Clustering Driven Iterated Hybrid Search for Vertex Bisection Minimization

Yan Jin, Bowen Xiong, Kun He, Jin-Kao Hao, Chu-Min Li, Zhang-Hua Fu

Clustering Driven Iterated Hybrid Search for Vertex Bisection Minimization

Yan Jin, Bowen Xiong, Kun He, Jin-Kao Hao, Chu-Min Li and Zhang-Hua Fu*

Abstract—The Vertex Bisection Minimization Problem (VBMP) is a relevant graph partitioning model with a variety of practical applications. This work introduces a clustering driven iterated hybrid search algorithm (CLUHS), which is the first approach that applies clustering to reinforce iterated local search for solving VBMP. The proposed CLUHS uses hierarchical clustering to build an initial solution, guide local search process and perform search diversification. Experimental studies on 137 benchmark instances show the high competitiveness of the proposed approach compared to the state-of-the-art methods. In particular, CLUHS finds new record-breaking solutions for 18 instances.

Index Terms—Vertex bisection, Graph partitioning, Clustering, NP-hard, Local search.

1 Introduction

The Vertex Bisection Minimization Problem (VBMP) [1] is to partition the vertices of a simple graph into two disjoint subsets of approximately equal-size in order to minimize the number of vertices in one subset adjacent with at least one vertex in the other subset. VBMP generally belongs to the family of graph partitioning problems [2], [3], [4]. It is also related to graph bisection problem [5], [6], vertex separator problems [7], [8] and the graph layout problem [9]. Practical applications of VBMP include route planning [10], very-large-scale-integration circuit design [11], network communications [12], image processing [13], and distributed computing [14]. In terms of complexity theory, VBMP is known to be NP-hard in the general case, though there are polynomially solvable special cases (e.g., trees and hypercubes) [1].

Due to the theoretical and practical importance of VBMP, a number of algorithms have been proposed for solving this problem. Exact approaches include branch-and-bound algorithms [15], integer linear programming formulation and quadratic programming formulation [16], [17], [18]. These algorithms can find optimal solutions for small graphs, but meet difficulties in solving large problem instances. In such cases, heuristics and metaheuristics are typically used to provide solutions of sufficient quality with reasonable computing efforts. The memetic algorithm of [19] applies four construction heuristics for population initialization, a specific crossover operator for offspring generation, and a local search procedure for solution improvement. The variable neighborhood search heuristic of [20] iterates a solution construction phase (using a simple random method and two greedy randomized adaptive search methods), and an improvement phase by means of six local search strategies based on swap, drop/add and multiple drop/add moves. The cellular processing heuristic of [21] uses a set of independent processing cells to explore different local optima by a limited-effort heuristic and gather useful information, and then uses the acquired information to help the processing cells to jump out local optima traps. The new constructive algorithm of [22] is based on the greedy randomized adaptive search procedure and generates the solutions very quickly, which can be embedded in local search algorithms and memetic algorithms.

Even if important progresses on solution methods for VBMP have been reached in recent years, research in this area can be considered to be quite limited compared to other related graph partitioning problems. Moreover, the performances of existing VBMP algorithms typically vary according to the sizes and types of problem instances. In this work, we aim to advance the state-of-the-art of practically solving VBMP by introducing the first CLUstering driven Iterated Hybrid Search (CLUHS) algorithm, which proves to be highly competitive compared to existing methods. The main contributions of this work are summarized as follows.

First, the proposed CLUHS algorithm is based on the original idea of using clusters within its search procedures. We first introduce a specific similarity metrics to measure the resemblance between two vertices in line with the objective function of VBMP, and then apply a hierarchical clustering algorithm to the vertices of the graph to group vertices with similar properties in the same clusters. After obtaining the clusters, the CLUHS algorithm uses them to guide the construction of its initial solution, constrain its local search process and perform search diversification.

Second, experimental studies on a set of 137 VBMP benchmark instances in the literature show that the proposed algorithm can attain the best known results for all the instances except five ones. In particular, the proposed

- *Y. Jin, B.W. Xiong and K. He are with the School of Computer Science and Technology, Huazhong University of Science and Technology, No. 1037, Luoyu Road, 430074, Wuhan, China. (E-mail: yanjin.china@hotmail.com, jinyan@mail.hust.edu.cn)*
- *J.K. Hao is with the Department of Computer Science, LERIA, University of Angers, 2 Boulevard Lavoisier, 49045 Angers, France and the Institut Universitaire de France, 1 rue Descartes, 75231 Paris, France.*
- *C.M. Li is with the Faculty of Computer Science, MIS, University of Picardie Jules Verne, 33 Rue St. Leu 80039 Amiens Cdex 01, France.*
- ** Corresponding author. Z.H. Fu is with the Institute of Robotics and Intelligent Manufacturing, the Chinese University of Hong Kong, Shenzhen, and the Shenzhen Institute of Artificial Intelligence and Robotics for Society, China.(E-mail: fuzhanghua@cuhk.edu.cn)*

IEEE Transactions on Computers, https://doi.org/10.1109/TC.2021.3128504

algorithm finds improved solutions (new upper bounds) for 18 instances.

The rest of the paper is organized as follows. The next section formally introduces related definitions and notations. Section 3 shows the hierarchical clustering algorithm with a new similarity measure. Section 4 introduces the proposed CLUHS, including the overall framework, cluster-based construction procedure, cluster-constrained local search and cluster-guided perturbation. Section 5 presents computational results and analysis, followed by the conclusion.

2 PRELIMINARIES

Definitions and notations

Given a simple undirected graph $G = (V, E)$ with a set V of n vertices and a set E of m edges, we use $N(v)$ to denote the set of adjacent vertices (also called neighbors) of vertex v, and $N[v]$ to denote $N(v) \cup \{v\}$. The degree of vertex v is then given by $d(v) = |N(v)|$. The *Vertex Bisection Minimization Problem* consists in partitioning V into two disjoint vertex subsets B and B' of equal sizes when $|V|$ is even or the sizes differ by 1 when $|V|$ is odd, i.e., $B \cup B' = V, B \cap B' = \emptyset, |B| = \lfloor |V|/2 \rfloor, |B'| = \lceil |V|/2 \rceil$, such that the cardinality of a subset C ($C \subseteq B$) is minimized, where C contains all the vertices in B that are adjacent to at least one vertex in B', i.e., $\forall v \in C, \exists u \in B' : u \in N(v)$.

Scoring function

A feasible candidate solution S is any partition of the vertex set V into two sets B and B' of approximately equal size. Its quality is given by the scoring (objective) function *score(S)*, which counts the number of vertices in C ($C \subseteq B$) whose vertices are adjacent to at least one vertex in B'. Formally, for a vertex $v \in C$, let $\sigma_B(v)$ denote $|N(v) \cap B|$ and $\sigma_{B'}(v)$ denote $|N(v) \cap B'|$. Then *score(S)* is given by:

$$score(S) = |\{v \in C : \sigma_{B'}(v) > 0\}|. \quad (1)$$

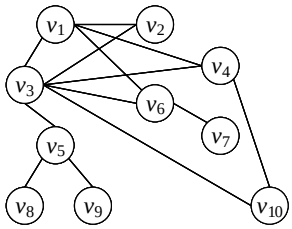


Fig. 1. An example graph with 10 vertices.

Given two candidate solutions S_1 and S_2, S_1 is better than S_2 if and only if $score(S_1) < score(S_2)$. For example, Figure 1 illustrates a simple graph with 10 vertices and 13 edges, while Figure 2 shows its two feasible solutions. Solution (b) is better than solution (a) since the *score* of (b) is smaller than the *score* of (a) ($2 < 3$).

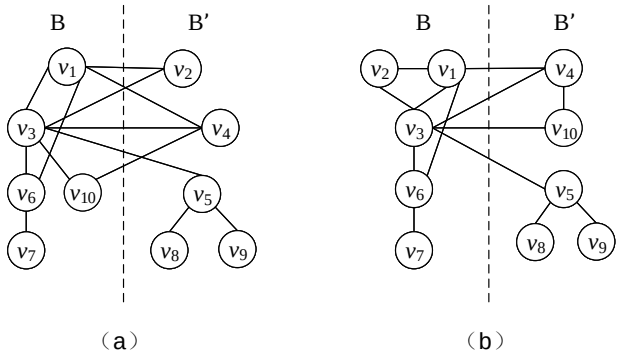


Fig. 2. Two feasible solutions: (b) is better than (a).

3 HIERARCHICAL CLUSTERING WITH SIMILARITY

Hierarchical clustering is an unsupervised machine learning technique which builds, from a set of given objects, a tree-like hierarchy of clusters called dendrogram on the basis of a similarity metric [23], [24]. According to the hierarchy of clusters being formed in a merging (bottom-up) or splitting (top-down) way, hierarchical clustering methods are divided into agglomerative clustering and divisive clustering. For the purpose of this work, we adopt agglomerative clustering to cluster the vertices in a graph.

