added vertex v, CC^2 forbids any vertex to be added into the candidate solution if its configuration has not been changed, i.e., $ConfChange[v] = 0$.

2.3 Best from Multiple Selection Heuristic

Local search algorithms usually need to pick an element (e.g., a vertex or variable) from a candidate set. For solving large instances, a commonly used heuristic is the best from multiple selection (BMS) heuristic (Cai, 2015), which is a probabilistic sampling method.

To gain a balance between the quality of the selected element and the time complexity of selection process, BMS randomly chooses k elements from a given candidate set, and then returns the best one according to a selection rule, where k is a parameter. BMS has been proved with theoretical basis that it can return an excellent quality element on large scale instances.

3. An Effective Construction Algorithm for MCDS with Reasoning Rule

In order to handle massive sparse graphs, it is essential to efficiently construct a high-quality initial solution for local search algorithm. We propose a simple and efficient greedy algorithm based on reasoning rules for constructing a feasible solution as the initial solution of local search.

3.1 The Reasoning Rule

Proposition 1 *Given a simple incomplete graph $G = (V, E)$, a vertex v is not contained in any optimal solution of MCDS if its closed neighborhood $N[v]$ forms a clique.*

Proof: Let D^{opt} be an arbitrary optimal solution of MCDS problem for graph G, and v be a vertex such that its closed neighborhood $N[v]$ forms a clique C. Supposing that $v \in D^{opt}$, since D^{opt} is connected and v connects only to vertices in C, there must be at least one vertex $u \in C \setminus \{v\}$ in D^{opt}. Since v is dominated by u, $D^{opt} \setminus v$ is also a connected dominating set, which contradicts with the assumption that D^{opt} is an optimal solution. Thus, $v \notin D^{opt}$. □

For a given graph $G = (V, E)$, the set of vertices V is divided into three sets V_{in}, V_{out} and V_{cand}.

- V_{in} consists of all articulation vertices of G calculated by Tarjan's algorithm (Hopcroft & Tarjan, 1973) , i.e., $V_{in} = A(G)$. These vertices exist in each feasible solution;
- V_{out} denotes a vertex set containing vertices from any clique $C \subseteq V$, which only connect to vertices in C according to the Proposition 1. These vertices are not contained in any optimal solution;
- $V_{cand} = V \setminus (V_{in} \cup V_{out})$ is the remaining vertex set.

Vertices in V_{in} and V_{out} are fixed in and excluded from optimal solutions, respectively. By excluding the vertices in V_{out} and forcing to select vertices in V_{in} for construction, the solution will be constructed from a reduced subset of V. Here, we propose a corresponding reasoning rule as follows.

Reasoning Rule for Construction: When picking vertex to construct an initial solution, the vertices in V_{out} should not be chosen to be added and the vertices in V_{in} should be selected with higher priority than those in V_{cand}.

3.2 The 1*hopReason* Method

Based on the reasoning rule above, we propose a novel construction algorithm named 1*hopReason*, to generate a high-quality initial solution. The psuedo-code of 1*hopReason* is shown in Algorithm 1. The 1*hopReason* algorithm contains a preprocessing method to obtain three vertex sets (lines 1–6) and a local greedy construction method using the reasoning rule for construction (lines 7–14).

At the beginning, Tarjan's algorithm (Hopcroft & Tarjan, 1973) is used to obtain an articulation vertex set $A(G)$ as V_{in}(line 1), while V_{out}, V_{cand} and D are initialized to $\emptyset$ (line 2). In lines 3–6, V_{out} and V_{cand} are generated iteratively by examining the vertices in $V \setminus V_{in}$.

Algorithm 1: Construction Method with Reasoning

Input: Graph $G = (V, E)$
Output: A connected dominating set D

```
1  V_in := A(G) obtained by using Tarjan's algorithm;
2  V_cand := V_out := D := ∅;
3  for ∀v ∈ V \ V_in do
4      if G[N[v]] is a clique then
5          V_out := V_out ∪ {v};
6      else V_cand := V_cand ∪ {v} ;
7  v_add := argmax_{v∈V_cand∪V_in} d(v);
8  D := D ∪ {v_add};
9  while D is not a feasible solution do
10     if N(D) ∩ V_in ≠ ∅ then
11         v_add :=a random vertex in N(D) ∩ V_in;
12     else
13         v_add := argmax_{v∈N(D)∩V_cand} score_cons(v);
14     D := D ∪ {v_add};
15 return D;
```

During each iteration, the algorithm checks whether $N[v]$ is a clique[1] (line 4), and if this is the case, then the vertex is added to V_{out} (line 5). If $N[v]$ is not a clique, v is put into V_{cand} (line 6). To accelerate the generating process, if the algorithm finds a vertex v satisfying the condition, i.e., $G[N[v]]$ is a clique, all vertices with the same degree as v in this clique are put into V_{out} as they meet the condition as well, and the other vertices in $N[v]$ are directly put into V_{cand}. Besides, vertices that have been already assigned to V_{out} or V_{cand} will no longer be visited. The complexity of lines 3–6 is $O(\sum_{v\in V}\sum_{i\in N[v]} d(i)) = O(|V| \times \Delta_G^2)$.

The construction process starts from a set D containing a vertex with the maximum degree (lines 7–8). At each step, it takes all 1-*hop* information into account using the reasoning rule. Specifically, if there exists a vertex v_{add} in $N(D) \cap V_{in}$, the algorithm will choose it first (lines 10–11). Otherwise, the algorithm picks a vertex v in $N(D) \cap V_{cand}$ with the largest $score_{cons}(v)$, which means the number of newly dominated vertices by adding v, breaking ties randomly (Cai, Hou, Wang, Luo, & Lin, 2020) (line 13). The complexity of lines 7–14 is $O(|V| \times \Delta_G)$. Figure 1 gives an example for the construction method.

4. Hybrid Dynamic Connectivity Maintenance Method

In order to handle the performance bottleneck caused by maintaining connectivity in MCDS problem, we introduce a hybrid dynamic connectivity maintenance method (HDC+ for short). After reviewing two main previous connectivity maintenance methods, we propose our novel tree-based connectivity maintenance method, and finally introduce the HDC+

1. It is implemented by checking whether the degree in the induced subgraph $G[N[v]]$ of each vertex in $N[v]$ are all equal to $|N(v)|$. The complexity for checking a vertex v is $O(\sum_{i\in N[v]} d(i))$.

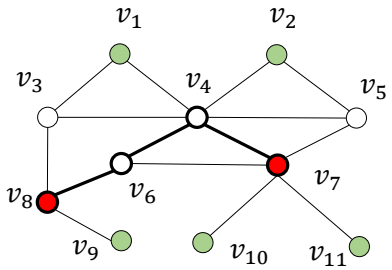


Figure 1: An example of the construction method. Firstly, the vertices set V is divided into three sets, namely V_{in}, V_{out} and V_{cand}, coloring red, green and white respectively. All articulation vertices are selected into V_{in}. There are 5 cliques(vertex set $\{v_1, v_3, v_4\}$, $\{v_2, v_4, v_5\}$, $\{v_8, v_9\}$, $\{v_7, v_{10}\}$ and $\{v_7, v_{11}\}$) where at least one vertex could be selected into V_{out} according to Proposition 1. Then, the algorithm will choose v_4, with the maximum degree, as the first vertex of D. After that, v_7 is selected because it is the only vertex in $N(D) \cap V_{in}$. For the next step, v_6 is chosen randomly from $\{v_3, v_6\}$ because they maximize the number of newly dominated vertices. After selecting v_8, D becomes a feasible solution.

method. For convenience of discussions on complexity, we will use notations $m = |V_D|$ and $n = |E_D|$, where $G[D] = (V_D, E_D)$ is the induced subgraph of current candidate solution D.

4.1 Previous Connectivity Maintenance Methods

Before introducing HDC+, we review two previous methods to handle the connectivity constraint, namely the substraction-based and addition-based methods.

The substraction-based method (SUB for short) is used by previous state-of-the-art MCDS algorithms such as RNS-TS (Wu et al., 2017). Moreover, SUB is applied to solve the weighted version of MCDS problem (Dagdeviren, Aydin, & Cinsdikici, 2017). In order to keep the connectivity of candidate solution D, during each iteration of local search, the algorithm maintains the candidate removal vertex set, defined as $candRemoval(D)=D \setminus A(G[D])$. The traditional approach to computing $A(G[D])$ is called Tarjan's algorithm (Hopcroft & Tarjan, 1973). It works as follows: depth first search (DFS for short) is used to determine whether the child vertex of a vertex u can access the ancestor vertex of u without passing through u.[2] If so, then u is not an articulation vertex, and otherwise it is an articulation vertex. The complexity of the substraction-based method is $O(m + n)$.

The addition-based method has two versions. The first version, used in ACO-RVNS (Bouamama et al., 2019) and in MCDS/TS (Morgan & Grout, 2007), works as follows: during each iteration, the algorithm starts from an empty candidate solution D, and iteratively adds a vertex from $N(D)$ to D until D becomes a feasible solution. The second version, used in GRASP (Li et al., 2017) and MSLS (Li et al., 2019), works as follows: the algorithm allows removing articulation vertices of D, so D may become disconnected. Thus, before vertex u is selected to be added, DFS is used to calculate the number of con-

2. The child vertex of u is the vertex visited directly after u, while father and ancestor vertices of u are the vertices visited directly and indirectly before u by DFS.

nected subgraphs of D that $v \in N(u)$ belongs to, and the vertex with the largest number is preferred. The complexity of both versions is $O(m)$.

The complexity of the above two methods is at least $O(m)$. This makes algorithm time-consuming when applied to massive graphs, hindering the performance on massive graphs.

4.2 Tree-Based Connectivity Maintenance Method

To lower the time complexity of connectivity maintenance, we present a novel tree-based connectivity maintenance (TBC for short) method, which is inspired by spanning trees.

For this purpose, we first introduce the definitions of a spanning tree and a leaf vertex. Given a connected graph $G = (V, E)$, a spanning tree $T = (V', E')$ is defined as a connected subgraph of G with $V' = V$, $E' \subseteq E$, without any cycles. Given a spanning tree $T = (V', E')$, a vertex $v \in V'$ is called a leaf vertex if $d_T(v) = 1$, where $d_T(v)$ denotes the degree of v in T, and the leaf set is defined as $LS(T) = \{v | d_T(v) = 1, v \in V'\}$.

Proposition 2 *Given a spanning tree $T = (V', E')$ and its corresponding leaf set $LS(T)$ of graph $G = (V, E)$, $LS(T)$ is a subset of $V \setminus A(G)$.*

Proof: For any vertex $v \in LS(T)$, after removing v and relevant edges through v from T, the remaining graph is still a spanning tree of $G[V \setminus \{v\}]$. So $G[V \setminus \{v\}]$ remains connected, which means that v is not an articulation vertex of G, inducing that $v \in V \setminus A(G)$. Thus, we can conclude that $LS(T) \subseteq V \setminus A(G)$. □

Based on the definition and proposition above, we describe TBC as follows. Given a candidate solution D, a spanning tree T of $G[D]$ and its corresponding leaf set $LS(T)$ are maintained during the search process. Each vertex $v \in LS(T)$ is allowed to be removed from D. All other vertices are forbidden to be removed. In contrast to the previous substraction-based method, given a current solution D, TBC calculates an approximate $candRemoval(D) = LS(T)$, which is a subset of $D \setminus A(G[D])$. In order to dynamically update $candRemoval(D)$, three updating rules of TBC are described as follows.

Construction Rule: Both T and $LS(T)$ with respect to $G[D]$ are constructed according to the candidate solution D by using breadth first search. The construction method is to set a randomly picked vertex in D as the root of T at first, and to iteratively expand T by a randomly picked vertex in current $LS(T)$ until all vertices have been visited. When a vertex v is picked, for each vertex $u \in (N(v) \cap D) \setminus T$, v is set to the father vertex of u, and u is added to T and the $LS(T)$ should be updated accordingly.

Removing Rule: When vertex v is selected to be removed, its father vertex u needs to be found. If $d_T(u) = 2$, meaning that u becomes a leaf vertex after removing v, $LS(T) = LS(T) \setminus \{v\} \cup \{u\}$. Otherwise, $LS(T) = LS(T) \setminus \{v\}$.

Adding Rule: When vertex v is added to the candidate solution D, one vertex $u \in N(v) \cap T$ needs to be selected as the father vertex of v. In order to render larger $candRemoval(D)$, we prefer not to select a leaf vertex in $LS(T)$, and thus we pick the vertex u with the maximum $d_T(u)$ among $\forall u \in N(v) \cap T$. If u is a leaf vertex, $LS(T) = LS(T) \setminus \{u\} \cup \{v\}$, else $LS(T) = LS(T) \cup \{v\}$. Lastly, v is set as the child of u in T.