A clustering algorithm needs a similarity measure to quantify the resemblance of the objects. In the context of solving VBMP, we design the following vertex similarity metric, which relies on the shared neighbors of two vertices. Let $\mathbf{A}$ denote the adjacency matrix of a graph $G = (V, E)$ and $\mathbf{I}$ denote the identity matrix of order $|V| \times |V|$. The similarity matrix $\mathbf{M}$ is calculated by $\mathbf{M} = (\mathbf{A} + \mathbf{I}) \cdot (\mathbf{A} + \mathbf{I})$, i.e., $\forall v_i, v_j \in V, i \neq j, \mathbf{M}_{ij} = |N[v_i] \cap N[v_j]|$. This metric considers that two vertices are more (less) similar if they share more (less) common adjacent vertices. For the studied VBMP problem, similar vertices will be grouped in the same cluster and placed in one subset of a vertex bisection to favor the minimization of the objective function. In Figure 1, the element $\mathbf{M}_{13}$ of the similarity matrix is equal to $|N[v_1] \cap N[v_3]| = 5$.

To reinforce that any two vertices in the same cluster have a high similarity, we use the complete linkage [25] to measure the similarity between two clusters. Note that we apply an agglomerative algorithm to build an incomplete tree, which means the clustering process will stop and return the clustering result Π when the maximum similarity between clusters goes below a threshold τ.

Figure 3 illustrates the hierarchical tree produced by the agglomerative clustering on the graph of Figure 1 based on the above similarity metric. Suppose that the similarity threshold τ is set to 2, an incomplete tree is returned with the clustering result $\Pi = \{\{v_1, v_2, v_3, v_6\}, \{v_4, v_{10}\}, \{v_5, v_8\}, \{v_7\}, \{v_9\}\}$.

4 ITERATED HYBRID SEARCH ALGORITHM

Our *CLUstering driven iterated Hybrid Search* (CLUHS) algorithm relies on a joint use of the popular hierarchical clustering technique and the powerful iterated local search

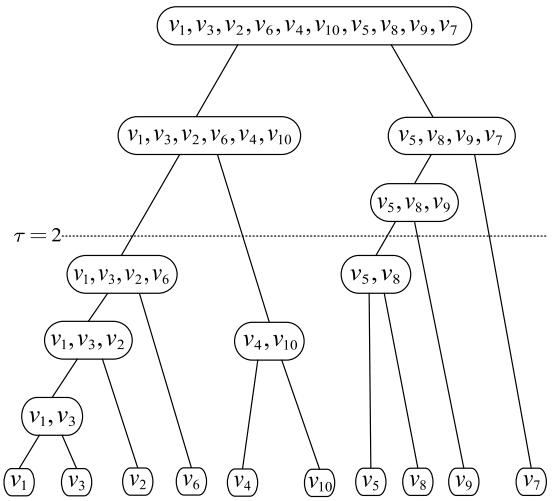


Fig. 3. Hierarchical clustering for the example graph.

Algorithm 1: Scheme of CLUHS for VBMP.

Input: A graph $G = (V, E)$
Output: The best solution found S^*

1 *CLUHS*(G)
2 **begin**
3 $\quad \Pi \leftarrow HClustering(G)$
4 $\quad S_0 \leftarrow Construction(\Pi)$ `// see Section 4.1`
5 $\quad S \leftarrow S_0, S^* \leftarrow S_0$
6 $\quad$ **repeat**
7 $\quad\quad S' \leftarrow CCLS(S)$ `// Cluster- constrained local search, see Section 4.2`
8 $\quad\quad$ **if** S' *is better than* S^* **then**
9 $\quad\quad\quad S^* \leftarrow S'$
10 $\quad\quad S \leftarrow Perturbation(S')$ `// see Section 4.3`
11 $\quad$ **until** $Time > T_{MAX}$
12 $\quad$ **return** S^*

framework. A key feature of our approach is to tackle VBMP with a set of operators guided by vertex clusters. Traditional local search algorithms for VBMP typically perform operations on vertices such as swap, drop and add. However, some vertices could be revealed to be closely related to each other by a hierarchical clustering and this information can be beneficially exploited by the search algorithm. This is the general idea in CLUHS, in which the search operators are guided by the vertex clustering result of the graph.

As shown in Algorithm 1, CLUHS starts with a hierarchical clustering step to obtain a clustering Π typically composed of multiple vertex clusters (line 3). Based on these clusters, CLUHS constructs a feasible solution S_0 composed of two subsets B and B' by merging clusters and adjusting the sizes of the subsets (line 4). Then, starting from S_0, the algorithm performs local optimization to find solutions of better quality (lines 6–11), which repeatedly alternates between the cluster-constrained local search procedure (line 7) and the cluster-guided perturbation procedure (line 10). The algorithm stops when the predefined maximum computation time T_{MAX} is reached and returns the best solution S^* found during the search.

4.1 Cluster-based construction procedure

CLUHS builds an initial solution $S_0 = (B, B')$ based on the clustering result Π provided by hierarchical clustering. The cluster-based construction procedure is composed of two steps: a greedy construction phase to create two initial subsets B and B' and an adjustment phase to establish the feasibility of the solution (see Algorithm 2). The greedy construction phase starts with $B = \emptyset$ and $B' = V$ (line 3). Then, it performs the following operations (lines 4 – 14). Determine a vertex v in B' with the *minimum degree* and further find the cluster Π_a that contains v (lines 5, 6). Move the vertices in cluster Π_a and their adjacent vertices in B' from B' to B. These operations are repeated until B contains $\lfloor n/2 \rfloor$ vertices. This procedure favors grouping vertices of the same cluster to the same subset and helps to minimize the objective function of the initial solution.

If moving the vertices of Π_a to B leads to $|B| > \lfloor n/2 \rfloor$, the solution will become infeasible. In this case, Π_a is not added to B (lines 7, 8), the cluster-based construction procedure finishes the greedy construction phase and switches to the adjustment phase to move vertices from B' to B one by one until a feasible solution S_0 (i.e., with two approximately equal-sized subsets) is reached (lines 15 – 18). The adjustment phase repetitively selects a vertex v with the maximum $\sigma_B(v)$ in B' to optimize the initial solution, and moves v from B' to B until B contains $\lfloor n/2 \rfloor$ vertices. Note that the vertices of some clusters are dispatched into B and B' during the adjustment phase. This cluster-based construction procedure generally provides an initial feasible solution S_0 of good quality that serves as the input for the iterated local search procedure for further improvement. The complexity of the construction procedure is $O(n^2)$.

4.2 Cluster-constrained local search

4.2.1 General working scheme

The key idea of the cluster-constrained local search (CCLS) is to effectively explore the search space of bisections by the cluster constrained *1-move* operator (see the next section). For the obtained initial solution $S_0 = (B, B')$, if all vertices of a cluster Π_a belong to one subset (B or B'), the cluster Π_a has the "locked" status and the cluster Π_a with its vertices will not take part in the following local search. Otherwise, if the vertices of cluster Π_a are dispatched into two subsets, Π_a has the "unlocked" status and its vertices are eligible for *1-move* operations.

The general CCLS procedure is described in Algorithm 3. The search of CCLS is constrained by the clusters and focuses on moving the vertices of "unlocked" clusters. This constraint has an immediate consequence of effectively reducing its search space. Basically, from an initial solution S_0, CCLS finds all the clusters with the "unlocked" status (line 6). It selects a vertex v in the subset B with the best move gain based on the fast evaluation function (see below) and v is not in the tabu list (line 7). Then, CCLS applies the *1-move* operator to displace vertex v from B to B', adds v to the tabu list and updates the related move gains (lines 8–10). At this point, the new solution S' becomes infeasible (unbalanced).

Algorithm 2: Cluster-based construction.

Input: A hierarchical clustering result Π
Output: A feasible initial solution S_0

1 *Construction*(Π)
2 **begin**
3 $B \leftarrow \emptyset$, $B' \leftarrow V$
// Greedy construction phase
4 **while** $|B| < \lfloor n/2 \rfloor$ **do**
5 $v = \arg\min_{v \in B'} d(v)$
6 Find the cluster Π_a that contains v
7 **if** $|\Pi_a \cap B'| + |B| > \lfloor n/2 \rfloor$ **then**
8 break
9 **else**
10 $B \leftarrow B \cup (\Pi_a \cap B')$
11 $B' \leftarrow B' \backslash (\Pi_a \cap B')$
12 **for** *each neighbor* u *of each vertex in* Π_a **do**
13 **if** $u \in B'$ *and* $|B| < \lfloor n/2 \rfloor$ **then**
14 Move u from B' to B
// Adjustment phase
15 **while** $|B| < \lfloor n/2 \rfloor$ **do**
16 $v = \arg\max_{v \in B'} \sigma_B(v)$
17 $B \leftarrow B \cup \{v\}$
18 $B' \leftarrow B' \backslash \{v\}$
19 **return** $S_0 = (B, B')$

Algorithm 3: Cluster-constrained local search.

Input: A solution $S_0 = (B, B')$
Output: A local optimum solution $S^b = (B, B')$

1 *CCLS*(S_0)
2 **begin**
3 $\omega = 0$
4 $S^b \leftarrow S_0$, $S \leftarrow S_0$
5 **while** $\omega < \Psi$ **do**
6 Find all "unlocked" clusters $\{\Pi_i\}$
7 Select a vertex v with the best move gain from clusters $\{\Pi_i\}$ in the subset B and v is not in tabu list T
8 $S' = S \oplus$ *1-move*(S, v)
9 Add v to tabu list T
10 Update move gains // Move gains are recorded in a bucket data structure
11 Select a vertex u with the best move gain from clusters $\{\Pi_i\}$ in subset B' and u is not in tabu list T
12 $S = S' \oplus$ *1-move*(S', u)
13 Add u to tabu list T
14 Update move gains
15 **if** $score(S) < score(S^b)$ **then**
16 $S^b \leftarrow S$
17 $\omega = 0$
18 **else**
19 $\omega = \omega + 1$
20 **return** S^b

Then, CCLS selects a vertex u in the larger subset B' with the best move gain and u is not in the tabu list, applies the *1-move* operator (see below) to move u from B' to B, adds u to the tabu list and updates the related move gains (lines 11–14). The two subsets B and B' are now recovered to be of approximately equal size, and the resulting solution becomes the new current S. Finally, S^b is used to record the best solution found during the CCLS procedure (line 16). CCLS stops and returns the best solution S^b found when S^b cannot be improved for Ψ consecutive solution transitions.

4.2.2 Move operators and neighborhood

CCLS explores candidate solutions of a cluster-constrained search space by making transitions from the current solution to a neighboring solution. Each transition is performed with the *1-move* operator. Formally, let $S = (B, B')$ be the current solution, v be a vertex belonging to a cluster with the "unlocked" status and $X = B$ *or* B'. The *1-move*(X, v) operation displaces vertex v from the subset X to its opposite subset (see examples in Figures 4 and 5). We use $S' = S \oplus$ *1-move*(X, v) to denote the neighboring solution S' obtained by applying *1-move* to S.

Let η be the number of vertices appearing in any cluster with the "unlocked" status and these vertices are not in the tabu list. Then there are η possible neighboring solutions. After a *1-move* operation involving vertex v, the vertex is added to the tabu list T and forbidden to join its original subset during the next tt (randomly selected from {1,2,. . . ,η}) iterations. Note that, CLUHS always performs two consecutive applications of the *1-move* operator to maintain the feasibility of new solutions.

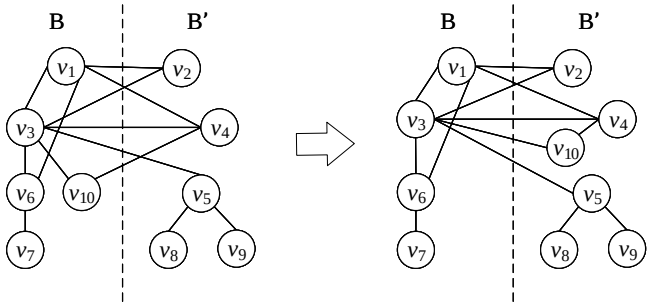


Fig. 4. An example of *1-move*(B, v_{10}) operation.

4.2.3 Fast evaluation function

To ensure the computational efficiency of the algorithm, it is critical to be able to identify quickly the most favorable *1-move* operation for solution transition at each iteration. For this purpose, we adopt a streamlining technique which enables a fast evaluation of the objective function value of a neighboring solution without recalculating it from scratch. This technique is inspired by [6] and based on bucket sorting.

For a *1-move*(X, v) operation applied to $S = (B, B')$, we define the move gain $\Delta(X, v)$ as the decrease in the objective value when S is changed into the neighboring solution S', i.e., $\Delta(X, v) = score(S) - score(S')$, where

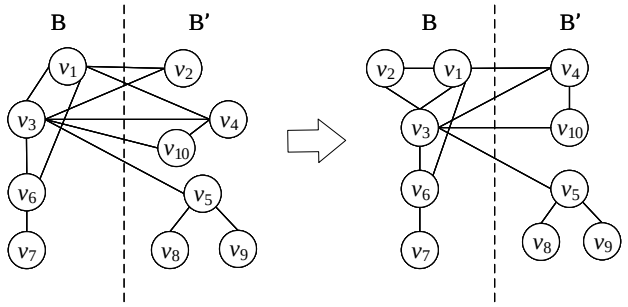


Fig. 5. An example of *1-move*(B', v_2) operation.

$score$ is the scoring function defined in Eq. (1). We illustrate the calculation of the move gain $\Delta(B, v)$ of *1-move*(B, v) as follows ($\Delta(B', v)$ is computed in the same way). $\Delta(B, v)$ is composed of two values: $\delta^+(v)$ and $\delta^-(v)$. The value $\delta^+(v)$ represents the increase in the objective function value if v is moved from B to B', and is obtained in two steps: 1) find all the adjacent vertices $N(v)$ of v in B; 2) calculate $\delta^+(v)$ as the number of vertices u in $N(v) \cap B$ that are not adjacent to any vertex in B' (these vertices will have an adjacent vertex in B' if v is moved into B'). The value $\delta^-(v)$ represents the decrease in objective function value if v is moved from B to B'. If v has adjacent vertices in B', $\delta^-(v)$ is 1 (v will no longer be counted in the objective function if it is moved into B'), otherwise, $\delta^-(v)$ is 0. To sum, $\Delta(B, v)$ is computed as follows.

$$\Delta(B, v) = \delta^-(v) - \delta^+(v) \tag{2}$$

$$\delta^+(v) = |\{u \in N(v) \cap B : \sigma_{B'}(u) = 0\}| \tag{3}$$

$$\delta^-(v) = 1 \quad if \quad \sigma_{B'}(v) > 0; 0 \quad otherwise \tag{4}$$

For example, given the solution S in Figure 4 (left graph), and applying *1-move*(B, v_{10}) to S leads to the neighboring solution shown in the right graph of Figure 4. $\Delta(B, v_{10})$ is given by $\Delta(B, v_{10}) = \delta^-(v_{10}) - \delta^+(v_{10}) = 1 - 0 = 1$, indicating that this *1-move*(B, v_{10}) leads to a neighboring solution of better quality with an objective gain of 1.

Each *1-move*(X, v) operation involves searching for the vertex v with the best move gain, recomputing the move gains for the affected vertices and updating the bucket structure. This structure always keeps the vertices ordered by their move gains in increasing order such that the most favorable vertex can be quickly identified and the unnecessary search for the best move is avoided. Moreover, once *1-move*(X, v) is performed, it is necessary to update σ_B and $\sigma_{B'}$ for each vertex $u \in N(v)$, as well as to update the move gain for each vertex $u' \in N[u]$ when $\sigma_{B'}$ of u changes. The vertex with the best move gain can be found in constant time $O(1)$ and the complexity of recomputing move gains is in linear time $O(n)$.

Another commonly used operator for the graph bisection problems in the literature is the *swap* operator [5], [6], [26], which exchanges a pair of vertices that belong to two different subsets and leads to a neighborhood of size $O(n^2)$. Since the best swap should be chosen from n^2 pairs of vertices in every step, it is more challenging to apply the *swap* operator while keeping fast evaluation of operation gains. However, the potential of the *swap* operator is presumably greater than the *1-move* operator. We believe that applying the cluster guided approach to design a fast evaluation function for the *swap* operator deserves future study.

4.3 Cluster-guided perturbation

The above local search procedure is able to visit different local optimal solutions by avoiding short term cycles with the tabu list. Still the search may get stuck in deep local optima traps. This happens when the best solution S^b cannot be improved for Ψ consecutive iterations. To enable the algorithm to continue its search, CLUHS applies a cluster-guided perturbation procedure to move the search to unexplored regions.

As presented in Algorithm 4, the cluster-guided perturbation first picks an "unlocked" cluster Π_U at random (line 3). Then, it selects randomly a "locked" cluster Π_B from B with probability $\frac{\rho_B}{\rho_B+1}$ and changes its status from "locked" to "unlocked", where ρ_B counts the number of "locked" clusters of B. Similarly, a "locked" cluster $\Pi_{B'}$ from B' is selected randomly and its status is changed to "unlocked" (lines 4 –6). Note that locking clusters allows to intensify the search in more constrained and promising space, but can make the search trapped in local optima. However, perturbation procedure needs to diversify the search, which is why it randomly selects a locked cluster in B and a locked cluster in B and unlocks them. Finally, we group the vertices of Π_U into the same subset (chosen between B and B' at random) and then change the status of Π_U from "unlocked" to "locked" (lines 7–12). To ensure a balanced partition, when a vertex of Π_U is moved from one subset to the other subset (say from B to B'), a vertex from an "unlocked" cluster in B' is randomly selected and moved from B' to B. After the perturbation procedure, CLUHS returns to the CCLS procedure with the perturbed solution as its new starting solution.