The complexity of the construction rule used to construct and reconstruct the spanning tree is $O(m + n)$. When adding a vertex v based on the **Adding Rule**, $N(v)$ has to be

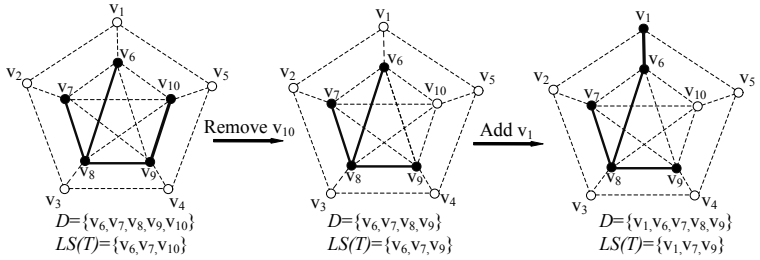


Figure 2: An example of the TBC method. (The solid nodes denote those vertices in the candidate solution D, while the solid edges compose the spanning tree T of $G[D]$.)

searched to update T and $LS(T)$, which has a complexity of $O(\Delta_G)$. When removing a vertex v based on the **Removing Rule**, only the father vertex of v needs to be searched to update T and $LS(T)$, which has a complexity of $O(1)$. To make the TBC method more comprehensive, we present its example in Figure 2 regarding to removing v_{10} and then adding v_1.

4.3 The HDC Method and Its Improved Version HDC+

For large graphs, TBC is substantially faster than previous connectivity maintenance methods. However, since TBC does not consider all candidate removal vertices, it may miss some high-quality options. As an extreme example, a candidate solution D with a vertex $v \in D$ such that $D \setminus \{v\}$ is also a connected dominating set. However, TBC does not consider this option if v is not a leaf vertex of the current spanning tree. In contrast, the substraction-based method using Tarjan's algorithm potentially considers more options for removal because it accurately determines the complete candidate removal vertex set, but its complexity is higher.

In order to take profit from the respective advantages of both methods, a balance must be achieved between the complexity and the accuracy of determining the candidate removal vertex set. We propose a heuristic called hybrid dynamic connectivity maintenance method (HDC) to switch between the two methods. Furthermore, we proposed an improved version called HDC+, by reconstructing the spanning tree when falling into local minima for certain steps.

We first define four parameters as below: 1) parameter $NoImpr$ denotes the maximum number of steps without improving the candidate solution, where $NoImpr \in [MinNoImpr, MaxNoImpr]$; 2) parameter $MinNoImpr$ denotes the minimum value of $NoImpr$, and it is also adopted as the absolute value of the change on $NoImpr$ in each step; 3) parameter $MaxNoImpr$ denotes the maximum number of steps using the current connectivity maintenance method; 4) parameter $reconsGap$ denotes the number of non-improving steps in order to activate a tree reconstruction process.

Considering the difference of complexity between the two connectivity maintenance methods, two sets of parameters are respectively set for SUB and TBC.

Algorithm 2 gives a formal description of the HDC and HDC+ heuristics. $stepNoImpr$ and $stepOneCon$ denote the number of non-improving steps and the number of steps adopting one current connectivity maintenance method, respectively.

Algorithm 2: the HDC+ heuristic

Input: the current solution D
Output: candidate removal vertex set $candRemoval(D)$

```
1  if stepNoImpr > NoImpr||stepOneCon > MaxNoImpr then
2      if curMethod==SUB then
3          curMethod :=TBC;
4          construct a spanning tree T based on Construction Rule;
5      else curMethod :=SUB ;
6      update NoImpr by some tricks;
7      stepOneCon := 0, stepNoImpr := 0;
8  if curMethod==TBC then
       /* HDC does not contain lines 9-10                                  */
9      if stepNoImpr%reconsGap == reconsGap − 1 then
10         reconstruct the spanning tree T based on Construction Rule;
11     candRemoval(D) := LS(T);
12 else
13     compute A(G[D]) based on Tarjan's algorithm;
14     candRemoval(D) := D \ A(G[D])
15 return candRemoval(D);
```

When the candidate solution has not been improved for $NoImpr$ iterations or the current method has been used for $MaxNoImpr$ iterations (line 1), the algorithm switches from the current connectivity maintenance method to the other one (lines 2–5). In particular, after switching to TBC, a spanning tree T of D and its corresponding leaf set $LS(T)$ need to be constructed (line 4). After the switching operation, $NoImpr$ needs to be updated (line 6). Specifically, if $stepNoImpr > NoImpr$, then the algorithm increases $NoImpr$ by $MinNoImpr$ so that it can search more exhaustively for better candidate solutions. Otherwise, if $stepOneCon > MaxNoImpr$, the algorithm decreases $NoImpr$ by $MinNoImpr$ in order to accelerate the search. Both $stepNoImpr$ and $stepOneCon$ should be reset to 0 (line 7). If the current method is TBC and the reconstruction condition is satisfied (line 9), then the spanning tree T will be reconstructed (line 10), which is the main difference between HDC and HDC+ heuristics. Lastly, the algorithm uses the selected connectivity maintenance method to calculate the candidate removal vertex set $candRemoval(D)$ and returns it (lines 8–15).

5. Two-Level Vertex Selection Heuristic

Local search algorithms typically use scoring functions to choose an operation to execute in each step. In this section, we introduce a two-level vertex selection heuristic for local search MCDS algorithms. While the primary scoring function for vertex selection is commonly used for MCDS problem, we propose a secondary scoring function to break ties when the vertices have the same value of the primary function. Note that we use an incremental evaluation method, which is the same as FastDS (Cai et al., 2020), for our scoring function.

5.1 The Primary Scoring Function

In the context of the MCDS problem, a scoring function is used to choose a vertex $u \in D$ for removal and a vertex $v \in N(D)$ for addition. Our algorithm adopts the frequency based scoring function (Wang et al., 2017) as the primary function for this purpose.

For each vertex $v \in V$, the frequency of v is denoted as $freq[v]$, which measures the number of steps that the vertex is not dominated. It can be simply calculated: in the beginning, $freq[v] = 1$ for $\forall v \in V$; then, at the end of each iteration of local search, $freq[v] = freq[v] + 1$ for each non-dominated vertex $v \in V$. The frequency based scoring function, denoted as *score*, is defined as follows:

$$score(v) = \begin{cases} \sum_{u \in C_1} freq[u], & v \notin D \\ -\sum_{u \in C_2} freq[u], & v \in D \end{cases}$$

where C_1 is the set of non-dominated vertices that would become dominated by adding v to D (the current candidate solution), and C_2 is the set of dominated vertices that would become non-dominated by removing v from D.

5.2 The Chrono-Safety Function

In addition to considering the *score* value, we also observe that some dominated vertices are more "endangered" than others, which means that they may become non-dominated more easily. An extreme case is where a vertex is only dominated by one vertex (either by itself or by one of its neighbors). Such vertices may become non-dominated due to one exchanging step. Based on this consideration, we define the *safety* of vertices, taking into account the differences among dominated vertices. We first give the definition of the domination degree.

Definition 2 *Given a connected graph $G = (V, E)$ and a candidate solution D, the domination degree of vertex v is defined as $dd(v) = |N[v] \cap D|$, for each vertex $v \in V$.*

This means that a vertex v with $dd(v) = k$ is dominated by k vertices. Thus, the larger the domination degree, the safer the corresponding vertex. We define a property of vertices named *safety*. The *safety* of a vertex v, denoted as $sf(v)$, is $dd(v)$ if $v \notin D$, or $-dd(v)$ if $v \in D$. Note that we introduce a negative sign for the case $v \in D$ just for the convenience that in both cases we always prefer larger *safety* value.

The safety-based ideas are inspired by the concept of subscore for the SAT problem (Cai & Su, 2013) which considers the satisfaction degree of clauses. Moreover, as far as we know, it is the first time that the definition of *safety* is applied to solve a graph optimization problem.

Besides *safety*, we also consider the additional property of vertex (i.e., *age*) to break ties for the sake of diversification. The *age* of a vertex v is defined as the number of steps since v was last chosen. The default value of *age* for those vertices never chosen before are set to $CurStep$, where $CurStep$ is the number of current steps during local search. During the search process, the algorithm prefers to select a vertex with larger *age* value for diversity.

Combining the two factors above, namely *safety* and *age*, our secondary scoring function is defined as the *chrono-safety* of a vertex v, denoted as $cs(v)$.

$$cs(v) = \begin{cases} age(v) \times sf(v), & v \notin D \\ (CurStep - age(v)) \times sf(v), & v \in D \end{cases}$$

In the above cs function, age dominates the selection strategy if there exist candidate vertices which have not been chosen for a relatively long time. Meanwhile, $safety$ dominates the selection strategy if vertices in candidate selection set are all recently chosen. Note that by introducing $CurStep$ for the case $v \in D$, we always prefer larger cs for both adding and removal.

5.3 Vertex Selection Rule in Our Algorithm

As introduced in Section 2, the BMS heuristic is a common technique in picking a good-quality element from a large candidate set, which is usually used in local search for large combinatorial optimization instances. In this work, we employ a variant of BMS, which gives higher priority to the important vertices (those in V_{in} and V_{out}) by giving them higher probability when sampling. By using the *score* function as the primary scoring function and the *chrono-safety* function as the secondary function, we design a two-level vertex selection heuristic based on BMS for our local search algorithm. It can be described as below.

Vertex Selection Rule: Sample the given candidate set for k times. A vertex in V_{in} (resp. V_{out}) is sampled twice when picking vertex to add (resp. remove). Return the sampled vertex v with the greatest $score(v)$, breaking ties by preferring the one with the greatest $cs(v)$. In our work, the BMS parameter k is set to 100.

In our algorithm, we employ the novel vertex selection rule to decide which vertices to be added or removed. To avoid visiting previous candidate solutions, we use the CC^2 strategy (Wang et al., 2017) in the adding process, and use the tabu strategy (Glover & Laguna, 1998) recording the adding operations to prevent removing a just added vertex for the next tt iterations. In our work, $tt = 3$. There exist an extreme case that all sampled vertices are forbidden by tabu or CC^2 strategies. For this case, the algorithm picks a random vertex from the sampled vertices. In contrast to previous BMS versions which mainly focus on adjusting the k value, the novelty mind of our BMS strategy is to optimize the sample procedure by giving a higher probability for special vertices. Specifically, the selection probability of vertices in V_{in} and V_{out} is twice that of vertices in V_{cand}.

6. A New Local Search Algorithm for MCDS Problem

Based on HDC+ and the *chrono-safety* vertex selection heuristic, we develop a local search algorithm for the MCDS problem called FastCDS. The pseudo-code of FastCDS is shown in Algorithm 3.

In the beginning, FastCDS initializes $score$, sf and age (line 1), and sets $curMethod =$ TBC which means that the TBC method is selected as the initial connectivity maintenance method (line 2). The non-improvement step $stepNoImpr$ and the number of steps used by the current connectivity maintenance method $stepOneCon$ are both set to 0 (line 2). Then, the algorithm constructs the initial candidate solution D using the proposed $1hopReason$ method (line 3). According to the construction rule, the algorithm builds a spanning tree T of D (line 4), and then the candidate removal vertex set $candRemoval(D)$ is initialized as $LS(T)$ (line 5).

During the local search procedure (lines 6–19), the algorithm first uses the HDC+ heuristic to update $candRemoval(D)$ (line 7). If D is a feasible solution, which means that

Algorithm 3: the FastCDS algorithm

Input: An undirected graph $G = (V, E)$, the *cutoff* time
Output: An obtained best solution D^*

```
1  initialize score(v), sf(v) and age(v), for ∀v ∈ V;
2  curMethod :=TBC, stepNoImpr := stepOneCon := 0;
3  D := 1hopReason(G), D* := D;
4  build a spanning tree T according to Construction Rule;
5  candRemoval(D) := LS(T);
6  while elapsed time < cutoff do
7      candRemoval(D) := HDC+(D);
8      if D is a connected dominating set then
9          D* := D, stepNoImpr := 0;
10         select a vertex u from candRemoval(D) using Vertex Selection Rule;
11         D := D \ {u};
12         if curMethod==TBC then update T and LS(T) according to Removing Rule;
13         continue;
14     select a vertex u from candRemoval(D) using Vertex Selection Rule;
15     D := D \ {u};
16     select a vertex v from N(D) ∩ N(G \ N[D]) using Vertex Selection Rule;
17     D := D ∪ {v};
18     if curMethod==TBC then update T and LS(T) according to Removing and
         Adding Rules;
19     stepNoImpr++, stepOneCon++;
20 return D*;
```

the algorithm has already found a connected dominating set of size $|D|$, D^* is updated to D and *stepNoImpr* is set to 0 (line 9). Then, the algorithm continues to find a solution of size $(|D| - 1)$, i.e., by removing a vertex v from D using the vertex selection rule (lines 10–11). If the algorithm selects the TBC method as the current connectivity maintenance method, the corresponding spanning tree T and its leaf set $LS(T)$ should be updated based on the removing rule (line 12).