5 Experiments

In this section, we carry out experiments to evaluate the proposed CLUHS algorithm on 137 Harwell-Boeing Sparse Matrix Collection instances[1] which are commonly used for testing VBMP algorithms. These instances come from various sources and have quite different structures and scales.

CLUHS was programmed in C++ and compiled by g++ -O3. All the experiments were conducted on a computer with an Intel Core i7-8750H processor (2.2GHz and 32GB RAM) running Ubuntu 18.04. Due to the stochastic characteristic of CLUHS, each instance was independently solved 10 times with different random seeds. The timeout limit T_{MAX} per run was set to 100 seconds for the tested graphs. CLUHS requires two main parameters τ and Ψ. Recall that τ is the similarity threshold used in hierarchical clustering and is set to be the maximum similarity of all vertex pairs $\times$ 0.2, i.e., $max(M_{ij}) \times 0.2, i, j = 1, \ldots, n$. If the obtained τ is smaller than 2, then $\tau = 2$. If it is larger than 5, then $\tau = 5$. The Ψ

1. https://goo.gl/NmX2yq

Algorithm 4: Cluster-guided perturbation.

Input: A solution $S = (B, B')$
Output: A perturbed solution $S' = (B, B')$.
Perturbation(S)
begin
  $\Pi_U \leftarrow$ Randomly select an "unlocked" cluster
  Let ρ_B ($\rho_{B'}$) be the number of "locked" clusters of B (B')
  Randomly select a "locked" cluster Π_B ($\Pi_{B'}$) from B (B') with probability $\frac{\rho_B}{\rho_B+1}$ ($\frac{\rho_{B'}}{\rho_{B'}+1}$)
  Change the status of Π_B and $\Pi_{B'}$ to "unlocked"
  Randomly select a subset X in $\{B, B'\}$ and let Y be the other subset
  for *each vertex* $v \in \Pi_U$ *and* $v \in X$ **do**
    Move v from X to the other subset Y
    Randomly select an "unlocked" cluster Π_Y from Y, and a vertex u from Π_Y
    Move u from Y to X
  Change the status of Π_U to "locked"
  return $S' = (B, B')$

is the maximum number of non-improving moves for CCLS and is set to be 250. In order to identify an appropriate value for a given parameter, we varied its values within a reasonable range ($\tau = max(M_{ij}) \times \alpha$, $\alpha \in [0.1, 0.2, 0.3, 0.4, 0.5]$, $\Psi \in [150, 200, 250, 300]$) and compared their performances. For the experiment on parameter setting, we used a random sample of 20 benchmark instances.

5.1 Comparison with the leading algorithms

To assess the proposed algorithm, we compare CLUHS with the five leading algorithms: CVBP [22], AMAGP [19], MAVBMP [19], CPA-MA [21], and VNS [20], which together hold the best known results for the 137 benchmark instances. CVBP was run on a computer with an Intel Core $i7-7500$ CPU (2.7 GHz and 32GB RAM) and terminated when the number of vertices in B is $\lceil n/2 \rceil$. AMAGP and MAVBMP were performed on a Dual Xeon, 6 core each machine with 24GB RAM and terminated when the number of generations is 200. CPA-MA was implemented in C++ and run on a computer with an Intel Core i5 processor (2.3GHz and 4GB RAM), and used the same timeout limit of 100 seconds as the stopping condition. We ran the VNS code (kindly shared with us by the authors) on our computer with the same timeout limit (100 seconds) used by CLUHS.

Table 1 and 2 present the computational results of CLUHS in comparison with those of the reference algorithms (CVBP, AMAGP, MAVBMP, CPA-MA and VNS) on a total of 137 benchmark instances. The 74 instances of Table 1 have less than 500 vertices, while the 63 instances of Table 2 have more than 500 vertices. Column 2-5 give the best results obtained by CVBP, AMAGP, MAVBMP and CPA-MA respectively. These results are directly extracted from their corresponding papers, CVBP, AMAGP and MAVBMP only report results on a subset of the considered 137 instances. The following columns give the detailed results of VNS (columns 6–10) and CLUHS (columns 11–15): the best scoring (objective) function value $score^*$, the average best scoring function value $Avg.$, the best running time t^* to reach the best objective value $score^*$ (in seconds), the average running time t_{avg} to reach $score^*$ across each of the 10 runs (in seconds) and the successful runs Suc for reaching $score^*$ over the 10 independent runs. The slash '/' symbol indicates that the given reference algorithm did not report results for the given test instance.

Furthermore, we summarize the comparative results between CLUHS and each reference algorithm in Table 3. Column 1 gives the pairs of two compared algorithms, column 2-4 show the number of instances for which CLUHS obtains a better, equal and worse result according to the $score^*$ indicator. The last column gives the *p-values* from the Wilcoxon signed-rank test, which indicates whether there exists a statistically significant difference in performance between CLUHS and the reference algorithms. In addition, considering the different computing platforms of the reference algorithms other than VNS, we halved the timeout limit by 50 seconds and included the comparison numbers in parentheses in Column 2-5.

From Table 1 and 2, one notices that CLUHS obtains best solutions of equal or better quality compared to the five reference algorithms for all the tested instances except 5 ones (in `italic`). Moreover, CLUHS improves the best known results for 6 out of 74 instances with less than 500 vertices, and 12 out of 63 instances with more than 500 vertices (in **bold**). This indicates that CLUHS is particularly suitable for solving large graphs, although it also performs very well on small graphs. Moreover, CLUHS obtains better average results for 47 instances, equal average results for 77 instances and worse average results for 13 ones compared to VNS. Besides, an interesting observation is that CLUHS substantially improves the best solutions of the reference algorithms. This is especially true for mbeacxc, mbeaflw and most improved instances with more than 500 vertices. From Table 3, we can observe that CLUHS performs the best in terms of the objective values ($score^*$). Even if the timeout limit is reduced by half, CLUHS performs better than CVBP, AMAGP, MAVBMP and CPA-MA. From the *p-value*, the statistical test reveals a significant difference in performance between CLUHS and each reference algorithm (p-value ≤ 0.05), which also shows the efficiency of the proposed CLUHS.

This experiment confirms the high competitiveness of our CLUHS algorithm compared to the leading VBMP algorithms. The improved new upper bounds for the 18 instances are valuable references for future research on the problem.

5.2 Analysis of the cluster guided strategy

The proposed CLUHS algorithm specifically exploits vertex cluster information from hierarchical clustering within its construction and search components. In this subsection, we report additional experiments to analyze the impacts of the cluster-based construction and cluster guided search strategy on the performance of CLUHS.

First, we compare the cluster-based construction with the following greedy construction. It starts with $B' = V$ and $B = \emptyset$ and selects a vertex v in B' with the minimum degree to move to B. This operation is repeated until B

TABLE 1
Comparison of CLUHS against the most recent VBMP algorithms on 74 instances with less than 500 vertices.

Instance	CVBP	AMAGP	MAVBMP	CPA -MA	VNS					CLUHS				
					$score^*$	$Avg.$	t^*	t_{avg}	Suc	$score^*$	$Avg.$	t^*	t_{avg}	Suc
494_bus	26	/	/	9	6	6	2.72	23.65	10/10	6	6	3.89	7.90	10/10
arc130	8	15	8	8	8	8	0.02	0.07	10/10	8	8	0.00	0.00	10/10
ash85	9	7	7	7	7	7	0.00	0.01	10/10	7	7	0.00	0.00	10/10
ash292	/	/	/	18	9	9.3	0.08	5.90	7/10	9	**9**	0.03	0.34	10/10
bcspwr01	5	3	3	3	3	3	0.00	0.00	10/10	3	3	0.00	0.00	10/10
bcspwr02	3	2	2	2	2	2	0.00	0.02	10/10	2	2	0.00	0.00	10/10
bcspwr03	8	6	5	5	4	4	0.02	0.42	10/10	4	4	0.01	0.02	10/10
bcspwr04	32	/	/	14	7	7	1.28	20.39	10/10	7	7	0.21	4.16	10/10
bcspwr05	33	/	/	14	7	7	1.50	6.39	10/10	7	7	0.12	1.06	10/10
bcsstk01	12	/	/	12	12	12	0.00	0.01	10/10	12	12	0.00	0.00	10/10
bcsstk04	24	/	/	24	24	24	0.10	0.19	10/10	24	24	0.00	0.01	10/10
bcsstk05	17	/	/	16	16	16	0.06	0.10	10/10	**15**	**15**	0.01	0.05	10/10
bcsstk06	68	/	/	36	36	36	1.25	34.49	10/10	36	36	0.06	2.85	10/10
bcsstk07	/	/	/	36	36	36	1.03	24.66	10/10	36	36	0.08	1.38	10/10
bcsstk20	/	/	/	8	7	7.4	4.80	14.63	9/10	*10*	*14.4*	48.05	57.65	2/10
bcsstk22	6	4	5	4	4	4	0.02	0.05	10/10	4	4	0.01	0.01	10/10
bcsstm07	/	/	/	36	36	37	0.63	48.23	6/10	36	**36**	0.09	1.33	10/10
can__144	6	6	6	6	6	6	0.03	0.11	10/10	6	6	0.00	0.01	10/10
can__161	16	22	24	16	16	16	0.11	2.21	10/10	16	16	0.01	0.06	10/10
can__292	27	/	/	25	18	18	0.25	5.28	10/10	18	18	0.03	0.06	10/10
can__445	53	/	/	44	38	39.6	4.32	18.45	2/10	38	**38**	0.12	2.96	10/10
curtis54	7	6	5	4	4	4	0.00	0.03	10/10	4	4	0.00	0.00	10/10
dwt__209	27	/	/	15	15	15	0.03	5.90	10/10	15	15	0.02	0.06	10/10
dwt__221	7	/	/	8	8	8.2	0.09	28.83	8/10	*8*	**8**	0.01	0.15	10/10
dwt__234	9	5	4	4	4	4	0.01	0.02	10/10	*8*	*8*	0.00	0.00	10/10
dwt__245	21	/	/	11	8	8.7	1.33	0.61	3/10	8	8.7	7.50	5.42	3/10
dwt__310	8	/	/	17	8	8	0.06	0.17	10/10	8	8	0.02	0.03	10/10
dwt__361	14	/	/	14	14	14	0.05	0.14	10/10	14	14	0.03	0.03	10/10
dwt__419	24	/	/	18	16	16	0.25	1.25	10/10	16	16	0.05	0.09	10/10
fs_183_1	/	/	/	20	15	15	0.15	1.06	10/10	15	15	0.08	0.47	10/10
fs_183_3	/	/	/	20	15	15	0.20	0.83	10/10	15	15	0.05	0.26	10/10
fs_183_4	/	/	/	20	15	15	0.17	0.51	10/10	15	15	0.06	0.37	10/10
fs_183_6	/	/	/	20	15	15	0.21	0.59	10/10	15	15	0.05	0.45	10/10
gent113	21	21	19	13	13	13	0.03	0.41	10/10	13	13	0.00	0.01	10/10
gre__115	22	22	20	18	18	18	0.01	0.04	10/10	18	18	0.00	0.01	10/10
gre__185	24	21	20	20	20	20	0.01	0.14	10/10	20	20	0.01	0.01	10/10
gre__343	/	/	/	28	28	28	0.04	0.13	10/10	28	28	0.03	0.04	10/10
gre_216a	/	/	/	21	21	21	0.02	0.05	10/10	21	21	0.01	0.01	10/10
gre_216b	/	/	/	21	21	21	0.03	0.05	10/10	21	21	0.01	0.01	10/10
hor__131	/	/	/	33	33	33.3	0.08	8.38	7/10	33	*33.7*	0.05	7.02	3/10
ibm32	/	/	/	9	9	9	0.00	0.01	10/10	9	9	0.00	0.00	10/10
impcol_a	/	/	/	20	20	20	0.02	10.84	10/10	20	20	0.01	3.48	10/10
impcol_b	19	18	17	15	15	15	0.00	0.01	10/10	15	15	0.00	0.00	10/10
impcol_c	22	29	23	21	21	21	0.03	0.14	10/10	21	21	0.00	0.01	10/10
impcol_d	/	/	/	18	17	17.5	0.15	1.44	5/10	17	**17**	0.03	0.05	10/10
impcol_e	/	/	/	31	30	30	0.03	3.74	10/10	30	30	0.01	0.01	10/10
lns__131	16	14	11	12	11	11	0.06	0.20	10/10	11	11	0.01	0.03	10/10
lund_a	/	/	/	21	20	20	0.08	0.51	10/10	20	20	0.01	0.05	10/10
lund_b	/	/	/	21	20	20	0.21	3.37	10/10	20	20	0.00	0.04	10/10
mbeacxc	/	/	/	210	206	207.3	35.66	34.04	2/10	**187**	**187.1**	4.23	32.44	9/10
mbeaflw	/	/	/	210	206	208.5	45.76	38.87	1/10	**187**	**187**	5.71	28.95	10/10
mbeause	/	/	/	182	199	201.1	16.18	13.05	1/10	**178**	**178**	0.19	3.52	10/10
mcca	/	/	/	20	18	18	0.18	0.22	10/10	18	18	0.01	0.03	10/10
nnc261	/	/	/	12	11	11	0.53	5.74	10/10	11	11	0.02	0.53	10/10
nos1	/	3	3	3	3	3	0.03	0.19	10/10	3	3	0.01	0.08	10/10
nos4	10	8	7	7	7	7	0.00	0.08	10/10	7	7	0.00	0.01	10/10
plat362	44	/	/	27	27	27.1	0.50	13.33	9/10	27	**27**	0.04	0.10	10/10
plskz362	/	/	/	17	10	10	0.47	1.63	10/10	10	10	0.03	0.07	10/10
pores_1	/	/	/	7	7	7	0.00	0.00	10/10	7	7	0.00	0.00	10/10
pores_3	/	/	/	12	12	12	0.77	6.86	10/10	12	12	0.07	0.24	10/10
saylr1	/	/	/	14	14	14	0.03	0.10	10/10	14	*15.2*	0.00	0.00	8/10
steam1	/	/	/	42	39	39	0.16	0.77	10/10	39	39	0.01	0.02	10/10
steam3	4	4	4	4	4	4	0.02	0.06	10/10	4	4	0.00	0.00	10/10
str____0	/	/	/	88	81	84	24.20	23.91	1/10	81	**81.2**	2.59	26.55	8/10
str__200	/	/	/	99	95	96.4	6.93	14.70	1/10	95	**95**	4.66	22.91	10/10
str__600	/	/	/	107	104	104.4	3.19	21.17	6/10	**102**	**102.6**	35.36	48.69	6/10
west0132	26	29	22	19	18	18	0.01	0.31	10/10	18	18	0.00	0.01	10/10
west0156	30	35	35	26	26	26	0.02	0.29	10/10	26	26	0.03	0.09	10/10
west0167	19	25	32	19	19	19	0.01	0.47	10/10	19	19	0.00	0.06	10/10
west0381	/	/	/	116	111	112.2	2.84	11.48	2/10	**110**	**110.5**	3.62	15.99	5/10
west0479	/	/	/	80	75	76.1	6.95	24.39	2/10	75	**75**	0.49	2.56	10/10
west0497	/	/	/	51	44	46.3	56.92	37.56	4/10	44	**44.8**	3.14	15.35	2/10
will57	5	4	4	3	3	3	0.00	0.01	10/10	3	3	0.00	0.00	10/10
will199	62	72	65	54	52	52	0.38	14.42	10/10	52	52	0.01	0.04	10/10

contains $\lfloor n/2 \rfloor$ vertices. For this comparison, we used the 63 large benchmark graphs with more than 500 vertices. The computational results show that the cluster-based construction obtains initial solutions of better and worse quality for 42 and 21 instances respectively compared to the greedy construction, which shows the benefit of the cluster-based construction. However, one observes that there are large gaps between the initial solution and the best solution found by CLUHS. Hence, the following cluster-constrained local search is more critical to the performance of CLUHS.

Then, we investigate the influence of this cluster guided search strategy by comparing CLUHS with a CLUHS variant (denoted as TS) where the main cluster guided components are disabled and replaced by conventional components. Specifically, TS applies, instead of the cluster-constrained *1-move* and cluster-guided perturbation, the classical *1-move* operator for its local search and a random perturbation for diversification, in which the search space is not constrained by the clusters and all vertices in B and B' should be considered at each step. TS shares the other components of CLUHS such as the tabu list and the fast move gain evaluations.

TABLE 2
Comparison of CLUHS against the most recent VBMP algorithms on 63 instances with more than 500 vertices.