If D is an infeasible solution, the algorithm tries to exchange two vertices (lines 14–17), i.e., removing a non-tabu vertex u chosen from $candRemoval(D)$ via the vertex selection rule, and then adding a vertex $v \in N(D) \cap N(G \setminus N[D])$ to D via the CC^2 strategy and the selection rule, where $G \setminus N[D]$ is the non-dominated vertex set and $N(D)$ contains vertices maintaining connectivity. Finally, the algorithm needs to update T, $LS(T)$, *stepOneCon*, and *stepNoImpr* accordingly (lines 18–19). When the time limit is reached, the best solution found (D^*) will be returned. The complexity of each iteration in the local search process is $O(|D|)$ or $O(\Delta_G)$ when using SUB or TBC methods respectively.

Moreover, we make use of a trick for the HDC+ heuristic. If $|D| < 100$ (that is, if the complexity of the SUB method is acceptable), using TBC becomes trivial and thus we only use the SUB method under such circumstance.

6.1 The Differences Between FastCDS and NuCDS

An early version of FastCDS called NuCDS has been published in the conference version (Li et al., 2020a). There are three main differences between FastCDS and NuCDS.

- FastCDS adopts an effective construction algorithm based on the reasoning rule namely 1hopReason to generate an initial solution, while NuCDS uses the previous greedy construction algorithm mentioned in (Khuller & Yang, 2019).

- FastCDS improves HDC method proposed in NuCDS, resulting in the HDC+ method. Specifically, when falling into the local minima for certain steps, HDC+ will reconstruct the spanning tree to diversify the search space.

- FastCDS and NuCDS use different selection rules. They combine the age property with $safety$ in different manners. Also, FastCDS assigns different selection priorities to different candidate vertices in V_{in}, V_{out} and V_{cand}.

7. Experimental Evaluation

In this section, we carry out extensive experiments to evaluate the effectiveness of FastCDS. We first introduce the benchmarks, the experiment setup and reporting methodology, so that the readers can understand the experimental parts more easily.

7.1 Benchmarks and Experiment Methodologies

For our experiments, we adopt several popular benchmarks, mainly divided into two groups, including classic benchmarks and massive graphs. We divide classic benchmarks into five groups: the instances of the maximum leaf spanning tree problem (Lucena et al., 2010), bus power flow test cases[3] , random geometric graphs (Jovanovic & Tuba, 2013), and random graphs from (Erdem et al., 2009) and from (Bouamama et al., 2019). In total, 121 classic instances are considered to evaluate the performance of FastCDS.

We also evaluated the algorithms on massive graphs, including massive real-world graphs from the Network Data Repository (NDR) (Rossi & Ahmed, 2015) and Stanford Large Network Dataset Collection (SNAP)[4], as well as large instances from the 10th DIMACS implementation challenge (DIMACS10)[5]. As in previous work (Lin et al., 2017), we only report the results on graphs from the SNAP and DIMACS10 benchmarks with at least 30,000 vertices (with the exception of a few cases with fewer vertices). Directed edges are considered as undirected edges if the instance is directed, and the maximal connected component is chosen if the instance is disconnected, so all instances are undirected connected. We do not report the results on graphs from the NDR benchmark with fewer than 100,000 vertices or fewer than 1,000,000 edges. Hence, we picked 22, 31 and 65 instances from SNAP, DIMACS10, and NDR benchmarks respectively, leading to totally 118 massive graphs.

3. http://labs.ece.uw.edu/pstca
4. http://snap.stanford.edu/data
5. https://www.cc.gatech.edu/dimacs10/

Parameter	Domain	Chosen value
$MinNoImpr$ for SUB	{100,1000}	100
$MinNoImpr$ for TBC	{10000,100000,1000000}	100000
$MaxNoImpr$ for SUB	{2,5,10}×$MinNoImpr$ for SUB	2 ×$MinNoImpr$ for SUB
$MaxNoImpr$ for TBC	{2,5,10}×$MinNoImpr$ for TBC	10 ×$MinNoImpr$ for TBC
$NoImpr$ for SUB	{1,2,5}×$MinNoImpr$ for SUB	1 ×$MinNoImpr$ for SUB
$NoImpr$ for TBC	{1,2,5}×$MinNoImpr$ for TBC	2 ×$MinNoImpr$ for TBC
$reconsGap$	{500,1000}	500

Table 1: Parameter tuning for the preliminary experiments.

We compare FastCDS with five state-of-the-art heuristic algorithms: MSLS (Li et al., 2019), ACO-RVNS (Bouamama et al., 2019), ACO-efficient[6], RNS-TS (Wu et al., 2017) and NuCDS (Li et al., 2020a). The codes of all competitors were kindly provided by the authors. RNS-TS was implemented in Java while FastCDS and the other competitors were implemented in C++ and complied by g++ with '-O3'. Data structures of all competitors were modified to handle massive graphs. All experiments were run on a server with Intel Xeon Platinum 8153 @2.00GHz with 512GB RAM under Centos 7.7.1908.

All algorithms were executed 10 times on each instance independently. The cutoff time was set to 1000 seconds for the classic benchmarks, and 3600 seconds for massive graphs.

The following information is reported: the best size (min), the average size (avg) of the solutions found over the 10 runs. The best min and avg found among the algorithms are shown in **bold**. If an algorithm fails to find a solution within the time limit, the results are marked as 'N/A'. We report the number of instances that get the best solution and the best average solution by #min and #avg. We also list the #feasible information, denoted as the number of instances that a solver can find a feasible solution.

In preliminary experiments, we tried different parameters, and we found that HDC+ is not very sensitive to parameter settings. The tried parameter values and chosen values are presented in Table 1.

7.2 Results on Classic Benchmarks

Most instances of classic benchmarks are so easy that all algorithms obtain the same solution quality very quickly. We ignore these easy instances, but report the average run time when all algorithms obtain the same minimal and average values in Figure 3, which shows the effectiveness of FastCDS. From this figure, FastCDS is more effective than its competitors, while performing a little worse than NuCDS on a few easy instances which can be solved less than 0.1 second. The results of the remaining 55 classic instances are shown in Table 2. FastCDS obtains better solutions than NuCDS, MSLS, ACO-efficient, ACO-RVNS and RNS-TS on 16, 32, 36, 22 and 29 instances respectively, while FastCDS fails to match the solutions obtained by some competitors only on 5 instances. Among the instances where FastCDS generates a solution with the same value as the best competitor, FastCDS obtains better average size on 35 instances with only 3 exceptions.

6. The authors would like to thank Christian Blum, the author of ACO-RVNS, for providing the improved version called ACO-efficient, which is specialized for massive graphs and performs better than ACO-RVNS on most large instances.

Instance	FastCDS		NuCDS		MSLS		ACO_e		ACO		RNS	
	min	avg	min	avg	min	avg	min	avg	min	avg	min	avg
v150_d10	**14**	**14**	**14**	**14**	**14**	14.6	**14**	14.6	**14**	14.5	**14**	**14**
ieee_300_bus	**129**	**129**	130	130.9	**129**	129.2	**129**	129.2	**129**	**129**	**129**	**129**
n1000_200_r100	**38**	**38**	**38**	**38**	39	39.6	39	39.1	**38**	38.2	**38**	38.5
n1000_200_r110	**34**	**34**	**34**	**34**	**34**	34.1	**34**	34.1	**34**	**34**	**34**	**34**
n1000_200_r120	**29**	**29**	**29**	**29**	**29**	30.3	30	30.2	**29**	**29**	**29**	**29**
n1000_200_r130	**26**	**26**	**26**	**26**	**26**	26.8	**26**	26.7	**26**	**26**	**26**	**26**
n1000_200_r140	**23**	**23**	**23**	**23**	**23**	23.1	**23**	23.4	**23**	**23**	**23**	**23**
n1000_200_r150	**21**	**21**	**21**	**21**	**21**	21.1	**21**	21.1	**21**	**21**	**21**	**21**
n1000_200_r160	**19**	**19**	**19**	**19**	**19**	19.6	36	36.3	**19**	**19**	**19**	19.1
n1500_250_r130	**49**	**49**	**49**	**49**	**49**	50	**49**	49.7	**49**	49.1	**49**	**49**
n1500_250_r140	**43**	**43**	**43**	43.7	44	44.3	44	44	**43**	43.9	**43**	43.9
n1500_250_r150	**40**	**40**	**40**	**40**	41	41.6	41	41.6	**40**	40.7	**40**	40.1
n1500_250_r160	**36**	**36**	**36**	**36**	37	37.7	**36**	36.3	**36**	**36**	**36**	**36**
n2000_300_r200	**41**	**41**	**41**	41.5	43	43.1	42	42.5	42	42.1	**41**	41.6
n2000_300_r210	**38**	**38**	**38**	**38**	**38**	38.6	**38**	38.5	**38**	**38**	**38**	**38**
n2000_300_r220	**35**	**35**	**35**	**35**	36	36.3	**35**	36	**35**	35.1	**35**	**35**
n2000_300_r230	**33**	**33**	**33**	**33**	34	34.8	34	34.5	**33**	**33**	**33**	33.2
n2500_350_r200	**59**	**59.4**	60	60	61	61.3	61	62.3	60	60.7	60	60.3
n2500_350_r210	**54**	**54**	**54**	54.9	56	57	57	58.2	55	56.1	55	56
n2500_350_r220	**51**	**51**	**51**	51.1	52	54.3	54	54.7	**51**	52.7	**51**	51.4
n2500_350_r230	**48**	**48**	49	49.1	50	50.6	50	51.7	**48**	49.5	49	49
n3000_400_r210	**74**	**74**	**74**	**74**	**74**	76.3	76	76.6	**74**	75.5	75	75.1
n3000_400_r220	**69**	**69.2**	70	70	71	71.9	71	71.6	70	70.8	70	70.3
n3000_400_r230	**64**	**64**	**64**	64.8	66	67	65	67.1	65	65.9	65	65.1
n3000_400_r240	**60**	**60**	**60**	60.9	61	62.5	62	62.8	61	61.7	61	61.2
n600_100_r110	**14**	**14**	**14**	**14**	**14**	14.5	**14**	14.6	**14**	**14**	**14**	**14**
n700_200_r100	**22**	**22**	**22**	**22**	**22**	22.5	**22**	22.9	**22**	**22**	**22**	**22**
n700_200_r110	**20**	**20**	**20**	**20**	**20**	**20**	**20**	20.5	**20**	**20**	**20**	**20**
n700_200_r120	**17**	**17**	**17**	**17**	**17**	**17**	**17**	17.8	**17**	**17**	**17**	**17**
n700_200_r70	**38**	**38**	**38**	38.1	39	39.6	39	39	**38**	38.1	**38**	38.5
n700_200_r80	**32**	**32**	**32**	**32**	33	33	33	33	**32**	**32**	**32**	**32**
n1000_ep0007	**179**	179.4	**179**	**179**	191	194	187	189	185	186.6	189	190.7
n1000_ep0014	**98**	98.9	**98**	**98**	105	105.5	104	105.3	101	103.1	103	105.3
n1000_ep0028	**59**	**59.6**	**59**	59.9	62	62.5	62	62.8	61	62.8	63	63.6
n1000_ep0056	**36**	**36.3**	37	37	37	37	37	37.8	37	37.8	38	38.2
n1000_ep0112	**21**	**21.6**	22	22	22	22	22	22	22	22	22	22.1
n1000_ep0224	**12**	**12**	**12**	12.5	**12**	**12**	**12**	12.7	**12**	**12**	**12**	12.2
n1000_r0048	276	276.7	**275**	277.3	276	276.5	**275**	**275.2**	**275**	275.8	280	283.7
n1000_r0070	127	127.3	**125**	126.7	**125**	**125**	128	129.3	127	128.4	132	134.4
n1000_r0100	**60**	**60.8**	61	61.8	62	62.4	64	65.4	64	65.4	65	66.1
n1000_r0140	**31**	**31.2**	32	32.3	33	33	33	33.3	32	32.9	34	34.6
n1000_r0207	**15**	**15**	**15**	15.1	16	16	16	16.2	**15**	15.7	16	16.6
n1000_r0308	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	7.7
n5000_ep0007	275	276.3	**264**	**265.6**	277	277.4	277	278.2	277	278.2	392	424.8
n5000_ep0014	**161**	162.3	163	164.2	**161**	**162.1**	164	164.2	165	166.2	206	220
n5000_ep0028	95	95	**94**	94	95	95.1	96	96	95	95.8	121	**25.8**
n5000_ep0056	**54**	**54.9**	56	56	55	55	55	55.6	56	56.1	76	100
n5000_ep0112	**31**	**31**	32	32	**31**	31.4	**31**	31.9	**31**	31.7	2707	2934
n5000_ep0224	**17**	**17**	18	18	**17**	**17**	**17**	**17**	**17**	17.1	3888	4045
n5000_r0048	270	270.3	**268**	**270**	275	275.8	280	284.6	280	284.6	319	336.1
n5000_r0070	**126**	**126**	127	128.4	133	133.4	132	136.4	132	136.2	147	150.3
n5000_r0100	**62**	**62**	63	64.4	68	68.42	70	71.5	72	72.5	74	78.2
n5000_r0140	**32**	**32.4**	33	33.2	36	36.71	37	37.4	36	36.9	39	41
n5000_r0207	**15**	**15.4**	16	16	17	17	18	18.9	16	16.8	2099	2797.3
n5000_r0308	**7**	**7**	**7**	7.6	8	8.28	9	9.5	8	8	3652	3876

Table 2: Results of FastCDS, NuCDS, MSLS, ACO-efficient, ACO-RVNS and RNS-TS on classic benchmarks. To save space, we denote ACO-efficient, ACO-RVNS and RNS-TS as ACO_e, ACO and RNS.