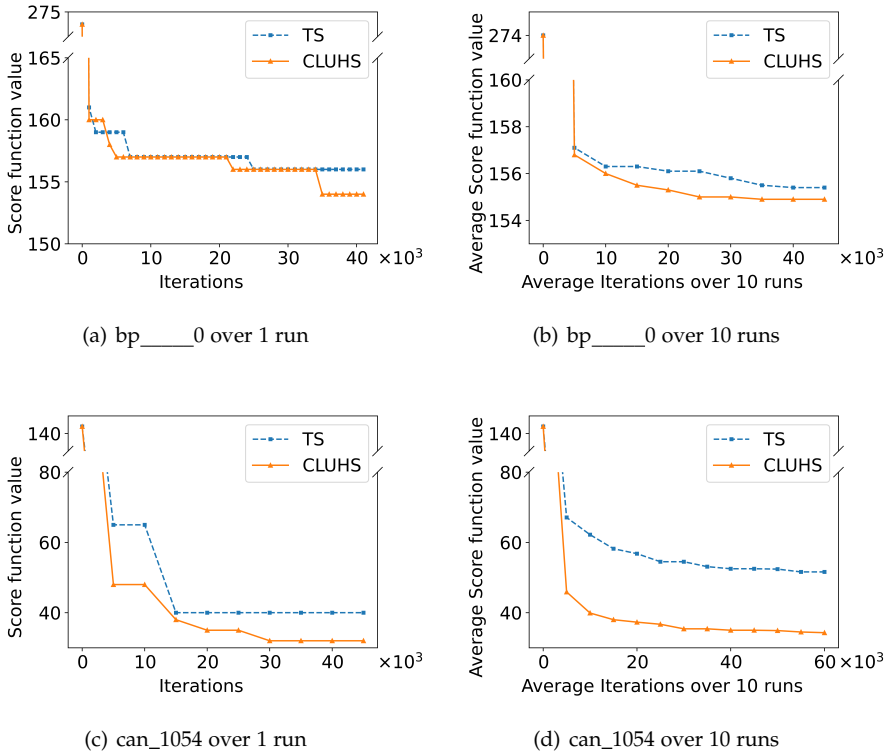
Instance	CVBP	AMAGP	MAVBMP	CPA -MA	VNS					CLUHS				
					$score^*$	$Avg.$	t^*	t_{avg}	Suc	$score^*$	$Avg.$	t^*	t_{avg}	Suc
685_bus	54	/	/	19	8	9.6	7.95	33.73	2/10	8	**8.6**	0.39	10.82	5/10
bcsstk08	/	/	/	70	60	62	5.33	31.24	5/10	60	**61.5**	1.45	5.17	5/10
bcsstk09	/	/	/	73	61	61	7.79	24.69	10/10	61	61	0.57	1.04	10/10
bcsstk11	/	/	/	48	102	138.3	92.09	88.35	1/10	**36**	**36**	1.13	6.09	10/10
bcsstk12	/	/	/	48	103	151.3	82.13	84.84	1/10	**36**	**36**	1.31	8.12	10/10
bcsstk19	/	/	/	4	4	7.1	5.93	35.61	8/10	4	**4**	2.68	20.04	10/10
bcsstk27	/	/	/	41	367	487.1	98.02	95.18	1/10	41	**42.6**	2.16	34.17	8/10
bcsstm27	/	/	/	41	413	490.7	98.71	95.91	1/10	41	**41.8**	2.06	17.03	8/10
blckhole	/	/	/	63	62	62.9	65.20	31.99	1/10	**61**	**61.8**	4.93	23.12	5/10
bp_____0	/	/	/	178	154	156.8	89.94	27.40	1/10	**153**	**153.8**	0.53	37.00	3/10
bp___200	/	/	/	191	184	185.6	10.82	26.78	4/10	**175**	**175.9**	20.71	49.59	2/10
bp___400	/	/	/	208	190	196.1	13.67	15.22	2/10	**184**	**185**	17.19	32.10	1/10
bp___600	/	/	/	211	195	201.5	30.99	31.48	1/10	195	**197**	7.62	28.46	1/10
bp___800	/	/	/	218	202	205.6	78.55	33.87	1/10	**199**	**199.6**	7.04	27.34	4/10
bp__1000	/	/	/	220	207	210	27.19	44.10	1/10	**203**	**204.2**	19.54	31.96	1/10
bp__1200	/	/	/	222	210	216.7	68.42	34.97	1/10	**207**	**208.9**	9.08	23.12	1/10
bp__1400	/	/	/	226	213	218.7	29.55	38.66	1/10	**211**	**211.4**	25.64	35.41	6/10
bp__1600	/	/	/	230	213	217.8	99.78	37.64	1/10	**211**	**211.9**	18.93	38.06	2/10
can__715	/	50	36	36	35	36.9	7.43	34.02	7/10	35	**35.2**	1.03	17.13	9/10
can__838	/	/	/	41	34	35	22.27	28.28	4/10	34	**34.7**	45.26	58.08	3/10
can_1054	/	/	/	45	28	29.4	39.00	72.68	6/10	*32*	*32.5*	11.94	32.12	5/10
can_1072	/	/	/	56	30	37	32.87	62.10	3/10	30	**31**	28.18	49.08	4/10
dwt__503	53	/	/	37	26	32.3	1.92	22.07	4/10	26	**26**	0.11	2.89	10/10
dwt__592	29	/	/	25	22	22.4	1.00	3.28	8/10	22	**22**	0.18	1.79	10/10
dwt__878	/	/	/	22	18	18	0.58	3.67	10/10	18	18	0.34	0.40	10/10
dwt__918	/	/	/	69	22	22	1.26	25.17	10/10	22	22	0.64	6.08	10/10
dwt__992	/	/	/	66	34	34	16.89	43.41	10/10	34	34	0.38	0.46	10/10
dwt_1005	/	/	/	77	33	33	8.82	19.90	10/10	33	33	0.54	6.93	10/10
dwt_2680	/	/	/	70	29	56.7	5.68	62.97	5/10	29	**40.5**	10.12	33.46	7/10
fs_541_1	/	/	/	19	19	19	4.05	7.31	10/10	19	19	0.31	3.10	10/10
fs_541_2	/	/	/	19	19	19	2.15	4.84	10/10	19	19	0.22	2.19	10/10
fs_541_3	/	/	/	19	19	19	1.69	4.69	10/10	19	19	0.54	2.22	10/10
fs_541_4	/	/	/	19	19	19	0.31	5.64	10/10	19	19	0.43	4.60	10/10
fs_680_1	/	/	/	12	6	6	0.33	1.49	10/10	6	6	0.14	0.18	10/10
fs_680_2	/	/	/	12	6	6	0.62	1.42	10/10	6	6	0.17	0.36	10/10
fs_680_3	/	/	/	12	6	6	0.63	1.84	10/10	6	6	0.17	0.25	10/10
fs_760_1	/	/	/	22	22	22	0.19	2.37	10/10	22	22	0.16	0.22	10/10
fs_760_2	/	/	/	22	22	22	0.34	2.61	10/10	22	22	0.17	0.24	10/10
fs_760_3	/	/	/	22	22	22	0.38	2.24	10/10	22	22	0.19	0.32	10/10
gr_30_30	/	/	/	43	30	30	1.61	3.19	10/10	30	30	0.37	0.82	10/10
gre__512	/	/	/	36	36	36	0.07	0.36	10/10	36	36	0.08	0.10	10/10
gre_1107	/	/	/	90	90	96.2	0.32	10.96	3/10	90	**91**	0.89	2.30	5/10
jagmesh1	/	/	/	26	26	26	0.34	0.67	10/10	26	26	0.42	0.53	10/10
jagmesh2	/	/	/	31	31	31	0.42	6.55	10/10	31	31	0.51	0.62	10/10
jagmesh3	/	/	/	33	33	33	0.76	2.79	10/10	33	33	0.64	0.78	10/10
jagmesh7	/	/	/	25	14	14.7	0.83	4.46	3/10	14	*18.4*	0.69	17.93	5/10
jpwh_991	/	/	/	87	63	64	1.64	6.45	5/10	63	*64.8*	1.41	2.26	1/10
lns__511	/	/	/	33	31	31.6	2.36	12.14	4/10	31	*31.9*	3.47	37.04	1/10
lshp1009	/	/	/	31	31	31	0.34	11.01	10/10	31	31	0.52	0.72	10/10
mcfe	/	/	/	91	101	135.7	92.72	71.86	1/10	**89**	**89.2**	3.00	33.89	8/10
nnc666	/	/	/	18	18	18.2	6.91	38.13	8/10	18	**18**	1.79	12.15	10/10
nos2	/	/	/	3	5	6.2	5.55	25.13	5/10	3	**3.4**	7.28	25.63	8/10
nos3	/	/	/	62	40	44.8	28.06	49.26	5/10	40	**40**	0.38	0.58	10/10
nos6	27	/	/	15	15	15	0.18	1.75	10/10	15	15	0.01	0.01	10/10
nos7	/	/	/	66	65	65	0.28	0.84	10/10	65	*67.4*	0.01	0.01	6/10
orsirr_2	/	/	/	67	50	53.2	7.25	26.09	5/10	*51*	**51**	0.20	0.31	10/10
saylr3	/	/	/	31	30	30.5	0.91	0.80	5/10	30	*51.2*	7.09	7.92	1/10
sherman1	/	/	/	30	30	30.7	0.55	0.82	3/10	30	*51.2*	8.30	15.03	1/10
sherman4	/	/	/	24	22	22.8	0.12	1.88	6/10	22	*30.6*	0.75	5.05	4/10
shl____0	/	/	/	102	82	82.1	4.63	22.99	9/10	82	**82**	1.79	24.04	10/10
shl__200	/	/	/	108	90	90.7	5.86	37.88	4/10	90	*91*	4.16	28.88	1/10
steam2	/	/	/	60	60	60	6.47	11.34	10/10	60	60	0.08	0.09	10/10
west0655	/	/	/	115	108	109.9	4.85	16.87	2/10	108	**108.4**	0.15	12.15	8/10

TABLE 3
Summarized Comparisons of CLUHS against each reference algorithm on a total of 137 benchmark instances.