7.3 Results on Massive Benchmarks

Table 3 compares FastCDS with its competitors on massive graphs, and shows the summarized results. The detailed results can be found in Tables 4 and 5. And we report the

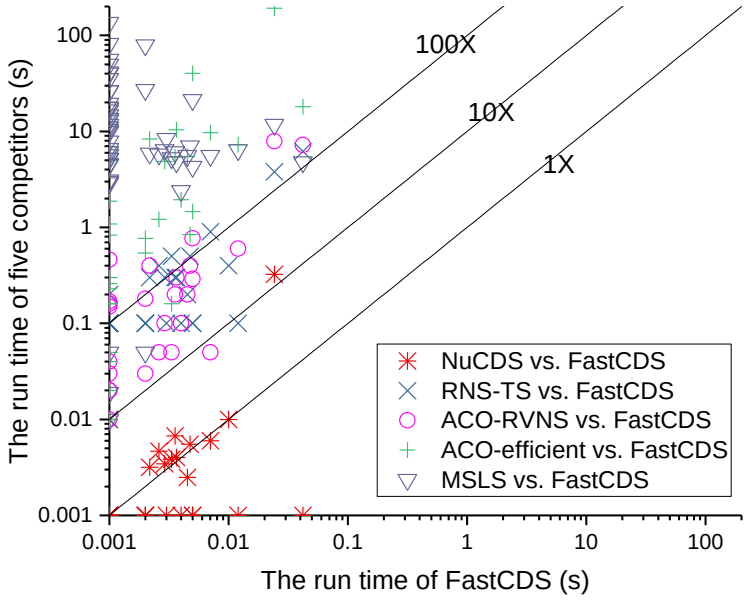


Figure 3: Average running time of FastCDS and competitors. The chosen instances are very easy that all algorithms obtain the same solution quality very quickly.

Benchmark		FastCDS	NuCDS	MSLS	ACO$_e$	ACO	RNS
DIMACS10(22)	#feasible	22	22	16	15	11	12
	#min	19	4	4	3	1	0
	#avg	20	4	2	1	0	0
SNAP(31)	#feasible	31	31	14	15	5	4
	#min	28	5	1	1	1	0
	#avg	29	3	1	1	0	0
NDR(65)	#feasible	65	61	15	18	2	10
	#min	59	10	2	2	1	0
	#avg	60	7	0	0	0	0

Table 3: Summarized results of FastCDS, NuCDS, MSLS, ACO-efficient, ACO-RVNS and RNS-TS on DIMACS10, SNAP and NDR benchmarks.

time (in seconds) when finding the *min* cost in column *time*. FastCDS significantly outperforms all competitors. FastCDS obtains the best solution on 106 instances, and it is the only algorithm that can solve all these 118 instances within the time limit. In addition, we need to point out an error in our previous work (Li et al., 2020a) that we mistakenly took 4 instances which cannot be initialized within the time limit as solvable instances. Among all those solvable instances, the best solution values obtained by FastCDS are on average 2.26%, 2.64%, 3.0%, 2.79% and 79.08% smaller than those found by NuCDS, MSLS, ACO-efficient, ACO-RVNS, and RNS-TS, respectively. The excellent results of FastCDS on massive graph are mainly attributed to the effectiveness of our proposed strategies.

Instance	FastCDS			NuCDS			MSLS		ACO_e	ACO	RNS
	min	avg	time	min	avg	time	min	avg	min	min	min
Amazon0302	47458	47580.1	1.62	**45815**	**45992.6**	3190.61	48225	48625.6	47969	N/A	N/A
Amazon0312	**51096**	**51153.9**	1731.2	52077	52257.8	3522.06	N/A	N/A	N/A	N/A	N/A
Amazon0505	**53029**	**53069.3**	1632.52	54041	54125.7	3582.33	N/A	N/A	N/A	N/A	N/A
Amazon0601	**46978**	**47012.6**	1900.63	47963	48195.4	3275.92	N/A	N/A	N/A	N/A	N/A
Cit-HepPh	**3223**	**3225.1**	3350.13	3263	3270	3508.45	3395	3407.2	3363	3417	31018
Cit-HepTh	**3177**	**3178.8**	1177.47	3197	3204.7	2998.91	3289	3303.3	3264	3320	24036
cit-Patents	**686740**	**686969.8**	86.58	734910	735877.2	2683.82	N/A	N/A	N/A	N/A	N/A
Email-EuAll	**2368**	**2368**	1.07	2371	2371	5.45	**2368**	2368.6	**2368**	2373	224163
p2p-Gnutella04	**2267**	**2267**	0.99	2268	2270	2297.17	2294	2295.6	2270	2279	6526
p2p-Gnutella24	**5468**	**5468**	1.25	5471	5471	10.68	5475	5477.1	5469	5470	23819
p2p-Gnutella25	**4556**	**4556**	0.76	4558	4558	4.18	4561	4562.7	**4556**	4557	19467
p2p-Gnutella30	**7226**	**7226**	5.82	7229	7229	55.23	7238	7240.5	7229	7231	34578
p2p-Gnutella31	**12672**	**12672.1**	134.54	12674	12674.4	267.24	12677	12678.7	12677	12683	61061
Slashdot0811	**14989**	**14989**	1.54	14990	14990	37.81	14993	14994.5	14994	N/A	76148
Slashdot0902	**16159**	**16159**	35.61	16160	16160	138.27	16170	16173.81	16169	N/A	81006
soc-Epinions1	**16667**	**16667**	290.36	**16667**	**16667**	77.64	16668	16668.8	16671	N/A	74677
web-BerkStan	**30887**	**30898**	2266.24	30967	30984.4	3542.31	31308	31332.3	N/A	N/A	N/A
web-Google	**86882**	**86905.4**	3079.25	86963	86992.3	3582.44	N/A	N/A	N/A	N/A	N/A
web-NotreDame	25565	25586.2	2.21	25664	25665.8	824.37	**25406**	**25462**	25686	25641	N/A
web-Stanford	11553	**11557.2**	1888.73	11581	11586.6	2351.61	**11541**	11600.3	11706	11742	N/A
WikiTalk	**35038**	**35038**	8.86	**35038**	**35038**	132.11	N/A	N/A	N/A	N/A	N/A
Wiki-Vote	**1101**	**1101**	0.09	**1101**	**1101**	0.48	**1101**	**1101**	**1101**	**1101**	3093
333SP	**1186502**	**1186858.1**	36.97	1233397	1233545.2	125.82	N/A	N/A	N/A	N/A	N/A
as-22july06	**2059**	**2059**	0.05	**2059**	**2059**	1.67	**2059**	**2059**	**2059**	**2059**	19917
audikw1	**10319**	**10432.7**	2112.62	11092	11146.7	3411.03	N/A	N/A	11864	N/A	N/A
belgium	1207728	1207843.7	26.78	**1201674**	**1201683.8**	20.68	N/A	N/A	N/A	N/A	N/A
cage15	**581885**	**582174.7**	206.46	680979	684575.2	800.59	N/A	N/A	N/A	N/A	N/A
caidaRo*erLevel	**47922**	**47935.1**	2363.89	47994	48002.8	3394.37	48361	48707.8	48422	N/A	N/A
citatio*iteseer	**49494**	**49502.5**	3148.56	49659	49696.8	3286.72	50488	51109	50434	N/A	N/A
cnr-2000	**24748**	**24751.6**	3303.93	24780	24788.9	2374.51	25177	25190.5	24880	N/A	N/A
coAutho*iteseer	**38104**	**38108**	2088.81	38123	38128.9	2630.0	38203	38453.8	38265	N/A	N/A
coAuthorsDBLP	**48774**	**48779.3**	3101.11	48809	48820.5	2482.52	49103	49141.5	48963	N/A	N/A
cond-mat-2005	**5119**	**5119**	21.46	5125	5127	345.32	5165	5169.3	5168	5175	33782
coPaper*iteseer	**34465**	**34494.6**	3443.34	34781	34813.7	3575.36	N/A	N/A	N/A	N/A	N/A
coPapersDBLP	**45745**	**45773.6**	3328.96	46392	46483.9	3546.35	N/A	N/A	N/A	N/A	N/A
ecology1	**358317**	**360050.6**	8.02	400020	403547.3	3599.42	N/A	N/A	N/A	N/A	N/A
eu-2005	**34206**	**34217.7**	3424.09	35146	35157.8	3320.62	N/A	N/A	N/A	N/A	N/A
G_n_pin_pout	**13481**	**13530.8**	713.19	14280	14339.6	3361.24	15124	15154.4	15040	15351	N/A
in-2004	86863	**86872.2**	2944.51	**86847**	86876.4	2247.82	N/A	N/A	N/A	N/A	N/A
kron_g5*-logn16	**3886**	**3886**	2.99	**3886**	3886.1	84.85	3888	3889.2	3887	N/A	53980
ldoor	**33436**	**33494.5**	40.08	34086	34177.7	3571.83	N/A	N/A	N/A	N/A	N/A
luxembourg	101673	101695	3558.5	**101552**	**101562.6**	3577.45	102284	102305.8	101968	N/A	113387
prefere*achment	**8356**	**8359.1**	3479.18	8544	8563.9	3240.6	9100	9116.3	9022	9155	N/A
rgg_n_2_17_s0	**22518**	**22604.3**	346.27	23146	23193.6	3570.98	23631	23662.3	23621	N/A	N/A
rgg_n_2_19_s0	**83791**	**83863**	5.45	84063	84288.9	3585.14	N/A	N/A	N/A	N/A	N/A
rgg_n_2_20_s0	**158867**	**158937.8**	11.69	162289	163454.6	3541.92	N/A	N/A	N/A	N/A	N/A
rgg_n_2_21_s0	**302163**	**302389.6**	24.72	316284	318593.7	3599.47	N/A	N/A	N/A	N/A	N/A
rgg_n_2_22_s0	**576770**	**577023.8**	48.17	620372	623562.6	3580.18	N/A	N/A	N/A	N/A	N/A
rgg_n_2_23_s0	**1103096**	**1103648.3**	110.17	1199201	1199450	140.13	N/A	N/A	N/A	N/A	N/A
rgg_n_2_24_s0	**2115143**	**2115679.2**	238.57	2325926	2326036.3	430.99	N/A	N/A	N/A	N/A	N/A
smallworld	**13785**	**13839.9**	197.25	14355	14481	649.51	15354	15374.6	15609	15646	N/A
uk-2002	**1160524**	**1160612.8**	1708.24	1184688	1184715.3	288.93	N/A	N/A	N/A	N/A	N/A
wave	**15744**	**15861.3**	226.38	16285	16365.8	3582.3	17504	17547.8	17792	N/A	N/A

Table 4: Results of FastCDS, NuCDS, MSLS, ACO-efficient, ACO-RVNS and RNS-TS on DIMACS10 and SNAP benchmarks.