Algorithms	#Better	#Equal	#Worse	*p-value*
CLUHS vs. CVBP	33(33)	9(9)	1(1)	4.4e-7(4.4e-7)
CLUHS vs. AMAGP	17(17)	7(7)	1(1)	6.2e-4(6.2e-4)
CLUHS vs. MAVBMP	14(14)	10(10)	1(1)	6.2e-7(3.9e-3)
CLUHS vs. CPA-MA	77(71)	58(55)	2(11)	3.7e-14(5.5e-8)
CLUHS vs. VNS	21	112	4	1.2e-3

To compare CLUHS with TS, we also used the 63 large benchmark graphs with more than 500 vertices and ran both CLUHS and TS 10 times to solve each instance under the same timeout limit as before (i.e., 100 seconds per run). To ensure a fair comparison, each run of CLUHS and TS was started with an initial solution provided by the construction procedure of Section 4.1. The comparison between TS and CLUHS shows that TS obtains 2 better results, 43 equal results and 18 worse results under the same timeout limit, indicating a clear performance deterioration without the use of clustering information in the local search and perturbation procedures.

Furthermore, Figure 6 shows the evolution profiles of CLUHS and TS on the bp_____0 instance and can_1054 instance: the best scoring (objective) function value vs. the number of iterations over 1 run and the average best scoring function value vs. the average number of iterations over 10 runs. One observes that CLUHS with the cluster guided search strategy converges more quickly toward high quality solutions than TS with the classic *1-move* operator for local search and a random perturbation either over 1 run or over 10 runs. This experiment confirms thus the relevance of the joint use of hierarchical clustering and iterated local search.



(a) bp_____0 over 1 run

(b) bp_____0 over 10 runs

(c) can_1054 over 1 run

(d) can_1054 over 10 runs

Fig. 6. Analysis of the cluster-guided strategy.

5.3 Analysis of Convergence

This subsection provides a convergence analysis of the proposed CLUHS algorithm and tries to explain why some instances are particularly hard for CLUHS. We selected 6 representative instances including 4 instances for which CLUHS failed to achieve the best known results, and 2 instances whose best known results were improved by CLUHS. We ran CLUHS 10 times on each instance with a cutoff time of 100 seconds per run. We collected the best objective value and the current objective values every 100 iterations.

In order to investigate the running behaviors of CLUHS in an intuitive way, we provide the running profiles in Figure 7, where X-axis represents iterations and Y-axis shows the average best objective values and average current objective values over 10 runs along the iterations. One notices that the solution quality of all instances is significantly improved in the initial search iterations and gradually converges to the best solution. For the four failed instances (orsirr_2, dwt_234, bcsstk20, can_1054), we observe that the solution quality on orsirr_2 and dwt_234 fluctuates slightly or even without fluctuations, such that the search cannot escape from the local optima. In addition, we find that the vertices of the largest cluster are more than half of the total vertices and the clustering is very uneven, so the cluster based strategy plays a small role on these two instances. Besides, we observe that the solution quality on bcsstk20 and can_1054 fluctuates greatly, such that the search may skip the region with the best solution. For the two improved solutions (bcsstk11 and bp_____0), the quality of the solution fluctuates appropriately to allow the search to escape many local optima to reach the solutions with better objective values.

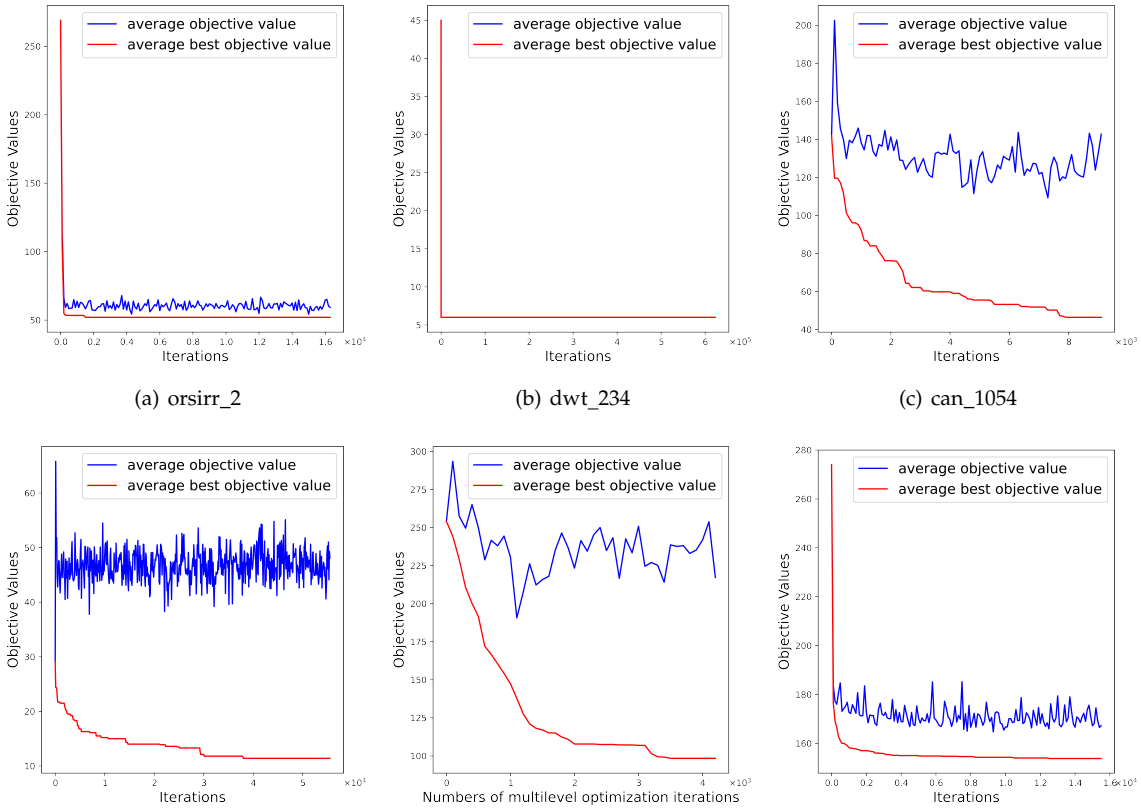
In summary, these convergence profiles indicate the appropriate fluctuates of the solution quality are beneficial for the search and explain to some extent why CLUHS does not perform well on some particular instances.

6 Conclusion

We presented a novel heuristic algorithm that combines hierarchical clustering and iterated local search for solving the NP-hard vertex bisection minimization problem. It uses a specific similarity measure between vertices to create clusters of vertices and employs the clusters within the main search components to generate good initial solutions, explore promising candidate neighboring solutions and perform search diversification.

Computational results on 137 benchmark instances from the literature showed that the proposed CLUHS algorithm is highly competitive compared to the most recent leading algorithms for the problem. In particular, CLUHS achieved all the current best known results except 5 instances and improved the previous best-known solutions for 18 instances. Additionally, we verified the impacts of the cluster-guided search strategy over the performance of the algorithm.

This work focused on designing cluster-guided local search operators. It would be interesting to study other operators with the help of clustering information such as swaps in local search or crossovers in population-based algorithms. Also, to some extend, the underlying idea of using clustering information with local search is of general nature. Given the excellent performance of the proposed



(a) orsirr_2 (b) dwt_234 (c) can_1054

(d) bcsstk20 (e) bcsstk11 (f) bp_____0

Fig. 7. Analysis of convergence charts of CLUHS.

algorithm for vertex bisection minimization, it is worth investigating the usefulness of this approach for solving other related graph partitioning problems.

ACKNOWLEDGMENTS

We are grateful to the anonymous referees for valuable suggestions and comments which have helped us to improve the paper. This work was supported by the National Natural Science Foundation of China (Grants No. 61602196,61972449) and the Shenzhen Science and Technology Innovation Commission under grant JCYJ20180508162601910, the National Key R&D Program of China under Grant 2020YFB1313300.