Two reasons accounting for why competitors cannot output feasible solutions on some instances are as follows: 1) The construction processes of MSLS, ACO and ACO-efficient have a complexity of $O(|V|^2)$. They are essential parts of multi-restart and ACO framework, so we reserve them; 2) The complexity of each step is $O(|V|^2)$ in RNS, often trapping in the first iteration. Because $|V|$ of some instances reaches to 10^7, competitors fail to output a feasible solution.

Instance	FastCDS			NuCDS			MSLS	ACO_e	ACO	RNS
	min	avg	time	min	avg	time	min	min	min	min
bn-huma*on_1-bg	**4212**	**4218.3**	3337.39	4417	4511.6	3598.07	4683	4692	N/A	N/A
bn-huma*on_2-bg	**3345**	**3353.4**	3595.88	3454	3487.5	3462.56	3836	3704	N/A	N/A
ca-coau*rs-dblp	**45745**	**45773.6**	3099.34	46392	46493.6	3396.21	N/A	N/A	N/A	N/A
ca-dblp-2012	**50997**	**51003.4**	1399.86	51015	51029.6	2479.22	51072	51193	N/A	316258
ca-holl*od-2009	**51503**	**51599.9**	3598.88	53062	53259.8	3589.33	N/A	N/A	N/A	N/A
channel*00-b050	**580856**	**581204.2**	76.14	653869	654154	879.59	N/A	N/A	N/A	N/A
dbpedia-link	**1533532**	**1533536.7**	3520.33	N/A	N/A	N/A	N/A	N/A	N/A	N/A
delaunay_n22	**1125248**	**1125473**	166.83	1188266	1188358.1	102.31	N/A	N/A	N/A	N/A
delaunay_n23	**2250110**	**2250432.9**	689.43	2368743	2376193.4	2900.29	N/A	N/A	N/A	N/A
delaunay_n24	**4507091**	**4507747.8**	188.48	4866995	4867078.2	158.46	N/A	N/A	N/A	N/A
friendster	**649542**	**649555.4**	1056.58	659802	659919.8	1170.39	N/A	N/A	N/A	N/A
hugebub*s-00020	**11431627**	**11475735.4**	3059.0	12438068	12554344.1	3234.79	N/A	N/A	N/A	N/A
hugetrace-00010	**6510298**	**6510786.9**	284.69	7014767	7119364.1	2857.97	N/A	N/A	N/A	N/A
hugetrace-00020	**8631122**	**8633275.2**	1717.65	9493668	9514712.4	356.83	N/A	N/A	N/A	N/A
inf-europe_osm	43987750	43992418.2	3599.73	**43785238**	**43785337**	3543.52	N/A	N/A	N/A	N/A
inf-germany_osm	9547995	9549133.5	3590.4	**9497412**	**9497590.6**	2771.98	N/A	N/A	N/A	N/A
inf-roadNet-CA	**969870**	**970396.2**	35.4	1015787	1015838.3	13.67	N/A	N/A	N/A	N/A
inf-roadNet-PA	**535031**	**535677.6**	17.75	539827	546456.7	3599.86	N/A	N/A	N/A	N/A
inf-road-usa	**14250176**	**14256927.6**	3594.54	14395692	14395784.3	223.37	N/A	N/A	N/A	N/A
rec-dating	**11739**	**11740.5**	3312.85	11745	11747.1	3306.1	11748	11754	N/A	N/A
rec-epinions	**9067**	**9067.3**	2013.69	9080	9084	2762.39	N/A	9100	N/A	N/A
rec-lib*eti-dir	**12956**	**12958.9**	3354.25	12977	13001.4	3497.2	N/A	13031	N/A	N/A
rgg_n_2_23_s0	**1103096**	**1103648.3**	110.17	1199201	1199450	140.13	N/A	N/A	N/A	N/A
rgg_n_2_24_s0	**2115143**	**2115679.2**	238.57	2325926	2326036.3	430.99	N/A	N/A	N/A	N/A
rt-retw*t-crawl	**82910**	**82916.1**	3446.08	83117	83120	3338.63	N/A	N/A	N/A	N/A
sc-ldoor	**33384**	**33447.5**	40.73	34151	34205.4	3531.43	N/A	N/A	N/A	N/A
sc-msdoor	**14673**	**14728.1**	462.77	15071	15101.8	3432.13	15279	15391	N/A	N/A
sc-pwtk	**8471**	**8512.7**	111.41	8757	8802.1	3451.43	8819	8931	N/A	N/A
sc-rel9	**120681**	**121304.9**	3546.2	124320	125264.9	3599.17	N/A	N/A	N/A	N/A
sc-shipsec1	**10631**	**10729**	607.79	10937	10998.8	2380.66	11791	11954	N/A	13766
sc-shipsec5	**13283**	**13694.4**	3042.98	13811	13921.5	2143.77	14839	15082	N/A	176869
soc-buzznet	**128**	**128**	9.9	**128**	**128**	38.79	**128**	**128**	N/A	99847
soc-delicious	**57662**	**57666.9**	1106.46	57685	57688.3	3037.08	N/A	N/A	N/A	535742
soc-digg	**70626**	**70637.9**	3596.03	70654	70664.9	3596.29	N/A	N/A	N/A	N/A
soc-dogster	**27216**	**27218.4**	3340.64	27284	27291.1	2614.78	27359	27391	N/A	N/A
socfb-A-anon	**204650**	**204656.5**	2767.83	206186	206809.1	3499.2	N/A	N/A	N/A	N/A
socfb-B-anon	**190246**	**190252.2**	124.78	192419	193223.6	2588.29	N/A	N/A	N/A	N/A
socfb-uci-uni	**1237712**	**1237862.4**	3453.03	1542814	16666186.3	3033.06	N/A	N/A	N/A	N/A
soc-flickr	105687	105694.6	23.83	**105659**	**105664.5**	3210.59	N/A	N/A	N/A	513557
soc-flickr-und	**296567**	**296575.6**	169.68	297000	297280	3577.6	N/A	N/A	N/A	N/A
soc-flixster	**91544**	**91544.2**	1807.06	91545	91545.6	47.51	N/A	N/A	N/A	N/A
soc-FourSquare	60980	**60981.7**	3502.39	**60979**	60982	2853.2	N/A	N/A	N/A	638426
soc-lastfm	**67424**	**67424**	5.88	67429	67429	558.08	N/A	N/A	N/A	N/A
soc-livejournal	**842865**	**842897.3**	1295.58	854550	854557.4	152.82	N/A	N/A	N/A	N/A
soc-liv*-groups	**1101690**	**1101710.4**	3595.97	N/A	N/A	N/A	N/A	N/A	N/A	N/A
soc-LiveMocha	**1424**	**1424**	85.51	1425	1426.9	2979.29	1430	1454	1476	103049
soc-ljo*al-2008	**1061607**	**1061692.4**	1962.46	1071978	1071990.7	347.33	N/A	N/A	N/A	N/A
soc-orkut-dir	**94975**	**95380.6**	3576.63	100708	101023.3	3570.36	N/A	N/A	N/A	N/A
soc-orkut	**113264**	**114206.2**	3597.64	120762	120854.1	3583.99	N/A	N/A	N/A	N/A
soc-pokec	**214958**	**215137.4**	3594.25	221800	222244.3	3592.73	N/A	N/A	N/A	N/A
soc-sinaweibo	**201399**	**201399**	510.02	N/A	N/A	N/A	N/A	N/A	N/A	N/A
soc-twi*r-higgs	**14961**	**14965.9**	2689.14	15164	15183.6	3378.36	15335	15375	N/A	N/A
soc-youtube	101716	101723.4	3235.94	**101657**	**101663.7**	3584.01	N/A	N/A	N/A	495497
soc-you*be-snap	235634	235641.4	85.2	**235575**	**235598.8**	2952.14	N/A	N/A	N/A	N/A
tech-as-skitter	**197800**	**197843.5**	3463.31	199864	200270.5	3598.07	N/A	N/A	N/A	N/A
tech-ip	**153**	**153.6**	2615.95	**153**	153.9	2790.7	N/A	170	N/A	N/A
twitter_mpi	**567536**	**567543**	1313.45	N/A	N/A	N/A	N/A	N/A	N/A	N/A
web-arabic-2005	**20646**	**20648.6**	410.96	20663	20670.9	676.52	20737	20730	N/A	162433
web-baidu-baike	**273145**	**273159.2**	428.08	273824	274001.6	3590.9	N/A	N/A	N/A	N/A
web-it-2004	**34548**	**34548.1**	7.15	34549	34549.3	132.54	34551	34557	N/A	N/A
web-uk-2005	**1728**	**1728**	10.62	**1728**	**1728**	1.62	**1728**	**1728**	**1728**	N/A
web-wik*dia2009	**394301**	**394339.1**	288.63	396303	397633.5	3598.67	N/A	N/A	N/A	N/A
web-wik*-growth	**117956**	**118043.8**	3591.71	118964	118976.4	1243.04	N/A	N/A	N/A	N/A
web-wik*ia_link	**190180**	**190191.4**	1050.96	190982	191063.83	2848.3	N/A	N/A	N/A	N/A
wikiped*link_en	**212242**	**212243.1**	150.98	**212242**	212257.7	2807.97	N/A	N/A	N/A	N/A

Table 5: Results of FastCDS, NuCDS, MSLS, ACO-efficient, ACO-RVNS and RNS-TS on NDR benchmarks.

7.4 Effectiveness of Proposed Strategies

As shown in Table 6, six modified versions of FastCDS are proposed to verify the effectiveness of each strategy of our algorithm on massive graphs. We compare FastCDS with FastCDS_1 to show the effectiveness of *chrono-safety*, with FastCDS_2 and FastCDS_3 to show the effectiveness of HDC+. Moreover, we compare FastCDS with FastCDS_4 to show the effectiveness of the novel age and safety combination method, and with FastCDS_5 to show the effectiveness of reconstruction method in HDC+, and with FastCDS_6, where all vertices are in V_{cand}, to show the effectiveness brought by the partition of vertices (i.e., V_{in}, V_{out} and V_{cand}). Meanwhile, when compare with FastCDS_2, which only uses SUB, FastCDS improves the average best solution by 9753.77 on massive graphs. The results of comparisons are shown in Table 7.

In addition, in our previous work, The HDC method of NuCDS improves the average best solution by 17.285% on massive graphs when comparing with its modified version which only uses SUB, while for FastCDS the number is 0.928% when compared with FastCDS_2, mainly because the initial solution of FastCDS is much better than that of NuCDS, which confirms the effectiveness of 1hopReason.

	FastCDS_1	FastCDS_2	FastCDS_3	FastCDS_4	FastCDS_5	FastCDS_6
HDC+	+	-	-	+	-	+
HDC	-	-	-	-	+	-
only-SUB	-	+	-	-	-	-
only-TBC	-	-	+	-	-	-
safety	-	-	-	+	-	-
chrono-safety	-	+	+	-	+	+
Partition	+	+	+	+	+	-

Table 6: Six modified versions of FastCDS, where "+" indicates that the version uses the corresponding strategy while "-" means not.

Benchmark		vs.FastCDS_1	vs.FastCDS_2	vs.FastCDS_3	vs.FastCDS_4	vs.FastCDS_5	vs.FastCDS_6
DIMACS10(22)	#Better	8	11	6	11	19	7
	#Worse	2	2	1	3	0	5
SNAP(31)	#Better	22	26	7	14	27	18
	#Worse	6	2	7	14	2	7
NDR(65)	#Better	32	58	24	43	52	42
	#Worse	21	0	10	13	4	9

Table 7: Comparing FastCDS with 6 modified versions on massive graphs. #Better and #Worse represent respectively the number of instances where FastCDS achieves better and worse results.

7.5 Further Comparison of Different Construction Algorithms

In this subsection, we carry out experiments to compare our construction algorithm, 1hopReason, with 3 construction algorithms, namely 1 hop*, 2 hop* and 1hopNoReason. Specifically, 1hop* and 2hop* denote the improved 1-hop and 2-hop local greedy algorithms (Khuller & Yang, 2019) respectively, which are the state-of-the-art local information greedy

Instance	$\|V\|$	$\|E\|$			1hopReason			1hopNoReason			1hop*	2hop*
			$\|V_{in}\|$	$\|V_{out}\|$	min	avg	time	min	avg	time	min	min
Amazon0302	262111	899792	7371	18283	**48321**	**48378.4**	1.44	48818	48843.5	0.98	61616	67775
Amazon0312	400727	2349869	17981	37365	**54686**	**54722.1**	2.88	56034	56081.1	2.67	72796	77667
Amazon0505	410236	2439437	19307	40088	**56568**	**56624.8**	2.95	57966	58022.5	2.82	75152	77311
Amazon0601	403364	2443311	12541	32057	**50682**	**50712**	3.45	51773	51788.2	2.69	67116	72354
Cit-HepPh	34401	420784	1169	1995	**3433**	**3444.6**	0.41	3498	3509.6	0.4	4510	3949
Cit-HepTh	27400	352021	1316	2039	**3349**	**3358.1**	0.34	3426	3431.3	0.34	4225	3952
cit-Patents	3764117	16511740	407793	709062	**700525**	**700605**	31.13	723239	723345.2	24.42	917079	924107
Email-EuAll	224832	339925	2223	202193	**2371**	**2371.8**	0.39	2373	2374.4	0.36	3175	4641
p2p-Gnutella04	10876	39994	1757	2484	**2295**	**2297.4**	0.04	2416	2429	0.04	3321	2653
p2p-Gnutella24	26498	65359	5147	10994	**5473**	**5475.1**	0.07	5585	5592.9	0.07	8005	6149
p2p-Gnutella25	22663	54693	4273	9324	**4561**	**4563.3**	0.06	4660	4673.8	0.06	6670	5063
p2p-Gnutella30	36646	88303	6888	16499	**7231**	**7231.9**	0.1	7374	7380.4	0.09	10513	8429
p2p-Gnutella31	62561	147878	12252	28806	**12676**	**12677.8**	0.16	12853	12880.9	0.16	18210	14359
Slashdot0811	77360	469180	13832	30164	**14998**	**14999.3**	0.48	15029	15032	0.47	20129	15254
Slashdot0902	82168	504230	14363	30855	**16175**	**16178**	0.54	16220	16223.1	0.49	21230	16530
soc-Epinions1	75877	405739	15936	41046	**16677**	**16677.2**	0.4	16703	16705.7	0.39	22109	16883
web-BerkStan	654782	6581871	12977	145874	**31373**	**31396.7**	19.4	31501	31519.7	6.34	37170	37065
web-Google	855802	4291352	59794	333051	**88026**	**88049.6**	5.7	89048	89078.1	4.43	112411	107650
web-NotreDame	325729	1090108	21780	183085	**25630**	**25654**	1.33	25744	25762.7	1.1	33005	29004
web-Stanford	255265	1941926	5458	50670	**11777**	**11792**	2.94	11818	11831.7	1.86	14063	13813
WikiTalk	2388953	4656682	34281	1842442	**35042**	**35042**	7.16	**35042**	**35042**	5.0	47596	40781
Wiki-Vote	7066	100736	1030	2471	**1101**	**1101.3**	0.09	1104	1104.4	0.09	1519	1106
333SP	3712815	11108633	0	8	1202378	1202555.8	13.96	**1202112**	**1202539.5**	12.52	1367950	1390253
as-22july06	22963	48436	1870	11927	**2059**	**2060.2**	0.06	2062	2063.1	0.05	2859	2346
audikw1	943695	38354076	0	2658	**12147**	**12186**	37.9	12159	12189.6	35.58	15988	19373
belgium	1441295	1549970	211952	52575	1216936	1217055.1	3.09	1217159	1217234.9	2.1	1260193	**1206836**
cage15	5154859	47022346	0	128	586757	**586947.2**	59.72	**586749**	586959.7	52.16	695138	1154163
caidaRo*erLevel	190914	607610	30228	52004	**48669**	**48682.7**	0.73	48977	48997.6	0.68	61017	53648
citatio*iteseer	268495	1156647	31071	49457	**50491**	**50502.9**	1.36	51237	51279.6	1.35	65702	61691
cnr-2000	325557	2738969	19545	126537	**25003**	**25019.8**	4.64	25174	25184.1	2.56	31275	36180
coAutho*iteseer	227320	814134	29867	145237	**38298**	**38306**	0.91	38743	38752.7	0.93	50459	40728
coAuthorsDBLP	299067	977676	38269	184302	**48983**	**48989.9**	1.46	49426	49436.7	1.12	66052	51728
cond-mat-2005	36458	171734	3049	19107	**5200**	**5203.3**	0.17	5262	5267.1	0.17	7033	5543
coPaper*iteseer	434102	16036720	10951	180347	**36014**	**36027**	15.54	36326	36341.1	15.44	45830	39529
coPapersDBLP	540486	15245729	16301	231168	**48218**	**48229.1**	14.9	48770	48785.3	14.22	61921	53475
ecology1	1000000	1998000	0	0	**359585**	**361515.6**	2.43	**359585**	**361515.6**	2.24	454883	424932
eu-2005	862664	16138468	24873	126473	**34782**	**34797.7**	23.7	34911	34932.6	15.21	43921	52841
G_n_pin_pout	99995	501198	53	54	15414	15437.6	0.68	**15404**	**15430.8**	0.51	19280	24187
in-2004	1353703	13126172	66448	521664	**87562**	**87576.6**	19.21	88044	88054.9	12.56	109688	112797
kron_g5*-logn16	55319	2456070	3251	8604	**3888**	**3889.2**	2.45	3897	3899.4	2.3	5223	3896
ldoor	952203	22785136	0	0	**34665**	**34713.8**	22.32	**34665**	**34713.8**	21.15	41290	47529
luxembourg	114599	119666	22426	3948	102257	102287.6	0.15	102248	102302.7	0.14	105146	**101870**
prefere*achment	100000	499985	0	0	**9171**	**9196.3**	0.54	**9171**	**9196.3**	0.51	12441	10122
rgg_n_2_17_s0	131067	728750	25	704	**23657**	**23709.3**	0.77	23688	23716.6	0.72	28144	28649
rgg_n_2_19_s0	524280	3269760	47	1274	84655	84700.7	3.73	**84639**	**84672.9**	3.48	100599	105553
rgg_n_2_20_s0	1048572	6891617	54	1723	**160652**	160803.3	7.34	160688	**160801.9**	6.99	190501	201899
rgg_n_2_21_s0	2097142	14487992	54	2267	306238	**306395.4**	16.22	**306182**	306444.4	15.83	363352	389032
rgg_n_2_22_s0	4194299	30359197	65	2908	585191	**585364.1**	32.47	**585139**	585434.4	34.46	694961	751498
rgg_n_2_23_s0	8388601	63501390	53	246	**1120430**	**1120729.8**	70.24	1120661	1120823.1	68.06	1329443	1449467
rgg_n_2_24_s0	16777215	132557200	68	232	2150059	2150502.1	144.87	**2150017**	**2150374.3**	142.7	2550995	2806078
smallworld	100000	499998	0	0	**15771**	**15821.1**	0.57	**15771**	**15821.1**	0.51	19543	26632
uk-2002	18459128	261556721	773160	3049470	**1175517**	**1175713.3**	263.63	1185305	1185487.3	253.68	1469643	1568781
wave	156317	1059331	0	13	**18060**	**18149**	1.25	18118	18171.3	1.07	23233	29430

Table 8: Results of 1hopReason, 1hopNoReason, 1hop*, 2hop* on DIMACS10 and SNAP benchmarks.

algorithms with the best approximation ratio as far as we know. 1hopNoReason is the modified version of 1hopReason which does not use reasoning rule for construction, meaning that V_{in} and V_{out} are empty sets. All construction algorithms are executed 10 times on each instance. For the sake of fairness, heap is adopted by all competitor algorithms to speed up the construction.

In Tables 8 and 9, $|V|$ and $|E|$ are reported to indicate the size of instances, and the size of V_{in}^G and V_{out}^G are also listed. The best solutions found by each construction method,

Instance	$\|V\|$	$\|E\|$			1hopReason			1hopNoReason			1hop*	2hop*
			$\|V_{in}\|$	$\|V_{out}\|$	min	avg	time	min	avg	time	min	min
bn-huma*on_1-bg	197084	42103873	613	8616	**4797**	**4814.4**	40.26	4847	4856.6	38.66	5976	8144
bn-huma*on_2-bg	160336	22783120	568	9987	**3791**	**3803**	21.5	3850	3860.3	20.92	4781	8155
ca-coau*rs-dblp	540486	15245729	16301	231168	**48218**	**48229.1**	15.69	48770	48785.3	14.43	61921	53475
ca-dblp-2012	317080	1049866	39811	195028	**51209**	**51215.8**	1.46	51689	51705.2	1.22	68844	53974
ca-holl*od-2009	1069126	56306653	20467	642608	**53771**	**53793.9**	60.44	55009	55036.9	54.22	71798	56429
channel*00-b050	4802000	42681372	0	0	**590396**	**590597.3**	58.29	**590396**	**590597.3**	45.96	679238	940553
dbpedia-link	11609793	78613912	1512635	7582394	**1533650**	**1533658.2**	105.66	1534805	1534816.2	98.0	1991671	N/A
delaunay_n22	4194304	12582869	0	47285	1141669	1141798.4	16.77	**1141562**	**1141734.7**	15.47	1361120	1418675
delaunay_n23	8388608	25165784	0	94442	**2283135**	2283952.3	36.56	2283640	**2283943.6**	29.67	2710218	2837288
delaunay_n24	16777216	50331601	0	188843	**4567254**	4567611.7	66.28	4567261	**4567572.8**	63.69	5432219	5675019
friendster	8556203	45604593	592852	4031990	**650511**	**650519.2**	67.41	654727	654769.9	67.37	862952	989617
hugebub*s-00020	21198119	31790179	0	0	**11598117**	**11599292.1**	56.02	**11598117**	**11599292.1**	48.16	12636584	14234612
hugetrace-00010	12057441	18082179	0	0	**6591721**	**6592692.4**	33.97	**6591721**	**6592692.4**	25.86	7185088	8137047
hugetrace-00020	16002413	23998813	0	0	**8744667**	**8744932.1**	36.62	**8744667**	**8744932.1**	32.5	9531885	10773517
inf-europe_osm	50912018	54054660	11007317	2065091	44158122	44159527.7	96.04	44165763	44166335.4	92.25	45556355	**43964376**
inf-germany_osm	11548845	12369181	2375511	660034	9616166	9616649.8	21.03	9618270	9618837.9	17.85	10027719	**9541824**
inf-roadNet-CA	1957027	2760388	326965	321923	**986843**	**987074**	3.83	992660	993239.9	3.54	1168170	1110911
inf-roadNet-PA	1087562	1541514	193743	191185	**544514**	**544612**	2.75	548360	548647	1.99	647236	616481
inf-road-usa	23947347	28854312	6229607	4818270	**14507920**	**14508553.8**	44.61	14537554	14538737.6	42.88	16423355	14893036
rec-dating	168791	17351416	10378	20464	**11748**	**11749.8**	17.57	11778	11781.9	17.38	15410	11779
rec-epinions	755200	13396042	6778	30927	**9084**	**9087.3**	34.6	9117	9118.4	14.13	11904	15651
rec-lib*eti-dir	220970	17233144	10564	20974	**12992**	**12994.3**	17.7	13151	13160.1	18.27	17238	13155
rgg_n_2_23_s0	8388601	63501390	53	246	**1120430**	**1120729.8**	70.24	1120661	1120823.1	68.06	1329443	1449467
rgg_n_2_24_s0	16777215	132557200	68	232	2150059	2150502.1	144.87	**2150017**	**2150374.3**	142.7	2550995	2806078
rt-retw*t-crawl	1112702	2278852	77400	773235	**82956**	**82960.8**	4.11	83019	83025.3	2.8	109538	131970
sc-ldoor	909537	20770807	0	0	**34598**	**34661.2**	20.54	**34598**	**34661.2**	19.41	41374	48090
sc-msdoor	404785	9378650	0	0	**15423**	**15466.9**	9.18	**15423**	**15466.9**	8.72	18327	20760
sc-pwtk	217883	5653217	18	93	8813	8979.7	5.35	**8789**	**8965.4**	5.23	9975	12261
sc-rel9	5921786	23667162	0	2	**158048**	**158529.9**	36.27	**158048**	158530	31.48	220689	405256
sc-shipsec1	139995	1705212	0	62	**12155**	**12192.5**	1.73	12159	12201.8	1.62	14576	21730
sc-shipsec5	178573	2197367	0	295	**15399**	**15439.1**	2.37	15421	15446.4	2.19	18591	26750
soc-buzznet	101163	2763066	39	8097	**132**	**133.8**	3.16	135	136.4	2.57	173	1764
soc-delicious	536108	1365961	53076	309216	**57775**	**57781.1**	1.78	57951	57956.3	1.48	76965	68871
soc-digg	770799	5907132	63693	379778	**70829**	**70837.3**	6.63	71227	71246.4	6.08	90818	87239
soc-dogster	426485	8543321	13253	37257	**27393**	**27399.5**	12.56	27530	27537.6	10.11	34984	29199
socfb-A-anon	3097165	23667394	186402	1783133	**204879**	**204882.1**	30.61	206312	206339.2	27.77	275096	249953
socfb-B-anon	2937612	20959854	178642	1777796	**190413**	**190418.7**	26.53	191257	191275.7	25.4	254291	241333
socfb-uci-uni	58790782	92208195	915979	39475684	**1239451**	**1239543**	136.35	1243296	1243365.2	120.12	1514103	18346561
soc-flickr	513969	3190452	97890	304640	**105784**	**105788**	3.59	106182	106193.4	3.21	136308	107996
soc-flickr-und	1624992	15476835	275157	972024	**296837**	**296842.6**	16.95	297890	297909	17.37	378457	306204
soc-flixster	2523386	7918801	91286	1615516	**91546**	**91546**	10.01	**91546**	**91546**	9.17	124228	113666
soc-FourSquare	639014	3214986	58726	342998	**61016**	**61018.5**	3.75	61303	61316.7	3.26	79552	61349
soc-lastfm	1191805	4519330	66834	604661	**67425**	**67425.3**	5.44	67454	67455.4	5.45	90697	74812
soc-livejournal	4033137	27933062	660655	1213284	**846856**	**846882.8**	40.32	862214	862378.1	32.11	1128679	884565
soc-liv*-groups	7489073	112305407	1039658	3620344	**1101915**	**1101918.4**	158.97	1102911	1102921.7	133.38	1417605	N/A
soc-LiveMocha	104103	2193083	698	7852	**1467**	**1469.7**	2.28	1484	1488	2.06	1986	1489
soc-ljo*al-2008	5363186	49514271	860018	1720984	**1065003**	**1065026.6**	64.84	1080167	1080302.6	60.41	1396969	1106565
soc-orkut-dir	3072441	117185083	40467	87972	**101751**	**101811.2**	124.95	105883	105952.8	122.7	136434	124974
soc-orkut	2997166	106349209	41581	85067	**121367**	**121427.3**	133.63	125751	125867.1	107.87	162252	139548
soc-pokec	1632803	22301964	122070	195110	**220918**	**220937.7**	26.98	228058	228129.6	25.23	300407	239208
soc-sinaweibo	58655820	261321018	201218	27497010	**201399**	**201399**	350.06	201402	201402	273.82	264134	N/A
soc-twi*r-higgs	456293	12508249	6600	29555	**15354**	**15362.5**	17.33	15491	15505.5	13.37	20163	20265
soc-youtube	495957	1936748	89466	199594	**101945**	**101954.9**	2.44	102318	102330.7	2.17	130689	108693
soc-you*be-snap	1134890	2987624	223409	650530	**235911**	**235919.8**	4.17	236319	236327.8	3.54	299557	248405
tech-as-skitter	1694616	11094209	111350	345469	**200977**	**200995.2**	14.75	202533	202571.9	12.11	251664	243606
tech-ip	2250498	21643497	1	46	175	**176.2**	166.44	**173**	**176.2**	21.97	226	177
twitter_mpi	9807184	99904238	537894	5126770	**567900**	**567908.9**	163.61	568435	568444.5	108.52	730280	N/A
web-arabic-2005	163598	1747269	17652	111666	**20785**	**20795.4**	1.88	20850	20858.9	1.64	26850	22773
web-baidu-baike	2107689	16996139	225464	574133	**273548**	**273556.6**	33.82	274845	274853.9	21.34	360969	302009
web-it-2004	509338	7178413	34127	447051	**34550**	**34552**	6.93	34599	34607.1	6.8	47188	34714
web-uk-2005	129632	11744049	1728	127904	**1728**	**1728**	10.63	**1728**	**1728**	10.39	2391	**1728**
web-wik*dia2009	1864433	4507315	314637	799224	**396023**	**396043.1**	6.49	398744	398789.2	5.79	511756	442291
web-wik*-growth	1870521	36532421	70473	128331	**118937**	**118952**	64.11	119924	119938.6	43.72	154510	122836
web-wik*ia_link	2787609	86753367	139648	256867	**190989**	**190997.9**	100.65	192855	192877.2	89.16	244728	210971
wikiped*link_en	3369169	31022996	209132	1379366	**212250**	**212251.1**	34.63	212566	212571.8	32.16	275693	245210