REFERENCES

[1] U. Brandes and D. Fleischer, "Vertex bisection is hard, too," *Journal Graph Algorithms and Applications*, vol. 13, no. 2, pp. 119–131, 2009.
[2] U. Benlic and J. Hao, "Hybrid metaheuristics for the graph partitioning problem," in *Hybrid Metaheuristics*, ser. Studies in Computational Intelligence, E. Talbi, Ed. Springer, 2013, vol. 434, pp. 157–185.
[3] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz, "Recent advances in graph partitioning," in *Algorithm Engineering - Selected Results and Surveys*, ser. Lecture Notes in Computer Science, L. Kliemann and P. Sanders, Eds., 2016, vol. 9220, pp. 117–158.
[4] T. N. Bui and B. R. Moon, "Genetic algorithm and graph partitioning," *IEEE Transactions on Computers*, vol. 45, no. 7, pp. 841–855, 1996.
[5] G. Lin and W. Zhu, "An efficient memetic algorithm for the max-bisection problem," *IEEE Transactions on Computers*, vol. 63, no. 6, pp. 1365–1376, 2013.
[6] F. Ma, J.-K. Hao, and Y. Wang, "An effective iterated tabu search for the maximum bisection problem," *Computers & Operations Research*, vol. 81, pp. 78–89, 2017.
[7] R. J. Lipton and R. E. Tarjan, "A separator theorem for planar graphs," *SIAM Journal on Applied Mathematics*, vol. 36, no. 2, pp. 177–189, 1979.
[8] U. Benlic and J.-K. Hao, "Breakout local search for the vertex separator problem," in *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013, pp. 461–467.
[9] J. Díaz, J. Petit, and M. Serna, "A survey of graph layout problems," *ACM Computing Surveys (CSUR)*, vol. 34, no. 3, pp. 313–356, 2002.
[10] D. Delling, A. V. Goldberg, T. Pajor, and R. F. Werneck, "Customizable route planning," in *International Symposium on Experimental Algorithms*, 2011, pp. 376–387.
[11] S. N. Bhatt and F. T. Leighton, "A framework for solving vlsi graph layout problems," *Journal of Computer and System Sciences*, vol. 28, no. 2, pp. 300–343, 1984.
[12] R. Klasing, "The relationship between the gossip complexity in vertex-disjoint paths mode and the vertex bisection width," *Discrete Applied Mathematics*, vol. 83, no. 1-3, pp. 229–246, 1998.
[13] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888–905, 2000.
[14] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, 2010, pp. 135–146.
[15] P. Jain, G. Saran, and K. Srivastava, "Branch and bound algorithm for vertex bisection minimization problem," *Advanced Computing and Communication Technologies*, pp. 17–23, 2016.
[16] N. Castillo-García and P. H. Hernández, "Two new integer linear programming formulations for the vertex bisection problem," *Computational Optimization and Applications*, vol. 74, no. 3, pp. 895–918, 2019.
[17] H. Fraire, J. D. Terán-Villanueva, N. C. García, J. J. G. Barbosa, E. R. del Angel, and Y. G. Rojas, "Exact methods for the vertex

bisection problem," in *Recent Advances on Hybrid Approaches for Designing Intelligent Systems*, 2014, pp. 567–577.
[18] P. Jain, G. Saran, and K. Srivastava, "A new integer linear programming and quadratically constrained quadratic programming formulation for vertex bisection minimization problem," *Journal of Automation Mobile Robotics and Intelligent Systems*, vol. 10, pp. 69–73, 2016.
[19] ——, "On minimizing vertex bisection using a memetic algorithm," *Information Sciences*, vol. 369, pp. 765–787, 2016.
[20] A. Herrán, J. M. Colmenar, and A. Duarte, "A variable neighborhood search approach for the vertex bisection problem," *Information Sciences*, vol. 476, pp. 1–18, 2019.
[21] J. D. Terán-Villanueva, H. J. Fraire-Huacuja, S. I. Martínez, L. Cruz-Reyes, J. A. C. Rocha, C. G. Santillán, and J. L. Menchaca, "Cellular processing algorithm for the vertex bisection problem: Detailed analysis and new component design," *Information Sciences*, vol. 478, pp. 62–82, 2019.
[22] N. C. García and P. H. Hernández, "Constructive heuristic for the vertex bisection problem," *Journal of Applied Research and Technology*, vol. 18, no. 4, pp. 187–196, 2020.
[23] F. Murtagh and P. Contreras, "Algorithms for hierarchical clustering: an overview," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 2, no. 1, pp. 86–97, 2012.
[24] A. Saxena, M. Prasad, A. Gupta, N. Bharill, O. P. Patel, A. Tiwari, M. J. Er, W. Ding, and C. Lin, "A review of clustering techniques and developments," *Neurocomputing*, vol. 267, pp. 664–681, 2017.
[25] W. H. Day and H. Edelsbrunner, "Efficient algorithms for agglomerative hierarchical clustering methods," *Journal of Classification*, vol. 1, no. 1, pp. 7–24, 1984.
[26] Q. Wu and J.-K. Hao, "Memetic search for the max-bisection problem," *Computers & Operations Research*, vol. 40, no. 1, pp. 166–179, 2013.

Yan Jin is currently an associate professor in the School of Computer Science and Technology, Huazhong University of Science & Technology (HUST), Wuhan, China. She received the Ph.D. degree in computer science from University of Angers, France, in 2016. Her research interests include reinforcement learning based intelligence computing and metaheuristics for solving large-scale combinatorial optimization problems, including graph coloring problem, maximum clique, packing, scheduling and vehicle routing problems.

Bowen Xiong currently works in NetEase Company, Guangzhou, China. He received his M.S. degree in the School of Computer Science and technology, Huazhong University of Science and Technology (HUST) in 2020, and his B.S degree in the School of philosophy, HUST in 2018. His research interests include combinatorial optimization problem and NP-hard problems such as maximum clique problem, maximal clique enumeration and graph bisection.

Kun He is currently a professor in the School of Computer Science and Technology, HUST, Wuhan, P.R. China, and a Mary Shepard B. Upson Visiting Professor in Engineering, Cornell University, NY, USA. She received her Ph.D. degree in the Department of Automatic Control from HUST in 2006; her B.S degree in the Department of Physics from Wuhan University in 1993, and her M.S. degree in the Department of Computer Science from Huazhong Normal University in 2002, Wuhan, P.R. China. Her research interests include algorithm design and analysis for NP hard problems, social networks and deep learning.

Jin-Kao Hao received the B.S. degree in computer science from the National University of Defense Technology, China, in 1982; the M.S. degree in computer science from the National Institute of Applied Sciences, Lyon, France, in 1987; the Ph.D. degree in constraint programming from the University of Franche-Comté, France, in 1991 and the Professorship Diploma (HDR, Habilitation Diriger des Recherches) from the University of Science and Technology of Montpellier, France, in 1998. Since 1999, he holds a full Professor position with the Computer Science Department at the University of Angers, France. His research lies in the design of effective algorithms and intelligent computational methods for solving large-scale combinatorial search problems. He is interested in various application areas including data science, complex networks, and transportation. He has authored or co-authored more than 250 peer-reviewed publications and co-edited 9 books in Springers LNCS series. He has served as an Invited Member of more than 200 Program Committees of International Conferences and is on the Editorial Board of 7 International Journals. Dr. Hao became a Distinguished Professor in 2010 and Senior Fellow of the Institut Universitaire de France since 2015.

Chu-Min Li received the B.S. and Ph.D. degrees in computer science from the University of Technology of Compiegne, France, in 1985 and 1990, respectively. He is currently a Professor of computer science with the University of Picardie Jules Verne. His research interests include the practical resolution of NP-hard problems, including SAT, CSP, MaxSAT, MinSAT, MaxClique, and GCP. He is particularly interested in the intrinsical relationships between these problems. One of his research directions is to find and exploit these relationships to solve them. A recent example is the exploitation of the relationships between MaxSAT and MaxClique to solve MaxClique.

Zhang-Hua Fu was born in Jiangxi, China, in 1984. He received the B.S. degree in communication engineering, the M.S. and Ph.D. degrees in computer science from the Huazhong University of Science and Technology, China, in 2005, 2007, and 2011, respectively. From 2012 to 2015, he was a Postdoctoral Researcher with the LERIA Laboratory, University of Angers, France. He is currently a Research Fellow with the Chinese University of Hong Kong, Shenzhen, China. He has published more than 20 international journal or conference papers. His research interests include combinational optimization, graph theory, operations research, artificial intelligence, and multiagent systems. He was the winner of several competition tracks of the 11th DIMACS Implementation Challenge in Collaboration with ICERM: Steiner tree problems (Providence, USA, 2014).