Table 9: Results of 1hopReason, 1hopNoReason, 1hop*, 2hop* on NDR benchmarks.

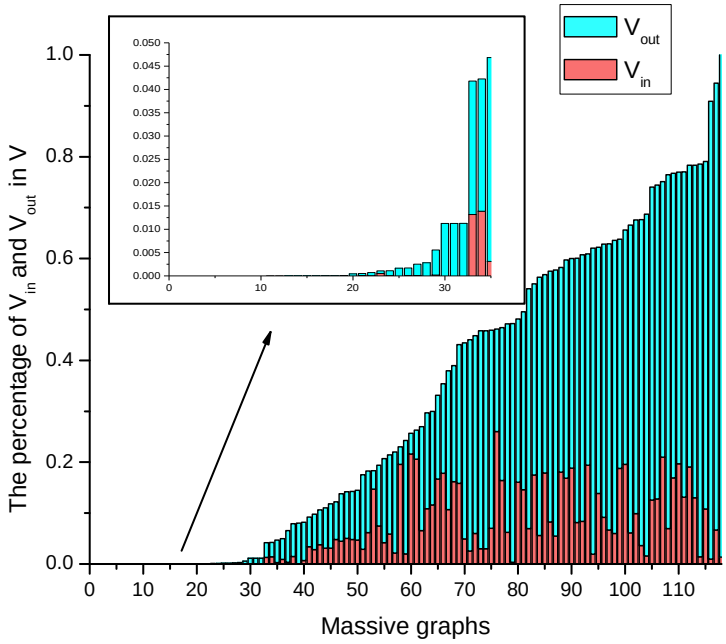


Figure 4: The percentage of the number of V_{in} and V_{out}.

min, and the average size (*avg*) of the solutions found by 1hopReason and 1hopNoReason over 10 runs are reported. We also report the average time (in second) of 1hopReason and 1hopNoReason, *time*, to show the slight time overhead brought by calculating $|V_{in}^G|$ and $|V_{out}^G|$. We report the detailed results on DIMACS10, NDR and SNAP here and the experimental results of all massive graphs are summarized in Table 10.

Moreover, we report the percentage of V_{in} and V_{out} in V on 118 instances in Figure 4. The results are summarized as follows.

- 1hopReason and 1hopNoReason can find better solution than 1hop* and 2hop* on almost all instances, indicating the effectiveness of our greedy construction method for massive graph.

- 1hopReason shows an overwhelming advantage over 1hopNoReason, confirming the effectiveness of the reasoning rule. The size of V_{in} and V_{out} and the comparison of *time* between 1hopReason and 1hopNoReason indicate that a considerable number of fixed vertices can be calculated within a short time for most instances. We observe that the reasoning method does not necessarily work if it meets some instances where only few vertices can be fixed.

- The Figure 4 indicates that the fixed vertices account for a large portion of all the vertices, thus significantly affecting the local search.

Benchmark	1hopReason			1hopNoReason			1hop*		2hop*	
	#min	#avg	time	#min	#avg	time	#min	#avg	#min	#avg
DIMACS10(22)	22	22	3.45	1	1	2.44	0	0	0	0
SNAP(31)	22	24	24.73	11	9	23.14	13	14	2	2
NRD(65)	59	58	46.24	13	14	39.01	0	0	3	3

Table 10: Summarized results of 1hopReason, 1hopNoReason, 1hop*, 2hop* on DIMACS10, SNAP and NDR benchmarks.

Benchmark	avg($\|V\|$)	avg($\|E\|$)	IoU_Pick	IoU_TBC
DIMACS10	473156.09	2099769.55	0.007	0.703
SNAP	2258488.06	22850623.58	0.005	0.489
NDR	6394792.03	32477015.62	0.012	0.433

Table 11: Different IoU on massive benchmarks. $avg(|V|)$ and $avg(|E|)$ denote the average number of vertices and edges in each benchmark.

7.6 Further Analysis to HDC+ Effectiveness

As stated in Section 4.3, the intuition of HDC+ is to take profit from respective advantages of TBC and SUB. Experimental results show that HDC+ is quite effective for massive graphs. We observe that the vertices picked in SUB and TBC significantly differ from each other, which provides an explanation to the success of HDC+ mechanism. In this subsection, extensive experiments are conducted to statistically confirm our observation. FastCDS is executed once and the cutoff time is set to 10000 seconds for the HDC+ analysis.

Given two vertex sets S_1 and S_2, $\frac{S_1 \cap S_2}{S_1 \cup S_2}$ is denoted as intersection-over-union (IoU) (Rezatofighi et al., 2019), which is adopted in Table 11 to analyse the performance of HDC+. Since TBC and SUB have same search space of adding vertices, we only consider the removing vertex set.

For IoU_Pick, S_1 is the picked vertex set of one SUB process, and S_2 is the picked vertex set of its following TBC process. Since the picked vertex set of TBC is much larger than that of SUB, for the sake of comparison, the former set only consists of top k_1 vertices selected in TBC with the the greatest $pick_f$[7] value, where k_1 is the number of vertices chosen in SUB. For IoU_TBC, S_1 and S_2 are the removed vertex sets for two consecutive TBC processes truncated by one SUB process. In Table 11, IoU_Pick and IoU_TBC are exhibited in the form of mean value among the benchmark. Observed from the results, we can obtain that:

- IoU_Pick indicates that vertices picked in SUB significantly differ from those in the following TBC. For further analysis, vertices picked in SUB and TBC are sorted in descending order according to $pick_f$. The top 20 vertices picked by SUB, which are the most frequently chosen ones, rank on average 34110.65 among those vertices picked by TBC. The above result ignores vertices not picked in TBC. For the top 20 vertices picked in SUB with greatest $pick_f$, there are on average 16.42 of them not chosen by

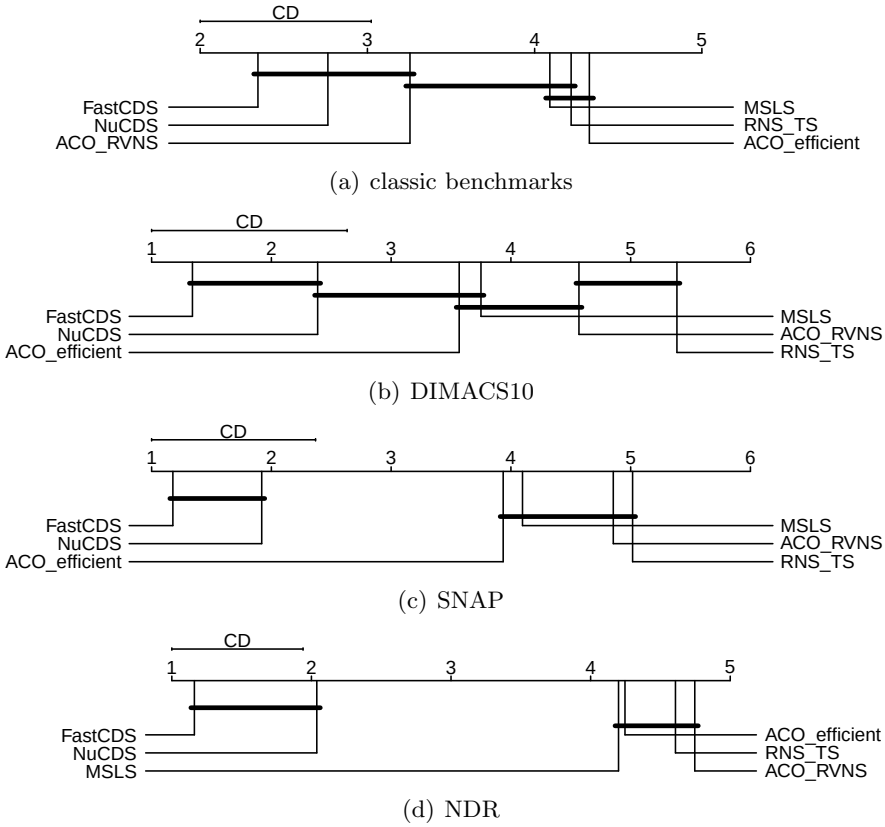
7. $pick_f(v)$ is used to denote the frequency that vertex v is chosen when using one connectivity method.

the following TBC. The results show that SUB and TBC focus on different vertices, resulting in diversifying the search space.

- IoU_TBC indicates that the vertices chosen in two consecutive TBC processes truncated by SUB differ from each other. Moreover, IoU_TBC decreases as the size of input graph grows. The results show that HDC+ mechanism significantly diversifies the search space, particularly when solving massive graphs.

We further analysed the steps of SUB and TBC. The experiment is carried out on DIMACS10, SNAP and NDR with the cutoff time of 3600 seconds. The average execution steps of SUB and TBC are 19652.45 and 43576123.07, respectively. The average number of switching between the SUB and TBC is 91.88. The average run-time for SUB and TBC are 204.39 seconds and 3382.34 seconds, respectively. TBC takes up the main proportion of time. SUB accounts for 0.045% steps, while it takes 5.700% time, which confirms the efficiency of TBC heuristic.

7.7 Critical Difference Analysis



(a) classic benchmarks

(b) DIMACS10

(c) SNAP

(d) NDR

Figure 5: Critical difference plots about FastCDS, NuCDS, MSLS, ACO-efficient, ACO-RVNS, RNS-TS on each benchmark.

In this subsection, we evaluate the statistical differences between the considered algorithms on each benchmark. First, Friedman Test (Friedman, 1937) was conducted with a null-hypothesis that all the algorithms are equivalent in terms of the performance. After the null-hypothesis was rejected, all pairwise comparisons were performed using Nemenyi post-hoc test. Finally, the results are shown in Figure 5 in the form of critical difference diagram (Garcia & Herrera, 2008). Note that we use a package of R called scmamp (Calvo & Santafé, 2016), which can be found at https://github.com/b0rxa/scmamp. The top line in the diagram is the axis on which it plot the average ranks of algorithms, and the lower the ranks, the better the algorithm. The critical difference is shown above each sub-figure and the algorithms that are not significantly different with significant level of 0.05 are connected. From Figure 5, FastCDS outperforms other competitors on each benchmark.

8. Conclusion

We proposed three new algorithmic components for MCDS problem, namely the reasoning based construction, the hybrid dynamic connectivity maintenance heuristic and the two-level vertex selection heuristic. An efficient local search algorithm named FastCDS was developed based on the three components. Extensive experiments were conducted to evaluate the performance of FastCDS. Experiments showed that FastCDS could outperform other state-of-the-art algorithms on almost all the instances, and confirmed the effectiveness of three components.

For future work, we found that our proposed framework designed for solving MCDS problem is not suitable for the weighted version through simple adjustments. Thus, we plan to further study the minimum weight connected dominating set problem, which is a generalization of MCDS problem. Inspired by the success of HDC+, we plan to apply it to other combinatorial optimization problem with connectivity constraint on massive graphs. Additionally, we would like to design a new algorithm for dynamic graphs which support incremental solving, rather than static graphs for the adaption of network applications in real time, such as the application of finding backbone in ever-changing wireless networks.

Acknowledgments

Shaowei Cai and Yiyuan Wang are corresponding authors. We would like to thank the anonymous referees for their helpful comments. This work was supported by Beijing Academy of Artificial Intelligence (BAAI), Youth Innovation Promotion Association, Chinese Academy of Sciences [No. 2017150], NSFC Grant 61806050, and the Fundamental Research Funds for the Central Universities 2412020FZ030.

References

Al-Karaki, J. N., & Kamal, A. E. (2008). Efficient virtual-backbone routing in mobile ad hoc networks. *Computer Networks*, *52*(2), 327–350.

Binkele-Raible, D., & Fernau, H. (2012). An exact exponential-time algorithm for the directed maximum leaf spanning tree problem. *Journal of Discrete Algorithms*, *15*,

43–55.

Bouamama, S., Blum, C., & Fages, J.-G. (2019). An algorithm based on ant colony optimization for the minimum connected dominating set problem. *Applied Soft Computing*, *80*, 672–686.

Cai, S. (2015). Balance between complexity and quality: Local search for minimum vertex cover in massive graphs. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pp. 747–753. AAAI Press.

Cai, S., Hou, W., Wang, Y., Luo, C., & Lin, Q. (2020). Two-goal local search and inference rules for minimum dominating set. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pp. 1467–1473. ijcai.org.

Cai, S., & Su, K. (2011). Local search with configuration checking for SAT. In *IEEE 23rd International Conference on Tools with Artificial Intelligence, ICTAI 2011, Boca Raton, FL, USA, November 7-9, 2011*, pp. 59–66. IEEE Computer Society.

Cai, S., & Su, K. (2013). Local search for boolean satisfiability with configuration checking and subscore. *Artificial Intelligence*, *204*, 75–98.

Calvo, B., & Santafé, G. (2016). scmamp: Statistical comparison of multiple algorithms in multiple problems. *R J.*, *8*(1), 248.

Chen, B., Jamieson, K., Balakrishnan, H., & Morris, R. (2002). Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *Wireless networks*, *8*(5), 481–494.

Chen, S., Ljubić, I., & Raghavan, S. (2010). The regenerator location problem. *Networks: An International Journal*, *55*(3), 205–220.

Cheng, X., Ding, M., Du, D. H., & Jia, X. (2006). Virtual backbone construction in multihop ad hoc wireless networks. *Wireless Communications and Mobile Computing*, *6*(2), 183–190.

Cheng, X., Huang, X., Li, D., Wu, W., & Du, D.-Z. (2003). A polynomial-time approximation scheme for the minimum-connected dominating set in ad hoc wireless networks. *Networks: An International Journal*, *42*(4), 202–208.

Chinnasamy, A., Sivakumar, B., Selvakumari, P., & Suresh, A. (2019). Minimum connected dominating set based rsu allocation for smartcloud vehicles in vanet. *Cluster Computing*, *22*(5), 12795–12804.

Dagdeviren, Z. A., Aydin, D., & Cinsdikici, M. (2017). Two population-based optimization algorithms for minimum weight connected dominating set problem. *Applied Soft Computing*, *59*, 644–658.

Deb, B., Bhatnagar, S., & Nath, B. (2003). Multi-resolution state retrieval in sensor networks. In *Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications, 2003.*, pp. 19–29. IEEE.

Erdem, E., Lin, F., & Schaub, T. (Eds.). (2009). *Proceedings of 10th International Conference on Logic Programming and Nonmonotonic Reasoning*, Vol. 5753 of *Lecture Notes in Computer Science*.

Fan, N., & Watson, J.-P. (2012). Solving the connected dominating set problem and power dominating set problem by integer programming. In *International conference on combinatorial optimization and applications*, pp. 371–383.

Fernau, H., Kneis, J., Kratsch, D., Langer, A., Liedloff, M., Raible, D., & Rossmanith, P. (2011). An exact algorithm for the maximum leaf spanning tree problem. *Theoretical Computer Science*, *412*(45), 6290–6302.

Fomin, F. V., Grandoni, F., & Kratsch, D. (2008). Solving connected dominating set faster than 2 n. *Algorithmica*, *52*(2), 153–166.

Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, *32*(200), 675–701.

Garcia, S., & Herrera, F. (2008). An extension on" statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons.. *Journal of machine learning research*, *9*(12).

Gendron, B., Lucena, A., da Cunha, A. S., & Simonetti, L. (2014). Benders decomposition, branch-and-cut, and hybrid algorithms for the minimum connected dominating set problem. *INFORMS Journal on Computing*, *26*(4), 645–657.

Glover, F., & Laguna, M. (1998). Tabu search. In *Handbook of combinatorial optimization*, pp. 2093–2229. Springer.

Hedar, A.-R., & Ismail, R. (2012). Simulated annealing with stochastic local search for minimum dominating set problem. *International Journal of Machine Learning and Cybernetics*, *3*(2), 97–109.

Hedar, A.-R., Ismail, R., El-Sayed, G. A., & Khayyat, K. M. J. (2019). Two meta-heuristics designed to solve the minimum connected dominating set problem for wireless networks design and management. *Journal of Network and Systems Management*, *27*(3), 647–687.

Hopcroft, J., & Tarjan, R. (1973). Algorithm 447: efficient algorithms for graph manipulation. *Communications of the ACM*, *16*(6), 372–378.

Jovanovic, R., & Tuba, M. (2013). Ant colony optimization algorithm with pheromone correction strategy for the minimum connected dominating set problem.. *Comput. Sci. Inf. Syst.*, *10*(1), 133–149.

Kann, V. (1992). *On the approximability of NP-complete optimization problems*. Ph.D. thesis, Royal Institute of Technology Stockholm.

Khuller, S., & Yang, S. (2019). Revisiting connected dominating sets: An almost optimal local information algorithm. *Algorithmica*, *81*(6), 2592–2605.

Li, B., Zhang, X., Cai, S., Lin, J., Wang, Y., & Blum, C. (2020a). Nucds: An efficient local search algorithm for minimum connected dominating set. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pp. 1503–1510.

Li, J., Wen, X., Wu, M., Liu, F., & Li, S. (2020b). Identification of key nodes and vital edges in aviation network based on minimum connected dominating set. *Physica A: Statistical Mechanics and its Applications*, *541*, 123340.

Li, R., Hu, S., Gao, J., Zhou, Y., Wang, Y., & Yin, M. (2017). Grasp for connected dominating set problems. *Neural Computing and Applications*, *28*(1), 1059–1067.

Li, R., Hu, S., Liu, H., Li, R., Ouyang, D., & Yin, M. (2019). Multi-start local search algorithm for the minimum connected dominating set problems. *Mathematics*, *7*(12), 1173.

Lin, J., Cai, S., Luo, C., & Su, K. (2017). A reduction based method for coloring very large graphs.. In *IJCAI*, pp. 517–523.

Lucena, A., Maculan, N., & Simonetti, L. (2010). Reformulations and solution algorithms for the maximum leaf spanning tree problem. *Computational Management Science*, *7*(3), 289–311.

Milenković, T., Memišević, V., Bonato, A., & Pržulj, N. (2011). Dominating biological networks. *PloS one*, *6*(8), e23016.

Misra, R., & Mandal, C. (2009). Minimum connected dominating set using a collaborative cover heuristic for ad hoc sensor networks. *IEEE Transactions on parallel and distributed systems*, *21*(3), 292–302.

Morgan, M., & Grout, V. (2007). Metaheuristics for wireless network optimisation. In *AICT*, pp. 15–15.

Ni, S.-Y., Tseng, Y.-C., Chen, Y.-S., & Sheu, J.-P. (1999). The broadcast storm problem in a mobile ad hoc network. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pp. 151–162. ACM.

Rezatofighi, H., Tsoi, N., Gwak, J., Sadeghian, A., Reid, I., & Savarese, S. (2019). Generalized intersection over union: A metric and a loss for bounding box regression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 658–666.

Rossi, R. A., & Ahmed, N. K. (2015). The network data repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pp. 4292–4293.

Ruan, L., Du, H., Jia, X., Wu, W., Li, Y., & Ko, K.-I. (2004). A greedy approximation for minimum connected dominating sets. *Theoretical Computer Science*, *329*(1-3), 325–330.

Sen, A., Murthy, S., & Bandyopadhyay, S. (2008). On sparse placement of regenerator nodes in translucent optical network. In *IEEE GLOBECOM 2008-2008 IEEE Global Telecommunications Conference*, pp. 1–6. IEEE.

Simonetti, L., Da Cunha, A. S., & Lucena, A. (2011). The minimum connected dominating set problem: Formulation, valid inequalities and a branch-and-cut algorithm. In *International Conference on Network Optimization*, pp. 162–169. Springer.

Solis-Oba, R., Bonsma, P., & Lowski, S. (2017). A 2-approximation algorithm for finding a spanning tree with maximum number of leaves. *Algorithmica*, *77*(2), 374–388.

Wang, Y., Cai, S., & Yin, M. (2017). Local search for minimum weight dominating set with two-level configuration checking and frequency based scoring function. *Journal of Artificial Intelligence Research*, *58*, 267–295.

Wu, J., & Dai, F. (2003). Broadcasting in ad hoc networks based on self-pruning. *International Journal of Foundations of Computer Science*, *14*(02), 201–221.

Wu, X., Lü, Z., & Galinier, P. (2017). Restricted swap-based neighborhood search for the minimum connected dominating set problem. *Networks*, *69*(2), 222–236.

Yu, J., Wang, N., Wang, G., & Yu, D. (2013). Connected dominating sets in wireless ad hoc and sensor networks–a comprehensive survey. *Computer Communications*, *36*(2), 121–134.